

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :

l'Institut National Polytechnique de Toulouse (INP Toulouse)

Présentée et soutenue le *25/09/2015* **par :**

THIBAUT PROBST

ÉVALUATION ET ANALYSE DES MÉCANISMES DE SÉCURITÉ DES RÉSEAUX DANS LES INFRASTRUCTURES VIRTUELLES DE CLOUD COMPUTING

JURY

HERVÉ DEBAR	Telecom SudParis	Rapporteur
ERIC TOTEL	Supélec Rennes	Rapporteur
JEAN-CHRISTOPHE CHOUARD	Atos	Examinateur
VINCENT NICOMETTE	INSA Toulouse - LAAS-CNRS	Examinateur
JEAN-PHILIPPE WARY	Orange	Examinateur
MOHAMED KAÂNICHE	LAAS-CNRS	Directeur de Thèse
ERIC ALATA	INSA Toulouse - LAAS-CNRS	Directeur de Thèse

École doctorale et spécialité :

MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de Recherche :

Laboratoire d'Analyse et d'Architecture des Systèmes

Directeur(s) de Thèse :

Mohamed Kaâniche et *Eric Alata*

Rapporteurs :

Hervé Debar et *Eric Totel*

À mes parents

Remerciements

Je remercie Jean Arlat, Directeur du Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du CNRS dans lequel ces travaux de thèse ont été effectués, pour m'avoir accueilli dans ce laboratoire. Je remercie également Karama Kanoun et Mohamed Kaâniche, en tant que responsables successifs de l'équipe Tolérance aux fautes et Sûreté de Fonctionnement informatique (TSF) au sein de laquelle j'ai travaillé dans d'excellentes conditions durant ces trois dernières années.

Je tiens à exprimer une profonde gratitude à mes encadrants, Mohamed Kaâniche, Directeur de Recherche au CNRS, Eric Alata, Maître de Conférences à l'INSA de Toulouse, et Vincent Nicomette, Professeur des Universités à l'INSA de Toulouse. Je les remercie d'abord de m'avoir offert l'opportunité de faire une thèse de doctorat sur un sujet très intéressant et dans un environnement très favorable. Ensuite, je tiens à souligner leurs qualités humaines, qui ont permis une très bonne entente entre nous, ce qui est important dans le déroulement d'une thèse. Bien sûr, je ne peux omettre de rappeler leurs compétences scientifiques et techniques, qu'ils ont chacun mis au service de ces travaux de manière complémentaire. J'avais déjà eu la chance de connaître Eric et Vincent en tant qu'enseignants lors de mon cursus d'ingénieur à l'INSA. Le privilège que j'ai eu de travailler avec eux et Mohamed en tant que chercheurs au LAAS m'a apporté encore plus de connaissances ainsi que d'autres méthodes de travail.

J'ai une pensée particulière pour Yves Deswarte, Directeur de Recherche au CNRS et membre de l'équipe TSF, dont la disparition en janvier 2014 nous a tous affecté. Il fut un précurseur dans la recherche en sécurité informatique et son intérêt envers ma thèse m'a permis d'avoir le privilège de travailler avec lui. Ses conseils ont d'ailleurs été déterminants pour mes travaux.

J'adresse également mes sincères remerciements aux membres du jury qui ont accepté de juger mon travail :

- Hervé Debar, Professeur des Universités à Telecom SudParis ;
- Eric Totel, Professeur des Universités à Supélec Rennes ;
- Jean-Christophe Chouard, Ingénieur à Atos ;
- Vincent Nicomette, Professeur des Universités à l'INSA de Toulouse ;
- Jean-Philippe Wary, Ingénieur à Orange ;
- Mohamed Kaâniche, Directeur de Recherche au LAAS-CNRS ;
- Eric Alata, Maître de Conférences à l'INSA de Toulouse.

Je suis très reconnaissant pour l'intêret qu'ils ont porté à mes travaux. Je suis ravi et honoré que M. Debar et M. Totel aient accepté d'être rapporteurs de ma thèse.

Je remercie également toutes les personnes du LAAS avec qui j'ai partagé de très bons moments durant ces trois années. Merci à tous les membres de l'équipe TSF, et notamment les doctorants et post-doctorants pour ces pauses cafés, parties de Tarot et Durak et autres soirées très agréables. Je ne peux oublier de remercier Kalou Cabrera Castillos, *de facto* relecteur officiel des manuscrits de thèse des doctorants l'équipe, qui, par sa perspicacité, a grandement contribué à améliorer la qualité de ce document. Je

tiens également à remercier Sonia Vierge et Caroline Malé, qui ont été successivement secrétaires de l'équipe. Elles m'ont très souvent facilité des démarches administratives assez fastidieuses.

Mes remerciement s'adressent aussi à toutes les personnes qui se sont intéressées à mes travaux. Je pense notamment aux partenaires industriels du projet dans lequel ma thèse s'inscrit, avec qui les échanges ont été très intéressants. À leurs côtés, j'ai notamment renforcé mes compétences en gestion de projet.

Je tiens aussi à exprimer ma gratitude à l'ensemble du personnel du Département Génie Electrique et Informatique (DGEI) de l'INSA de Toulouse, qui m'ont fait confiance et donné l'opportunité de pratiquer l'enseignement durant ces trois années de thèse. Je remercie particulèrement Christophe Chassot, Vincent Nicomette, Eric Alata, Ernesto Exposito et Thierry Monteil.

J'ai également une pensée pour tous mes amis qui se sont intéressés à mes travaux et m'ont questionné, conseillé ou simplement écouté. Pouvoir partager mon expérience et prendre du recul sur mon travail avec eux de temps à autre fut important et constructif pour moi, notamment car la plupart travaillent dans le même domaine que moi.

Enfin, mes pensées vont vers ma famille. Mes parents m'ont toujours soutenu dans mes choix et permis d'avancer dans d'excellentes conditions. Je sais aujourd'hui que sans eux, je n'aurais jamais fait cette thèse et ce manuscrit n'existerait donc pas. Ma sœur Virginie et mon beau-frère Armel m'ont apporté un soutien décisif et une réelle écoute dans plusieurs épreuves, aidant à garder le cap. Pour conclure, j'adresse un remerciement tout particulier à Audrey, qui sait déjà la place essentielle qu'elle a dans cette aventure fantastique.

Table des matières

Introduction générale **1**

1 Contexte et problématique **5**
1.1 Le cloud computing 5
1.1.1 Définitions et caractéristiques 6
1.1.2 Modèles de service et de déploiement 7
1.1.3 Technologies associées 8
1.2 Problématiques de sécurité 12
1.2.1 La sûreté de fonctionnement 12
1.2.2 La sécurité 14
1.2.3 Menaces, vulnérabilités et attaques dans le cloud 15
1.3 Mécanismes de sécurité réseau dans le cloud 20
1.3.1 Pare-feu virtuels 20
1.3.2 Systèmes de détection d'intrusion 21
1.4 Analyse et évaluation de la sécurité 26
1.5 Conclusion 28

2 État de l'art **31**
2.1 Analyse d'accessibilité réseau 31
2.1.1 Analyse statique 31
2.1.2 Analyse dynamique 33
2.1.3 Comparaison 34
2.2 Évaluation des systèmes de détection d'intrusion 34
2.2.1 Évaluation par description et analyse 35
2.2.2 Évaluation expérimentale 36
2.3 Conclusion et objectifs de la thèse 43

3 Présentation générale de l'approche **45**
3.1 Hypothèses principales 46
3.1.1 Environnement 46
3.1.2 Contraintes et cas d'utilisation 46
3.1.3 Conformité 47
3.2 Principes 48
3.2.1 Clonage de l'infrastructure virtuelle 49
3.2.2 Analyse des contrôles d'accès réseau 56
3.2.3 Évaluation des systèmes de détection d'intrusion réseau 57
3.3 Conclusion 57

4 Analyse des contrôles d'accès réseau 59
4.1 Accessibilités . . . 59
4.2 Analyse statique . . . 61
4.2.1 Parcours des configurations . . . 61
4.2.2 Résolution logique des accessibilités . . . 62
4.3 Analyse dynamique . . . 66
4.3.1 Principes . . . 67
4.3.2 Complexité . . . 69
4.4 Analyse des déviances . . . 70
4.5 Conclusion . . . 71

5 Évaluation des systèmes de détection d'intrusion réseau 73
5.1 Trafic d'évaluation . . . 73
5.1.1 Modélisation sous forme d'automates . . . 74
5.1.2 Génération des automates . . . 75
5.1.3 Rejeu des automates . . . 76
5.2 Campagnes d'attaque . . . 80
5.2.1 Dictionnaire d'attaques . . . 80
5.2.2 Sessions d'attaque . . . 81
5.2.3 Exécution des campagnes d'attaque . . . 82
5.2.4 Calcul des métriques d'évaluation . . . 84
5.2.5 Analyse des incohérences de détection . . . 85
5.3 Conclusion . . . 85

6 Prototype et expérimentations 87
6.1 Plateforme de maquettage et d'expérimentation . . . 87
6.2 Prototype . . . 89
6.2.1 Architecture logicielle . . . 89
6.2.2 Intégration . . . 94
6.3 Expérimentations et résultats . . . 94
6.3.1 Scénario expérimental . . . 94
6.3.2 Résultats expérimentaux . . . 98
6.4 Conclusion . . . 104

Conclusion générale 107

Bibliographie 111

Table des figures

1.1 Modèle visuel du NIST pour le cloud computing (extrait de [40]) 6
1.2 Types d'hyperviseurs 10
1.3 Intérêt des VXLAN pour la migration (extrait de [3]) 11
1.4 Arbre de la sûreté de fonctionnement 12
1.5 Pare-feu virtuels dans le cloud 22
1.6 Techniques de détection d'intrusion (extrait de [120]) 22
1.7 Niveaux de déploiement des sondes d'IDS/IPS dans le cloud 24

2.1 Exemples de courbes ROC pour différents IDS (extrait de [80]) 42

3.1 Cas d'utilisation 47
3.2 Processus global de l'approche d'évaluation et d'analyse 48
3.3 Vue hiérarchique des composants d'un cloud 50
3.4 Types de réseaux virtuels 52
3.5 Processus d'importation des machines virtuelles 54

4.1 Intéractions entre les modules d'analyse statique 62
4.2 Organigramme de l'algorithme de résolution des accessibilités 65

5.1 Exemple d'automate modélisant les échanges réseau d'un usage légitime (en vert) et d'un usage malveillant (en rouge) d'un serveur FTP 75
5.2 Génération d'un automate 77

6.1 Infrastructure physique et virtuelle de la plateforme d'expérimentation . 89
6.2 Interactions entre les composants du prototype 91
6.3 Scénario déployé sur la plateforme d'expérimentation 95
6.4 Règles de filtrage du pare-feu en mode pont edge-1 95
6.5 Règles de filtrage du pare-feu en mode pont edge-2 96
6.6 Règles de filtrage du pare-feu en mode pont edge-3 96
6.7 Règles de filtrage du pare-feu en mode pont edge-4 96
6.8 Règles de filtrage du pare-feu en mode hyperviseur pour le réseau inet1 97
6.9 Règles de filtrage du pare-feu en mode hyperviseur pour le réseau inet2 97
6.10 Règles de filtrage du pare-feu en mode hyperviseur pour le réseau inet3 97
6.11 Règles de filtrage du pare-feu en mode hyperviseur pour le réseau inet4 98
6.12 Règles de filtrage du pare-feu en mode hyperviseur pour le réseau externe 98
6.13 Graphe d'accessibilité généré par l'analyse statique 99
6.14 Graphe d'accessibilité généré par l'analyse dynamique 99
6.15 Taux de détection des NIDS 103
6.16 Précision des NIDS 103

Liste des tableaux

1.1 Classification des méthodes pour la sécurité 16
1.2 Rôles dans le cloud 17
1.3 Classification d'attaques par vecteur d'attaque 19
1.4 Classes d'incidents dans le cloud 20
1.5 Avantages et inconvénients des niveaux de déploiement des sondes d'IDS/IPS dans le cloud 25
1.6 Caractéristiques de fonctionnement d'IDS pour le cloud 26

2.1 Avantages et inconvénients des types d'analyse d'accessibilité réseau . . 34
2.2 Matrice de confusion d'IDS 40

3.1 Informations de configuration requises pour le clonage d'un centre de données virtuel 51

4.1 Exemple de matrice d'accessibilité 60
4.2 Descriptions des prédicats des configurations 63
4.3 Exemples des prédicats des configurations 64

5.1 Attributs du dictionnaire d'attaques 80

6.1 Matrice d'accessibilité définie pour le scénario expérimental 94
6.2 Entrées du dictionnaire d'attaques et nombre d'états des automates . . 101
6.3 Résultats des campagnes d'attaque 102

Introduction générale

Ce manuscrit rapporte des travaux de thèse menés au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du CNRS à Toulouse, dans l'équipe Tolérance aux fautes et Sûreté de Fonctionnement informatique (TSF). Il traite de la thématique de l'évaluation et de l'analyse des mécanismes de sécurité réseau dans les infrastructures virtuelles de cloud computing. Cette thèse a été effectuée dans le cadre du projet français Investissements d'Avenir *Secured Virtual Cloud* (SVC), regroupant deux partenaires académiques, dont le LAAS, et sept partenaires industriels.

Contexte

Le réseau Internet est en évolution constante, comme en témoignent les chiffres associés [11]. L'intensification de son utilisation est possible grâce à l'augmentation continue des capacités de calcul, de stockage et de débit. Dans ce contexte, la notion de service informatique s'est également fortement développée ces dernières années, que ce soit pour les particuliers ou les professionnels. Afin d'assurer ces services, les systèmes répartis sont aujourd'hui prédominants. Le paradigme du cloud computing est depuis quelques années en train de prendre une place centrale pour la délivrance et la consommation de services, comme le montrent plusieurs études [14, 18]. Le cloud computing permet à des utilisateurs de consommer des ressources informatiques en tant que services de différentes natures, avec différents niveaux de contrôle sur les technologies utilisées. Les ressources proposées comprennent des infrastructures, des plateformes de développement et d'exécution ou des applications. Elles sont généralement hébergées chez un fournisseur de services. Un des objectifs principaux du modèle cloud est de permettre aux clients de réduire les coûts de déploiement et d'opération de ressources qui étaient hébergées traditionnellement dans leurs locaux. Le modèle de service cloud *Infrastructure as a Service* (IaaS) permet la création et la gestion d'infrastructures virtualisées, rassemblant des centres de données virtuels composés de machines virtuelles, réseaux virtuels et espaces de stockage virtuels. Il est ainsi possible de créer ou d'exporter tout un système d'information dans le cloud.

Problématique

Le cloud computing, comme tout système informatique réparti, est continuellement exposé à de nombreuses menaces aux origines diverses [5]. Ainsi, la sécurité du cloud est aujourd'hui une préoccupation très importante des fournisseurs et utilisateurs [14, 18]. Pour se prémunir des attaques reposant sur l'utilisation des réseaux, des mécanismes de sécurité réseau sont déployés pour protéger les données hébergées dans les infrastructures virtuelles. Les pare-feu sont responsables du filtrage de paquets afin de contrôler l'accès réseau. Les systèmes de détection d'intrusion sont en charge de détecter les

attaques survenant sur les canaux de communication. L'objectif des administrateurs sécurité (des clients ou des fournisseurs) est de prévenir et de détecter les attaques tout en ne perturbant pas le bon fonctionnement du cloud. Rendre les pare-feu et systèmes de détection d'intrusion efficaces conjointement n'est pas une tâche aisée. En effet, les produits déployés doivent être maintenus à jour, correctement configurés et positionnés sur le réseau. De plus, les environnements cloud évoluent constamment au cours du temps. Les clients existants peuvent ajouter ou supprimer des instances de machines virtuelles et des réseaux ou modifier des configurations ; de nouveaux clients peuvent souscrire à des services et créer de nouvelles infrastructures ; les fournisseurs administrent également certains composants du cloud. Cette dynamicité peut avoir des impacts négatifs sur la sécurité réseau dans le cloud. Par conséquent, il est important de surveiller et d'analyser régulièrement le niveau de sécurité réseau des infrastructures cloud, afin d'adapter et d'améliorer le déploiement des outils de sécurité réseau. Nous avons donc identifié le besoin de concevoir des méthodes pour faciliter les tâches d'analyse et d'évaluation de la sécurité des infrastructures cloud. Nous pensons que de telles méthodes doivent s'inscrire dans un processus d'audit automatisé et dont la durée est minimisée, de manière à convenir à un environnement dynamique fondé sur les services. À ce jour, ce problème reste ouvert car il n'existe pas, à notre connaissance, de solutions satisfaisantes pour l'évaluation et l'analyse automatisées de la sécurité réseau dans le cloud.

Objectifs

Ces travaux de thèse portent sur la conception de méthodes et le développement d'outils pour l'audit des mécanismes de sécurité réseau dans les infrastructures virtuelles de cloud computing. Nous nous intéressons aux deux mécanismes mentionnés précedemment : le contrôle d'accès réseau et la détection d'intrusion réseau.

Pour être capable d'analyser et d'évaluer ces mécanismes, il faut pouvoir notamment étudier comment ils réagissent vis-à-vis de l'injection de trafic spécifique comme des attaques réseau. L'injection d'un tel trafic peut fortement perturber le réseau et les hôtes des infrastructures des clients. En effet, l'exécution d'attaques vers ces hôtes peut mettre en danger le fonctionnement des applications déployées qui en sont la cible. D'autre part, certaines applications nécessitent un fonctionnement continu, et les audits pouvant avoir une durée significative, il n'est pas envisageable d'arrêter des services réseau pour mener ces audits. La solution que nous avons donc retenue consiste à mener les opérations d'audit sur une copie de l'infrastructure dont la sécurité réseau est à évaluer.

Une fois la copie de l'infrastructure réalisée, nous pouvons mener les opérations d'audit réseau. Cependant, pour évaluer les systèmes de détection d'intrusion, il serait trop fastidieux et coûteux de lancer des attaques sur le réseau de façon désordonnée entre n'importe quels hôtes et sur tous les services réseau possibles. Il est ainsi essentiel d'organiser les attaques réseau de façon pertinente, en cherchant d'abord les canaux de

communication autorisés pour ensuite y exécuter des attaques. Nous procédons donc d'abord à une analyse des contrôles d'accès réseau afin de déterminer les accessibilités réseau. Cette analyse est menée de deux manières, statiquement et dynamiquement, afin d'identifier d'éventuelles déviances dans l'application des contrôles d'accès.

À partir des accessibilités réseau trouvées, l'établissement et l'exécution de campagnes d'attaque nécessite une démarche méthodique. Il existe de nombreuses vulnérabilités et beaucoup d'exploits associés, ciblant différentes applications. Par ailleurs, il est important de mélanger le trafic malveillant à du trafic légitime, de façon à observer les réactions des systèmes évalués par rapport à plusieurs types de trafic. Nous avons donc cherché à modéliser et rejouer certains comportements réseau d'applications vulnérables. Il s'agit des comportements liés aux activités malveillantes et légitimes vis-à-vis desquelles les systèmes de détection d'intrusion sont évalués. Puis, nous avons optimisé au mieux l'exécution d'attaques et de requêtes légitimes ciblant ces applications, en fonction des accessibilités trouvées afin de ne génerer que des flux autorisés par les pare-feu.

Ainsi, nous proposons dans cette thèse une approche pour automatiser l'évaluation et l'analyse des mécanismes de sécurité réseau dans les infrastructures virtuelles de cloud computing. Cette approche vise à analyser les contrôles d'accès réseau d'une infrastructure virtuelle donnée, et à évaluer l'efficacité des systèmes de détection d'intrusion dans la surveillance de cette infrastructure. Elle est constituée de trois phases, et tire profit des bénéfices du cloud pour automatiser le processus d'audit de bout en bout. Des rapports d'évaluation et d'analyse sont fournis afin de mettre en évidence des déviances, incohérences ou faiblesses dans le déploiement des outils de sécurité. Cette thèse a donc donné lieu à quatre contributions principales associées à des environnements virtuels de type cloud :

- une méthode d'analyse statique automatisée des contrôles d'accès réseau [121],
- une méthode d'analyse dynamique automatisée des contrôles d'accès réseau [121],
- une méthode d'exécution de campagnes d'attaque automatisée pour l'évaluation des systèmes de détection d'intrusion réseau [122],
- un prototype expérimental mettant en œuvre les trois méthodes précédentes.

Plan de la thèse

Le début du manuscrit est dédié à la présentation du contexte des travaux et de l'état de l'art associé. Ainsi, le Chapitre 1 introduit le contexte scientifique de ces travaux de thèse, axé autour du cloud computing, de la sécurité, et de l'évaluation et l'analyse de la sécurité. Nous y énonçons également la problématique à laquelle notre approche apporte une réponse. Le Chapitre 2 présente l'état de l'art associé à nos travaux. Il se concentre sur les domaines de l'analyse d'accessibilité et de l'évaluation des systèmes de détection d'intrusion.

La cœur du manuscrit présente en détails les contributions de la thèse. Ainsi, le Chapitre 3 donne une vue d'ensemble de l'approche que nous proposons pour l'évalua-

tion et l'analyse des mécanismes de sécurité réseau dans les infrastructures virtuelles de cloud computing. Nous y détaillons en particulier la première phase de l'approche sur le clonage, car c'est une étape préalable indispensable aux phases d'audit. Le Chapitre 4 décrit nos méthodes d'analyse statique et dynamique des contrôles d'accès réseau, ainsi que l'analyse des déviances dans les résultats. Le Chapitre 5 explique comment nous évaluons les systèmes de détection d'intrusion réseau. Nous y expliquons d'abord comment nous avons modélisé, généré et rejoué le trafic d'évaluation. Puis, nous détaillons l'utilisation de celui-ci à travers l'exécution de campagnes d'attaque. Enfin, nous précisons comment nous étudions la réaction des systèmes de détection d'intrusion pour calculer des métriques d'évaluation.

La fin du manuscrit, à travers le Chapitre 6, présente la plateforme de maquettage et d'expérimentation utilisée dans le cadre des travaux afin de développer et tester les outils développés. Ces derniers constituent un prototype que nous détaillons. Enfin, nous décrivons nos expérimentations, puis exposons et discutons les résultats obtenus.

La conclusion générale conclut la thèse et présente les perspectives de recherche futures à partir de notre travail.

Chapitre 1

Contexte et problématique

Sommaire

1.1 Le cloud computing 5
1.1.1 Définitions et caractéristiques 6
1.1.2 Modèles de service et de déploiement 7
1.1.3 Technologies associées 8
1.2 Problématiques de sécurité 12
1.2.1 La sûreté de fonctionnement 12
1.2.2 La sécurité 14
1.2.3 Menaces, vulnérabilités et attaques dans le cloud 15
1.3 Mécanismes de sécurité réseau dans le cloud 20
1.3.1 Pare-feu virtuels 20
1.3.2 Systèmes de détection d'intrusion 21
1.4 Analyse et évaluation de la sécurité 26
1.5 Conclusion 28

Ce chapitre introduit le contexte scientifique et technologique dans lequel s'inscrivent les travaux de cette thèse. L'objectif est de comprendre les enjeux et problématiques auxquels nos contributions proposent de répondre par la suite. D'abord, nous présentons les deux domaines majeurs associés à nos travaux : le cloud computing et la sécurité informatique. Puis, nous nous intéressons à l'association de ces deux domaines en décrivant les mécanismes de sécurité réseau dans le cloud que nous considérons dans la problématique. Enfin, nous introduisons les thèmes de l'analyse et de l'évaluation de la sécurité, qui sont traités plus en détail dans nos contributions.

1.1 Le cloud computing

Le cloud computing est un paradigme émergeant dans le monde de l'informatique, né de l'évolution des systèmes distribués et des grilles informatiques. Cependant, bien qu'il ait plusieurs similarités avec ces derniers, le cloud computing se différencie par certaines caractéristiques et modèles de service et de déploiement. Ainsi, nous exposons d'abord quelques définitions, puis décrivons les caractéristiques, modèles de service et de déploiement spécifiques à ce paradigme. Nous présentons également quelques nouvelles technologies associées qui rendent possible le modèle cloud.

1.1.1 Définitions et caractéristiques

Le terme anglais *cloud computing*, ou plus simplement *cloud*, est également couramment employé en langue française. Plusieurs traductions, peu utilisées, ont été tentées : informatique en nuage, informatique nuagique, ou encore infonuagique. Selon le *National Institute for Standards and Technology* (NIST), le cloud computing est *un modèle pour permettre l'accès réseau ubiquitaire, facile et à la demande à un ensemble partagé de ressources informatiques (réseaux, serveurs, stockage, applications et services) configurables, qui peuvent être rapidement provisionnées et libérées avec un mininum d'efforts de gestion ou d'intéraction avec le fournisseur du service* [106]. La définition d'*informatique en nuage* fournie par le dictionnaire français Larousse [90] est répertoriée en tant qu'expression pour le mot *nuage* : *modèle d'organisation informatique permettant l'accès à des ressources numériques dont le stockage est externalisé sur plusieurs serveurs*. En informatique, l'origine de l'utilisation du terme anglais *cloud*, ou nuage en français, viendrait de l'utilisation historique de nuages pour représenter les réseaux informatiques, et notamment Internet. Cette représentation a pour but de masquer la complexité interne des réseaux et d'en abstraire une représentation simple. Elle a donc été reprise pour masquer la complexité des divers éléments composant le cloud, et le peu d'information sur la localisation et le traitement des données hébergées dans le cloud.

Au delà des définitions proposées, le NIST a défini un modèle de cloud computing [106] comportant cinq caractéristiques essentielles, trois modèles de service et quatre modèles de déploiement, comme présenté dans la Figure 1.1. Ce modèle de

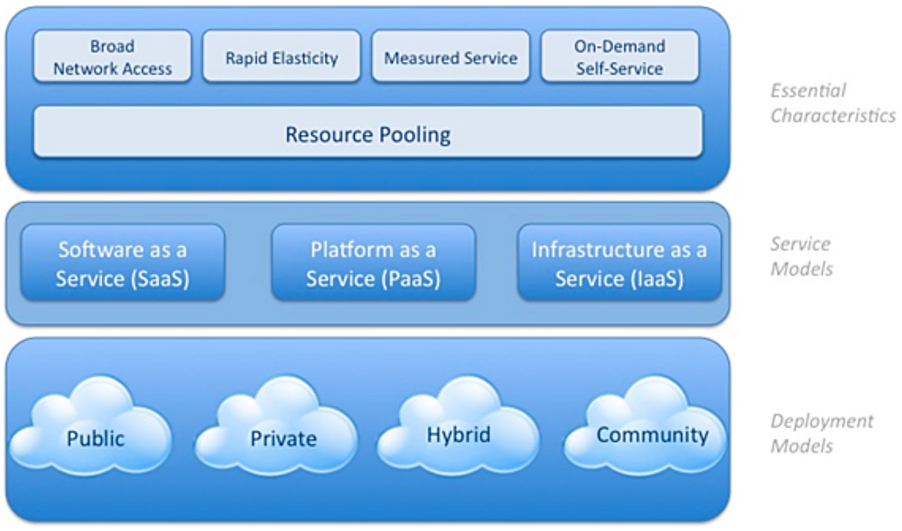


FIGURE 1.1 – Modèle visuel du NIST pour le cloud computing (extrait de [40])

cloud possède les cinq caractéristiques essentielles suivantes :

- Libre-service à la demande : l'utilisation des services et ressources est entièrement automatisée et c'est l'utilisateur, au moyen d'une console de commande, qui met en place et en gère la configuration à distance.
- Large accès réseau : les services sont accessibles depuis le réseau Internet par des équipements traditionnels et hétérogènes, légers ou lourds.
- Mise en commun de ressources : les ressources informatiques du fournisseur sont regroupées et utilisées pour servir plusieurs clients selon un modèle de co-résidence, avec des ressources dynamiquement allouées selon les demandes. Généralement, le client ne connait pas la localisation exacte des ressources qui lui sont fournies, bien qu'il puisse être en mesure de spécifier cette localisation à certains niveaux plus abstraits (comme le pays, la région, ou le centre de données).
- Elasticité rapide : les ressources peuvent être provisionnées et libérées de manière élastique et automatique, pour répondre rapidement et de manière extensible à la demande. Pour le client, les ressources apparaissent illimitées et peuvent être affectées en n'importe quelle quantité à n'importe quel moment.
- Facturation à l'usage : les systèmes de cloud contrôlent automatiquement et optimisent l'utilisation des ressources par des moyens de mesure à certains niveaux d'abstraction appropriés au type de service. L'utilisation des ressources peut être surveillée, contrôlée, et rapportée, afin d'offrir de la transparence pour le fournisseur et le client du service.

1.1.2 Modèles de service et de déploiement

Une infrastructure cloud est la collection de matériels et logiciels rendant possible les cinq caractéristiques essentielles du cloud computing. Elle peut être vue comme contenant une couche physique et une couche d'abstraction. La couche physique comprend les ressources matérielles nécessaires au support de services cloud fournis, et inclut serveurs, stockage et composants réseau. Conceptuellement, la couche d'abstraction se situe au-dessus de la couche physique. Elle comprend les logiciels déployés sur la couche physique, qui présentent les caractéristiques essentielles du cloud et permettent au fournisseur d'assurer différents types de service. Sur cette base, il existe différents modèles de service et de déploiement qui permettent de mettre en œuvre des services sur une infrastructure cloud.

Le cloud possède trois modèles de services, qui définissent le type de service offert :

- *Software as a Service* (SaaS) : le client peut utiliser les applications du fournisseur qui s'exécutent sur une infrastructure cloud. Les applications sont accessibles par une interface légère telle qu'un navigateur Web ou une interface de programme. Le client ne gère pas l'infrastructure cloud sous-jacente incluant réseau, serveurs, systèmes d'exploitation, stockage, ou même les fonctions de l'application à l'exception de paramètres de configuration utilisateur limités.
- *Platform as a Service* (PaaS) : le client peut déployer des applications sur une infrastructure cloud en utilisant les langages, librairies, services et outils suppor-

tés par le fournisseur. Le client ne gère pas l'infrastructure cloud sous-jacente incluant réseau, serveurs, systèmes d'exploitation, ou stockage, mais contrôle les applications déployées et potentiellement les paramètres de configuration de l'environnement hébergeant les applications.

– *Infrastructure as a Service* (IaaS) : le client peut provisionner du traitement, du stockage, des réseaux, et d'autres ressources informatiques fondamentales grâce auxquelles le client peut déployer et exécuter n'importe quel type de logiciels, pouvant inclure des systèmes d'exploitation et des applications. Le client ne gère pas l'infrastructure cloud sous-jacente mais contrôle les systèmes d'exploitation, le stockage et les applications déployées, et contrôle potentiellement de façon limitée certains composants réseau.

De plus en plus de nouveaux modèles de service non définis initialement par le NIST apparaissent suivant les acronymes *XaaS*, où *X* représente le service offert, comme par exemple : *Storage*, *Network*, *Security*... Ces services sont alors généralement inclus dans les trois modèles de service de référence.

Le cloud possède quatre modèles de déploiement, qui définissent comment est gérée l'infrastructure cloud sur laquelle sont déployés les services.

– Cloud privé : l'infrastructure cloud est utilisée exclusivement par une seule organisation. Elle peut être gérée par l'organisation elle-même, une tierce partie, ou une combinaison des deux. L'infrastructure peut être placée physiquement dans les locaux de l'organisation ou à l'extérieur.
– Cloud communautaire : l'infrastructure cloud est utilisée exclusivement par une communauté de clients d'organisations aux intérêts communs. Elle peut être gérée par une ou plusieurs organisations de la communauté, une tierce partie, ou une combinaison des deux. L'infrastructure peut être placée physiquement dans les locaux d'une ou plusieurs organisations de la communauté ou à l'extérieur.
– Cloud public : l'infrastructure cloud est utilisée pour le grand public. Elle peut être gérée par une organisation industrielle, académique, gouvernementale ou une combinaison de plusieurs organisations. L'infrastructure est placée physiquement dans les locaux du fournisseur de services.
– Cloud hybride : l'infrastructure cloud est une combinaison de deux ou plusieurs infrastructures cloud disctinctes (privée, communautaire ou publique) qui restent des entités à part entière mais sont liées entre elles par des technologies standardisées ou propriétaires afin de permettre la portabilité des données et des applications.

1.1.3 Technologies associées

Pour assurer les cinq caractéristiques essentielles du cloud permettant de délivrer des services, de nombreuses technologies sont utilisées. En effet, le cloud computing est *essentiellement un modèle économique, rendu possible par des avancées technologiques permettant la modulation des ressources* [86]. Ces avancées sont principalement articulées autour de la virtualisation, qui a permis une nouvelle et meilleure exploitation des

ressources informatiques physiques.

1.1.3.1 Virtualisation

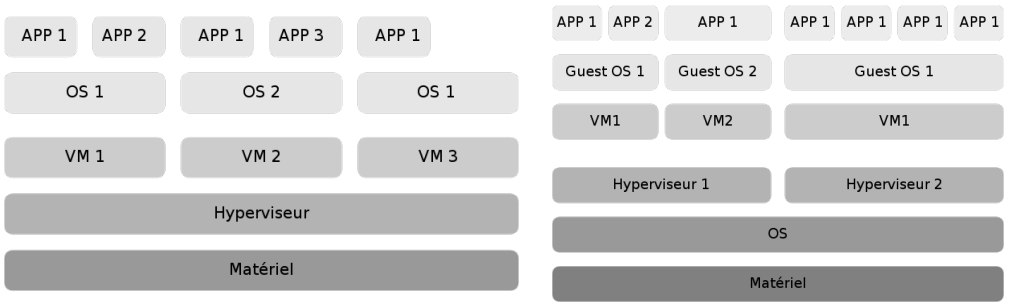
La virtualisation est un concept né dans les années 1960, autour des travaux d'IBM sur de puissants ordinateurs appelés *mainframe*. Il s'agit d'une technique de segmentation des ressources matérielles physiques, temporelles et spatiales entre plusieurs machines virtuelles. Dans [119], une machine virtuelle ou *virtual machine* (VM) est définie comme *une copie d'une machine réelle, efficace et isolée*. La virtualisation permet, entre autres, d'exécuter plusieurs systèmes d'exploitation sur une même machine physique. La segmentation et l'ordonnancement des machines virtuelles sont effectués par une entité qui vient se placer entre le matériel et les machines virtuelles : le gestionnaire de machines virtuelles ou *Virtual Machine Monitor* (VMM), également appelé hyperviseur.

Il existe trois principaux types de virtualisation [128] : l'émulation, la para-virtualisation et la virtualisation complète. Dans l'émulation, le système virtualisé est exécuté comme une application dans l'espace mémoire utilisateur. C'est le noyau du système d'exploitation hôte qui contrôle les appels système pour l'accès aux ressources matérielles. Dans le cas de la para-virtualisation, le code du système virtualisé est modifié pour pouvoir explicitement faire des appels à l'hyperviseur sous-jacent. En virtualisation complète, le système virtualisé n'a pas conscience de l'être et envoie ses instructions au matériel. Ces instructions sont alors interceptées par l'hyperviseur qui émule les ressources matérielles.

Deux types d'hyperviseurs prédominent : les hyperviseurs de type 1 (ou *bare metal*) et les hyperviseurs de type 2 (ou *hosted*). Les hyperviseurs de type 1 (Figure 1.2a) se chargent immédiatement après l'exécution du firmware (BIOS ou UEFI) de la machine et ne s'appuient que sur les services offerts par le firmware. VMware ESXi [31], Hyper-V [32], Xen [27] et KVM [12] sont des exemples d'hyperviseurs de type 1. Les hyperviseurs de type 2 (Figure 1.2b) s'exécutent après le chargement du système d'exploitation et s'appuient sur les services du noyau. VirtualPC [6], VirtualBox [17] et QEMU [23] sont des exemples d'hyperviseurs de type 2. L'assistance matérielle pour la virtualisation, offerte sur certaines architectures, permet aux machines virtuelles d'accéder directement au matériel en restant sous le contrôle de l'hyperviseur. Celui-ci peut pré-configurer les évènements sur lesquels il est notifié et doit intervenir. Cela augmente de manière significative les performances de la virtualisation.

1.1.3.2 Virtualisation de réseaux et réseaux virtuels

La virtualisation permet d'élaborer de nouvelles entités virtuelles, en plus des machines virtuelles. En effet, il est possible de constituer des infrastructures virtuelles, qui comprennent des centres de données virtuels ou *virtual datacenters* (vDC), contenant eux-mêmes des machines virtuelles, des espaces de stockage virtuels et des réseaux virtuels. Ces derniers, produits de la virtualisation de réseaux, permettent d'établir les communications entre machines virtuelles au sein des infrastructures virtuelles. En



(a) Hyperviseur de type 1

(b) Hyperviseur de type 2

FIGURE 1.2 – Types d'hyperviseurs

outre, le nombre et la taille des réseaux virtuels a poussé un consortium [1] de vendeurs ou éditeurs de matériel et logiciels réseau à créer le standard *Virtual eXtensible Local Area Network* (VXLAN) [97] pour répondre à ces contraintes.

Virtualisation externe de réseaux

La virtualisation de réseaux est un concept initialement utilisé dans les infrastructures réseau physiques traditionnelles. On parle alors de virtualisation externe de réseaux. Les réseaux locaux ou *Local Area Network* (LAN) sont divisés ou combinés en réseaux logiques appelés réseaux locaux virtuels ou *Virtual Local Area Network* (VLAN). Cela permet de créer des domaines de *broadcast* (domaines de diffusion) gérés par les commutateurs indépendamment de l'emplacement où se situent les équipements et terminaux : ce sont des domaines de diffusion gérés logiquement. Un identifiant (VLAN ID) est utilisé pour marquer l'appartenance des paquets à un VLAN. Il est véhiculé dans les liaisons entre commutateurs et cœurs de réseau/routeurs à travers des liens appelés *trunks*.

Virtualisation interne de réseaux

La virtualisation de réseaux dans les infrastructures virtuelles est appelée virtualisation interne de réseaux. Cela consiste à émuler les composants réseau : interfaces, liens, équipements et protocoles. Les ressources physiques du réseau sont alors partagées entre les machines virtuelles. Il est à noter que la virtualisation externe de réseau peut bien sûr également être utilisée dans les réseaux virtuels. En outre, le récent modèle *Software-Defined Networking* (SDN) offre de nouvelles possibilités pour les fournisseurs de service IaaS. Il s'agit d'une nouvelle manière de concevoir et gérer les architectures réseau, où le plan de contrôle est entièrement dissocié du plan de données. En effet, le plan de contrôle n'est alors pas déployé sur les mêmes équipements qui sont en charge de transmettre les données. Par exemple, les décisions de routage sur un commutateur sont déportées vers un contrôleur dédié, tandis que la fonction de transmission reste sur le commutateur. Le réseau est alors en quelque sorte "programmable".

1. VMware, Arista Networks, Cisco Systems, Citrix, Dell, Red Hat, Juniper Networks, Intel, Broadcom, Storvisor, Cumulus Networks, Pica8, Mellanox, OpenBSD.

VXLAN

Dans les infrastructures cloud, on peut potentiellement trouver un très grand nombre de machines virtuelles, et donc un très grand nombre de réseaux virtuels. Les VLANs ne permettent d'assigner que 4096 identifiants différents (les VLAN ID étant codés sur 12 bits). De plus, les VLANs ne permettent pas de répartir des réseaux logiques en traversant les frontières établies par les équipements de niveau 3 du modèle OSI (par exemple, un VLAN ne peut être réparti à différents endroits d'Internet) et restreignent donc leur utilisation. Ceci peut être un obstacle à la migration de machines virtuelles requérant le franchissement de frontières de niveau 3. Ces problèmes d'extensibilité sont résolus par le concept de VXLAN, qui consiste à encapsuler les trames de niveau 2 (pouvant bien entendu déjà appartenir à un VLAN et donc contenir un en-tête de VLAN) dans des paquets UDP de niveau 4, capables de traverser les domaines de diffusion. Les segments VXLAN sont identifiés par un identifiant : le *VXLAN Network Identifier* (VNI), codé sur 24 bits et permettant donc d'identifier plus de 16 millions de réseaux différents. Des tunnels sont ainsi créés sur des réseaux IP de niveau 3, permettant d'avoir des segments VXLAN à différents endroits. Les points d'entrée de tels tunnels, appelés *VXLAN Tunnel End Point* (VTEP), sont en charge d'encapsuler les trames et de les transmettre dans les paquets UDP, ou de décapsuler les paquets UDP et de transmettre les trames aux bons destinataires (en fonction du VNI). Les VXLAN résolvent donc le problème de la limitation de réseaux logiques dans le cloud, et permettent de franchir les frontières de niveau 3 pour répartir des réseaux logiques. Ceci est nécessaire par exemple lors de la migration d'une machine virtuelle entre deux machines physiques séparées par plusieurs frontières de niveau 3, comme le montre la Figure 1.3. Sur cette figure, la machine virtuelle grise assignée à un VXLAN peut, contrairement à la machine virtuelle rouge assignée à un VLAN, migrer au delà du périmètre de niveau 3 et se retrouve instantanément connectée sur le bon réseau IP après migration.

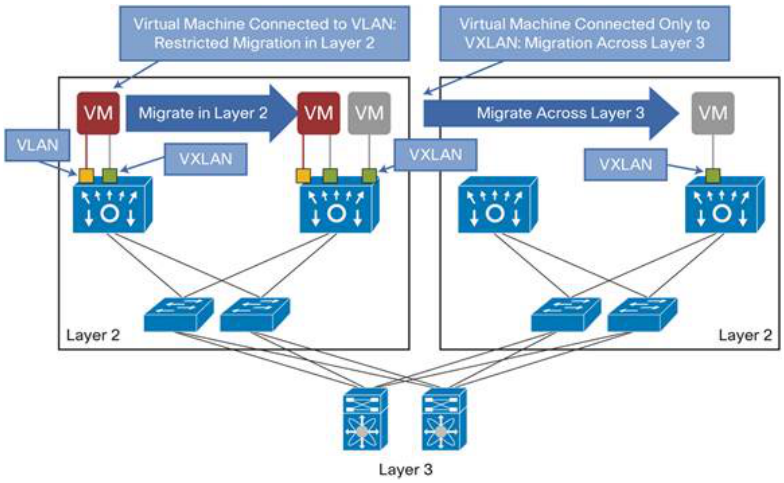


FIGURE 1.3 – Intérêt des VXLAN pour la migration (extrait de [3])

1.2 Problématiques de sécurité

Les infrastructures cloud sont, comme tout système informatique réparti, exposées à des problématiques de sécurité. En effet, le nombre et la diversité des utilisateurs de ces infrastructures ainsi que la quantité importante de composants matériels et logiciels impliquent la présence de menaces de sécurité. La sécurité est un domaine de la sûreté de fonctionnement, qu'il convient d'abord d'introduire. Pour ce faire, nous utilisons les termes et définitions tels que présentés dans le *Guide de la Sûreté de Fonctionnement* [44] et dans le projet *Malicious and Accidental Fault Tolerance for Internet Applications* (MAFTIA) [120]. Ensuite, nous nous intéressons à la sécurité, et particulièrement à la sécurité réseau dans le cloud computing.

1.2.1 La sûreté de fonctionnement

La sûreté de fonctionnement d'un système informatique est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre. Le service délivré par un système est son comportement tel que perçu par son, ou ses utilisateurs ; un utilisateur est un autre système (humain ou physique) qui interagit avec le système considéré. La non-sûreté de fonctionnement est lorsque la confiance ne peut plus, ou ne pourra plus, être placée dans le service délivré. La sûreté de fonctionnement est articulée autour de trois axes : les attributs la caractérisant, les entraves empêchant sa réalisation, et les moyens pour l'atteindre. La Figure 1.4 présente ces axes sous forme d'arbre.

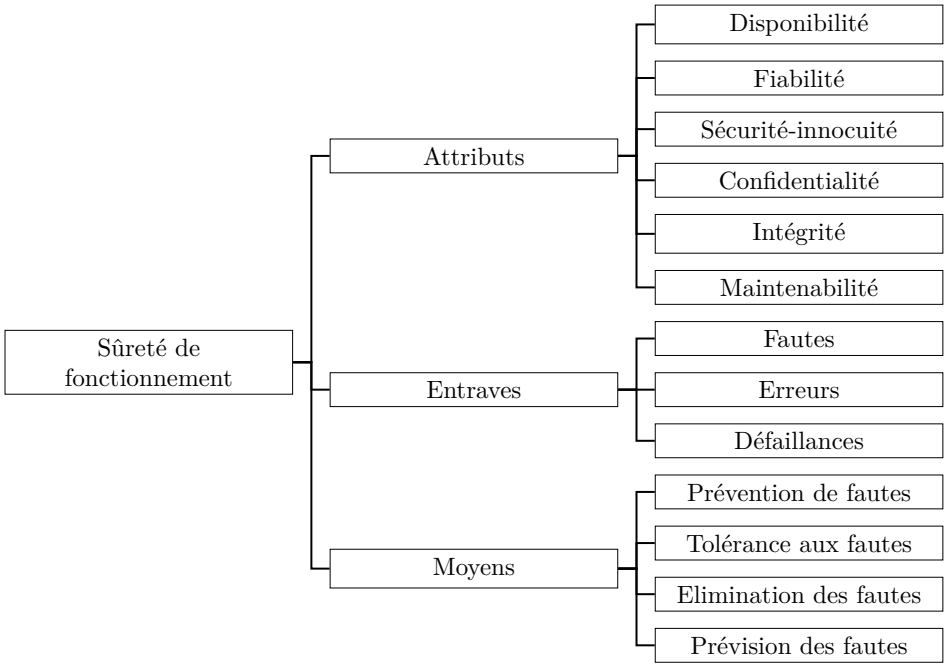


Figure 1.4 – Arbre de la sûreté de fonctionnement

1.2.1.1 Attributs de la sûreté de fonctionnement

Les attributs de la sûreté de fonctionnement permettent d'exprimer les propriétés attendues d'un système, et d'apprécier la qualité du service délivré, telle que résultant des entraves et des moyens de s'y opposer. Il s'agit de la disponibilité (aptitude à être prêt à l'utilisation), de la fiabilité (continuité de service), de la sécurité-innocuité (absence de conséquences catastrophiques), de la confidentialité (absence de divulgations non autorisées de l'information), de l'intégrité (absence d'altérations inappropriées de l'information) et de la maintenabilité (aptitude aux réparations et aux évolutions).

1.2.1.2 Entraves à la sûreté de fonctionnement

Les entraves à la sûreté de fonctionnement sont les circonstances indésirables, mais non attendues, causes ou résultats de la non-sûreté de fonctionnement. Il s'agit des fautes, des erreurs et des défaillances. Il existe une relation de causalité entre fautes, erreurs et défaillances. Une faute est la cause adjugée ou supposée d'une erreur. Les fautes peuvent être classées selon leur cause phénoménologique (physiques ou dues à l'homme), leur nature (accidentelles ou intentionnelles), leur phase de création ou d'occurrence (au cours du développement ou en opération), leur situation par rapport aux frontières du système (internes ou externes), et leur persistance (permanentes ou temporaires). Une erreur est la partie de l'état d'un système susceptible de provoquer une défaillance. Une défaillance survient lorsque le service délivré dévie de l'accomplissement de la fonction du système.

1.2.1.3 Moyens pour la sûreté de fonctionnement

Les moyens pour la sûreté de fonctionnement sont les méthodes et techniques permettant de fournir au système l'aptitude à délivrer un service conforme à l'accomplissement de sa fonction, et de donner confiance dans cette aptitude. Il s'agit de la prévention de fautes, de la tolérance aux fautes, de l'élimination des fautes et de la prévision des fautes. La prévention de fautes consiste à empêcher, par construction, l'occurrence ou l'introduction de fautes. Elle est principalement obtenue par des méthodes de spécification et de développement relevant de l'ingénierie des systèmes. La tolérance aux fautes consiste à fournir un service à même de remplir la fonction du système en dépit des fautes. Elle est mise en œuvre par la détection d'erreurs et le rétablissement du système. L'élimination des fautes consiste à réduire le nombre et la sévérité des fautes. Elle peut être réalisée pendant la phase de développement d'un système par vérification, diagnostic et correction, ou pendant sa phase opérationnelle par maintenance. La prévision des fautes consiste à estimer la présence, la création et les conséquences des fautes. Elle est effectuée par évaluation du comportement du système par rapport à l'occurrence des fautes, à leur activation et à leurs conséquences.

1.2.2 La sécurité

Le domaine de la sûreté de fonctionnement qui nous intéresse est celui de la sécurité-immunité, que nous appelons plus simplement sécurité. Dans [69], la sécurité, qui correspond au terme anglais *security*, est définie comme la *capacité du système informatique à résister à des agressions externes physiques ou logiques*. Les *Information Technology Security Evaluation Criteria* (ITSEC) [135] définissent la sécurité comme la combinaison de trois propriétés : la confidentialité, l'intégrité et la disponibilité de l'information. On parle donc de sécurité de l'information, des systèmes d'information ou des données. Une information est *un élément de connaissance traduit par un ensemble de signaux selon un code déterminé, en vue d'être conservé, traité ou communiqué* [8]. Une donnée est *une représentation d'une information sous une forme conventionnelle adaptée à son exploitation* [8]. Il est à noter qu'il existe aussi des méta-données, qui correspondent à des informations indirectes reliées aux informations ou aux services. Il s'agit par exemple de l'instant de création, de modification ou de destruction d'une donnée, de l'identité de la personne qui a réalisé une opération sur une donnée, de l'emplacement d'une donnée, etc.

La sécurité possède ses propres termes, dont nous rappelons les définitions avant de détailler les trois propriétés qui la composent, puis de donner une vue d'ensemble des moyens pour la sécurité.

1.2.2.1 Terminologie

Parmi les termes les plus utilisés dans le domaine de la sécurité informatique, on retrouve notamment *vulnérabilité*, *attaque*, *intrusion*, *menace*, *risque* et *incident*. Les termes *vulnérabilité*, *attaque* et *intrusion* sont définis à la fois dans [44] et dans [120], mais nous retenons les définitions de [120] car ce document est plus récent. Les définitions d'*intrusion* et *menace* sont issues de [44], et celle d'*incident* vient de la norme ISO 20000 [70].

- Vulnérabilité : faute créée durant le développement du système, ou durant l'opération, pouvant être exploitée afin de créer une intrusion.
- Attaque : faute d'intéraction malveillante, à travers laquelle un attaquant cherche à délibérement violer une ou plusieurs propriétés de sécurité. Il s'agit d'une tentative d'intrusion.
- Intrusion : faute malveillante externe résultant d'une attaque qui a réussi à exploiter une vulnérabilité.
- Menace : possibilités et probabilités d'attaque contre la sécurité. Une menace est définie par le processus d'attaque, par la cible et par le résultat (conséquences de la réussite d'une attaque).
- Risque : résultat de la combinaison de menaces et de vulnérabilités.
- Incident : évènement qui ne fait pas partie des opérations standards d'un service et qui provoque ou peut provoquer une interruption de service ou altérer sa qualité.

1.2.2.2 Propriétés

La sécurité est la combinaison de trois propriétés (la confidentialité, l'intégrité et la disponibilité), dont nous reprenons ici les définitions de [69].

La *confidentialité* est la propriété d'une information de ne pas être révélée à des utilisateurs non autorisés à la connaître. La désormais célèbre vulnérabilité *Heartbleed* [10], présente dans la bibliothèque logicielle de cryptographie OpenSSL, permet des attaques violant la confidentialité des données. En effet, en exploitant cette faille, un attaquant peut récupérer des informations stockées en mémoire sur les machines ciblées.

L'*intégrité* est la propriété d'une information de ne pas être altérée. Le ver ILOVEYOU [50], apparu en 2000, se propageait par le biais de la messagerie Microsoft Outlook. Une fois exécuté, il remplaçait de nombreux fichiers présents sur le système. En altérant l'information de ces fichiers, il violait la propriété d'intégrité des données.

La *disponibilité* est la propriété d'une information d'être accessible lorsqu'un utilisateur autorisé en a besoin. À titre d'exemple, les plateformes de jeux-vidéos en ligne de Sony et Microsoft ont été plusieurs fois la cible d'attaques de déni de service, rendant inaccessibles ces plateformes et violant la propriété de disponibilité des données.

D'autre propriétés comme l'authenticité, l'auditabilité, l'intimité, la pérénnité, ou l'exclusivité sont parfois employées. Cependant, elles restent exprimables en termes de confidentialité, intégrité ou disponibilité. Par exemple, l'authenticité d'un message correspond à l'intégrité de son contenu (une donnée) ainsi que de son origine (une méta-donnée).

1.2.2.3 Moyens pour la sécurité

Le Tableau 1.1, extrait de [120], présente une classification des méthodes disponibles pour garantir la sécurité vis-à-vis des attaques, vulnérabilités et intrusions, en fonction des quatre moyens pour la sûreté de fonctionnement.

1.2.3 Menaces, vulnérabilités et attaques dans le cloud

Le cloud computing est particulièrement exposé à des menaces et vulnérabilités pouvant être de natures variées. Nous présentons les principales caractéristiques des attaquants, menaces et vulnérabilités présentes dans le cloud.

1.2.3.1 Nature des attaquants

Pour comprendre qui peuvent être les attaquants dans le cloud, nous partons du principe que tout acteur du cloud est un attaquant potentiel. Un acteur est une personne pouvant remplir un ou plusieurs rôles. Il existe de nombreux rôles possibles dans le cloud. Celui-ci étant aussi un modèle économique, certains rôles sont des fonctions à tendance plus économique (liée aux notions de service, client et fournisseur). Par ailleurs, les infrastructures cloud impliquant de nombreux composants technologiques,

	Attaque humaine	Attaque technique	Vulnérabilité	Intrusion
Prévention	Dissuasion, lois, pression sociale, service secret...	Pare-feu, authentification, autorisation...	Spécifications formelles et semi-formelles, méthodes rigoureuses de développement et gestion	Prévention et élimination des attaques et vulnérabilités
Tolérance	Prévention des vulnérabilités, élimination et tolérance aux intrusions		Prévention et élimination des attaques, tolérance aux intrusions	Détection et recouvrement d'erreurs, masquage des fautes, détection d'intrusion, gestion des fautes
Elimination	Contre-mesures physiques, capture de l'attaquant	Maintenance préventive et corrective visant à supprimer les agents malveillants	Preuve formelle, model-checking, inspection, test, maintenance préventive et corrective, incluant les patchs de sécurité	Elimination des attaques et des vulnérabilités
Prévision	Collecte de renseignements, évaluation des menaces...	Analyse des agents malveillants latents	Evaluation des vulnérabilités, des difficultés de les exploiter, de leurs conséquences potentielles...	Prévision des vulnérabilités et des attaques

Tableau 1.1 – Classification des méthodes pour la sécurité

d'autres rôles sont à tendance plus technologique (liée aux contrôles sur les composants). Le Tableau 1.2 reprend les différents rôles tels que cités dans la littérature, selon qu'ils sont d'une orientation économique ou technologique. Dans la suite de ce document, nous nous focaliserons sur les rôles économiques de client et fournisseur de services.

En outre, les notions d'attaquant interne (*insider* en anglais) et attaquant externe (*outsider* en anglais) sont difficiles à définir dans ce contexte. Selon [63], un attaquant interne est une personne menant son attaque depuis l'intérieur du système attaqué, tandis qu'un attaquant externe mène son attaque depuis l'extérieur du système. Le *Computer Emergency Response Team* (CERT) de l'Université de Carnegie Mellon a défini un attaquant interne comme *un ancien ou actuel employé, contractant ou autre partenaire ayant un accès sur le réseau, système ou les données d'une organisation et utilisant cet accès dans un objectif néfaste affectant la confidentialité, l'intégrité ou la disponibilité de l'information des systèmes d'information de l'organisation* [57].

Deux types principaux d'attaquant interne pouvant exister dans le cloud sont différenciés dans [58] : employé du fournisseur de cloud (administrateur pouvant cibler par exemple une organisation cliente ou son employeur) et employé de l'organisation utilisatrice du cloud (pouvant cibler par exemple les données du cloud ou les données externes de son employeur). Le premier type d'attaquant interne correspond à un administrateur corrompu du fournisseur de cloud, réalisant du vol d'informations sensibles dont les motivations sont généralement financières. Il peut également être un

Auteurs	Rôles économiques	Rôles technologiques
NIST [112]	Consommateur de services cloud Fournisseur de services cloud Développeur de services cloud Distributeur de services cloud	
Bauer et Adams [48]	Vendeurs Fournisseur de services Consommateurs du cloud Utilisateurs finaux	
Vaquero *et al.* [142]	Fournisseur de services Utilisateur de services Fournisseur d'infrastructure	
Leimeister *et al.* [92]	Client Fournisseur de services Fournisseur d'infrastructure Agrégateur de fournisseurs de services Fournisseur de plateforme Consultant	
Eucalyptus [73]	Manager Manager de services cloud Utilisateur de cloud Architecte des données cloud Administrateur de stockage cloud	Administrateur système Opérateur d'ordinateur Administrateur réseau Administrateur de stockage Administrateur de base de données Utilisateur final Architecte de cloud Développeur de code Administrateur de cloud Opérateur de cloud Architecte d'application cloud Développeur cloud

Tableau 1.2 – Rôles dans le cloud

administrateur employé de l'organisation cliente, même si ceci n'est pas fréquent dans le cloud car l'administrateur est généralement employé par le fournisseur de cloud. Un employé qui veut nuire à son employeur (le fournisseur de cloud) pourrait faire des dégâts à une organisation cliente pour desservir la réputation du fournisseur de cloud. Les quatre niveaux d'administrateurs considérés ici sont :

1. Administrateurs d'applications.
2. Administrateurs système.
3. Administrateurs des machines virtuelles.
4. Administrateurs de l'hébergement.

Un administrateur a les privilèges des administrateurs de tous les niveaux qui lui sont inférieurs (par exemple, un administrateur système a également les privilège de l'administrateur d'applications). Le second type d'attaquant interne est celui qui cherche (potentiellement dirigé par une source externe) à obtenir un accès non autorisé aux données de son organisation qui utilise le cloud. Toujours dans [58], un troisième type d'attaquant interne est même proposé, proche du précédent, mais à la différence que

l'employé de l'organisation utilise des services cloud pour mener une attaque sur son propre employeur, mais pas forcément sur des données localisées dans le cloud. En résumé, la classification des attaquants internes du cloud de [58] sépare donc les administrateurs du fournisseur de cloud (avec différents niveaux ciblant son employeur ou une organisation cliente) et employé de l'organisation cliente (ciblant son employeur à travers le cloud). Dans [63], les attaquants internes sont différenciés selon : fournisseur de services, employé du fournisseur de services et employé (avec des privilèges d'accès autorisés) dans l'organisation de l'utilisateur du cloud. Toujours d'après [63], un attaquant externe correspond simplement à un utilisateur n'appartenant pas à l'organisation de sa victime. Son rayon d'action est relativement vaste, pouvant agir depuis Internet et pouvant généralement facilement s'introduire dans le périmètre cloud en souscrivant à un abonnement par exemple.

1.2.3.2 Classes d'attaques, incidents, et menaces

La littérature offre différentes manières de classer des attaques informatiques [81], en fonction de différents critères : type d'attaquant, objectif de l'attaque, vecteur d'attaque, cible de l'attaque, résultat de l'attaque, etc. Etant donné la complexité des rôles parmi les acteurs du cloud, la diversité des cibles et des objectifs potentiels, il est plus pertinent d'utiliser le vecteur d'attaque pour obtenir une classification ordonnée et facilement compréhensible. Par conséquent, pour présenter les grandes familles d'attaques informatiques, nous avons retenu une classification reposant sur le vecteur d'attaque comme critère, à l'instar de celle proposée dans [78]. Un vecteur d'attaque est le moyen pour une attaque d'atteindre sa cible. Le Tableau 1.3 présente une classification des attaques informatiques par vecteur d'attaque. Elle permet de donner une vue d'ensemble des catégories d'attaques informatiques principales, mais n'a pas pour but de lister toutes les attaques existantes. Nous y avons répertorié les principales attaques retrouvées dans les environnements cloud. Ces attaques incluent des vulnérabilités connues et traditionnellement retrouvées dans les infrastructures physiques. Cependant, leur exploitation et leur visibilité sont facilitées par la concentration actuelle de capacités importantes de calculs et de stockage, comme expliqué dans [131]. Par ailleurs, on retrouve aussi de nouvelles formes d'attaque spécifiques aux environnements virtuels comme les attaques par reconnaissance et canaux auxiliaires [125,138,150]. Cependant, nous ne détaillons pas ces attaques car elles ne font pas partie des attaques réseau à couvrir par les mécanismes de sécurité que nous considérons dans cette thèse.

D'autre part, Alert Logic réalise régulièrement des études [35] sur les incidents dans les infrastructures cloud, à partir de la collecte de données en grande quantité. Les incidents y sont triés selon six classes, présentées dans le Tableau 1.4. Il est à noter que les termes attaque et incident sont ici confondus. On constate que ces incidents sont de même nature que ceux retrouvés dans des infrastructures traditionnelles (hors cloud). Des études sont d'ailleurs menées sur des incidents survenus dans des cloud et ceux survenus dans des infrastructures traditionnelles, afin d'effectuer une comparaison entre les deux types d'incident. On constate dans leur rapport que même si les

Niveau 1	Niveau 2	Niveau 3	Niveau 4	Niveau 5
Malware	Virus			
	Vers			
	Chevaux de Troie			
Dénis de service	Hôte	X-DoS	XML Parser DoS	
			XML Attribute Blowup	
			XML Entity DoS	
		H-DoS		
		ReDoS		
		Public Key DoS		
		Ping of Death		
	Réseau	Flooding	TCP flooding	
			UDP flooding	
			ICMP flooding	
			SIP flooding	
			XML flooding	
			STP BPDU flooding	
		Smurfing		
Attaques applicatives	Débordements de tampon	Pile		
		Tas		
	Débordements d'entier			
	Chaînes de format			
Attaques réseau	Spoofing	ARP Spoofing	Attaques par Gratuitous ARP	
			Attaques par réponse ARP	
			Attaques par requête ARP forgée	
		IP Spoofing		
		STP BPDU Spoofing		
	Vols de sessions	Interceptions SSL/SSH		
	Attaques sur les réseaux sans-fil			
	Attaques applicatives Web	Injections	Injections XML	Injections de référence d'entité
				Injections par échappement de caractères
			Injections SQL	
			Injections xPath	
		XSS		
		CSRF		
		Violations de gestion d'authentification et de session	SOAPAction Spoofing	
		Attaques SOAP	WS-Addressing Spoofing	
			XML Signature Element Wrapping	
			Redirections de référence	
			Attaque SOAP Array	
	Attaques sur les VLAN	VLAN Hopping	Switch Spoofing	
			Double tagging	
	Attaques sur les VXLAN	VXLAN Spoofing		
Attaques physiques	Attaques DMA			
	Van Eck phreaking			
Attaques sur les mots de passe	Estimations	Attaques par force brute		
		Attaques par dictionnaire		
Attaques d'écoute et reconnaissance	Sniffing	Canal auxiliaire	Etude des traces	
			Etude des accès	
			Etude temporelle	
	Mapping	Vérifications de co-résidence	Vérifications par temps de réponse réseau	
			Vérifications par adressage IP	
			Vérifications par passerelle réseau	
	Scanning	Scan de ports	Scan TCP	
			Scan UDP	
			Scan ICMP	

Tableau 1.3 – Classification d'attaques par vecteur d'attaque

attaques applicatives Web constituent une proportion prédominante des incidents du cloud, ces derniers sont de plus en plus proches que ceux historiquement rencontrés dans les infrastructures traditionnelles.

Classe d'incident	Définition
Attaques applicatives	Tentatives d'exploit sur des applications ou services ne s'exécutant pas sur le protocole HTTP.
Attaques par force brute	Tentatives d'exploit comptant un grand nombre de combinaisons pour trouver une faille.
Activité de malware/botnet	Activité malveillante d'un logiciel installé sur un hôte et pouvant communiquer à l'aide d'un canal de commande et contrôle.
Attaque de reconnaissance	Activité centrée sur les balayages et la cartographie de réseaux, applications ou services.
Scan de vulnérabilité	Découverte automatique de vulnérabilités.
Attaques applicatives Web	Attaques ciblant l'interface, la logique ou la base de données d'une application Web.

Tableau 1.4 – Classes d'incidents dans le cloud

De son côté, la Cloud Security Alliance (CSA) [79] a défini sept grandes classes de menaces pour le cloud :

- Abus et usage néfaste du cloud.
- APIs et interfaces non sécurisées.
- Malveillance interne.
- Problèmes liés aux technologies de partage.
- Perte ou fuite de données.
- Détournement de compte ou de service.
- Profil de risque inconnu.

Ces efforts de classification témoignent de la prise de conscience des dangers liés aux menaces dans le cloud et de l'importance actuelle de sécuriser ces environnements.

1.3 Mécanismes de sécurité réseau dans le cloud

Les mécanismes de sécurité réseau ont pour but d'assurer la sécurité des hôtes et applications des réseaux auxquels ils appartiennent. Nous nous intéressons particulièrement à deux mécanismes de sécurité réseau :

- Le contrôle des accès réseau, réalisé par les pare-feu ou *firewall*.
- La détection d'intrusion, réalisée par les systèmes de détection d'intrusion ou *Intrusion Detection Systems* (IDS).

Ces mécanismes ont pour but d'appliquer la politique de sécurité au sein des réseaux qu'ils protègent.

1.3.1 Pare-feu virtuels

Un pare-feu est un *outil permettant de contrôler le trafic circulant entre l'intérieur et l'extérieur d'un périmètre de sécurité* [105]. Le périmètre de sécurité *constitue la limite entre le réseau que l'on considère comme sûr ou que l'on désire protéger et le reste de l'Internet* [105]. Les pare-feu peuvent offrir différents services, mais nous nous

intéressons au service de base, à savoir le contrôle d'accès au niveau réseau. La fonction principale du contrôle d'accès réseau est le filtrage de paquets, opéré par des règles de filtrage permettant d'interdire ou autoriser certains types de trafic. Ces règles sont établies en fonction des paramètres des en-têtes protocolaires situés dans les paquets. Elles sont appliquées sur chaque paquet transitant par le pare-feu, en respectant un certain ordre de priorité dans l'application des règles. La plupart des pare-feu sont généralement à suivi d'état ou plus simplement à états (*stateful* en anglais), c'est-à-dire qu'ils vérifient en plus l'appartenance des paquets à une connexion en cours (comme une connexion TCP) avant de les autoriser. Ils vérifient ainsi que chaque paquet autorisé par les règles initie une nouvelle connexion ou fait bien partie d'une connexion déjà existante. En outre, différents niveaux d'inspection peuvent être réalisés sur les paquets. Le niveau le plus courant est le *Deep Packet Inspection* (DPI) qui s'attache à regarder le contenu d'un paquet en profondeur pour autoriser l'accès ou non. Les pare-feu applicatifs Web ou *Web Application Firewall* (WAF), dédiés à la protection des serveurs Web, sont également de plus en plus répandus.

La complexité des architectures cloud, couplée à la complexité des applications actuelles, rend les exigences en contrôle d'accès réseau plus importantes. Le travail des pare-feu, en charge de ces contrôles d'accès, est de ce fait rendu plus compliqué. Le périmètre de sécurité est plus complexe à dessiner que dans un environnement traditionnel, du fait de la multiplicité des organisations présentes dans le cloud. Ainsi, le positionnement des pare-feu et leur comportement sont des aspects stratégiques. Pour maintenir une défense en profondeur dans les infrastructures virtuelles, il existe les deux types de pare-feu virtuels suivants, illustrés par la Figure 1.5 :

- Pare-feu en mode pont : machine virtuelle déployée comme passerelle des réseaux virtuels, capable de router, filtrer et traduire les adresses du trafic entrant et sortant. Ces pare-feu sont généralement contrôlés par les clients.
- Pare-feu en mode hyperviseur : composant logiciel embarqué dans l'hyperviseur qui filtre le trafic envoyé ou reçu par les machines virtuelles sans prendre en compte la topologie réseau. Il est généralement contrôlé par le fournisseur de services, mais certaines règles peuvent être appliquées par les clients seulement sur certains réseaux virtuels, ce qui implique de donner un contrôle partiel aux clients sur ce type de pare-feu.

1.3.2 Systèmes de détection d'intrusion

D'après [120], la détection d'intrusion *concerne l'ensemble des pratiques et mécanismes utilisés pour la détection d'erreurs pouvant conduire à une défaillance de sécurité, et/ou pour la détection d'attaques.* Toujours d'après [120], un IDS est *l'implémentation des pratiques et mécanismes de détection d'intrusion.* Les IDS ont donc pour rôle de détecter et/ou bloquer (on parle dans ce cas d'*Intrusion Prevention System*) des attaques survenant au sein d'un système ou d'un réseau. Ils peuvent être déployés sur l'hôte surveillé, on parle alors de *Host-Based Intrusion Detection System* (HIDS) ; ou bien sur le réseau surveillé, on parle alors de *Network-Based Intrusion Detection*

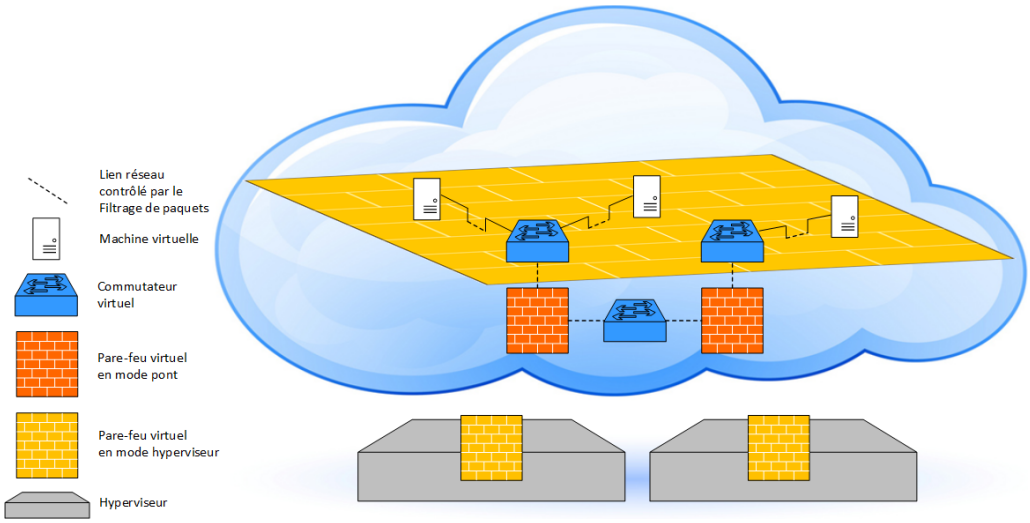


FIGURE 1.5 – Pare-feu virtuels dans le cloud

System (NIDS).

Comme le montre la Figure 1.6, il existe deux techniques principales de détection : par signatures (*signature-based detection* ou *misuse detection* en anglais) ou par comportements (*anomaly detection* en anglais). L'approche par signatures consiste à détecter des attaques en vérifiant si les observations correspondent à des attaques connues, tandis que l'approche par comportements (ou par détection d'anomalies) consiste à détecter une attaque en vérifiant que les observations ne correspondent pas à des comportements légitimes de référence. Certains IDS combinent les deux approches afin d'obtenir de meilleurs résultats.

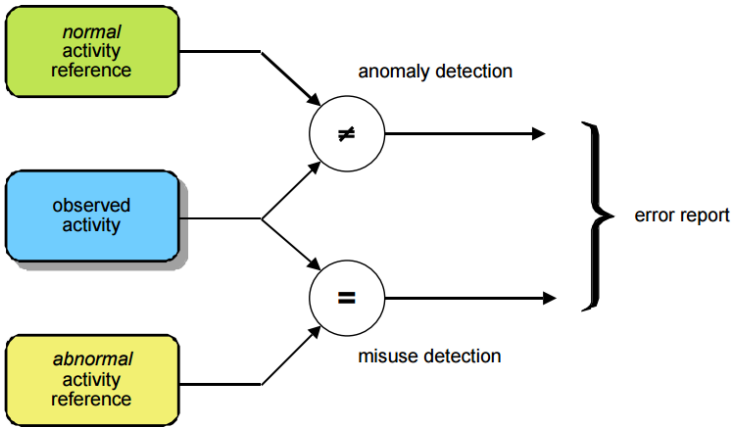


FIGURE 1.6 – Techniques de détection d'intrusion (extrait de [120])

1.3.2.1 Niveaux de déploiement des sondes dans le cloud

Dans une infrastructure physique traditionnelle, les sondes de détection sont généralement placées en entrée des réseaux à surveiller, ou dans les systèmes d'exploitation à surveiller. Dans le cloud, la multitude d'organisations partageant l'infrastructure et l'ajout de la virtualisation rend la stratégie de positionnement des sondes plus délicate à établir. La *Cloud Security Alliance* (CSA) a décrit plusieurs niveaux de déploiement possibles de sondes IDS/IPS dans le cloud [41] :

1. Virtuel interne : la sonde est déployée sur un équipement virtuel qui relie les commutateurs virtuels par un pont et est reliée à une interface réseau physique de l'hôte.
2. Physique externe : la sonde est déployée sur un équipement physique dédié relié à l'hôte physique qui redirige le trafic depuis ses commutateurs virtuels grâce à la mise en miroir du trafic ou *port mirroring* an anglais, ou transmet directement le trafic à la sonde qui l'intercepte.
3. VMM : la sonde est déployée au sein du gestionnaire de machines virtuelles en utilisant les APIs offertes.
4. Système invité : la sonde est déployée au sein du système d'exploitation des machines virtuelles.
5. Applicatif : la sonde est déployée au sein même du code de l'application surveillée.
6. Hybride : la sonde est déployée au sein du VMM pour intercepter le trafic et l'envoyer vers une sonde virtuelle interne ou physique externe.

La Figure 1.7 montre ces différents niveaux de déploiement (identifiés par les mêmes numéros sur la figure) au sein d'une infrastructure virtuelle. Le Tableau 1.5 présente les avantages et inconvénients de chacun des niveaux de déploiement possibles. Ce tableau montre que pour déployer des sondes, un compromis est à apprécier entre facilité de déploiement, mise à l'échelle, performances, et exposition à des failles.

1.3.2.2 Caractéristiques de fonctionnement dans le cloud

Les nouvelles approches pour la détection d'intrusion dans le cloud proposent des caractéristiques de fonctionnement propres à l'environnement dans lesquels ces systèmes sont censés évoluer.

La plupart des solutions proposées sont distribuées, on parle alors de *Distributed IDS* (DIDS). En effet, dans un environnement traditionnel, une seule sonde positionnée à l'entrée du réseau d'une entreprise pouvait suffire pour surveiller les attaques venant de l'extérieur. Comme expliqué précédemment, le périmètre de contrôle est plus poreux dans le cloud, et il convient donc de positionner des sondes des manière distribuée pour surveiller l'ensemble des différents trafics possibles et éviter les surcharges. Un des défis est alors la capacité de synchronisation des sondes et de corrélation des évènements détectés. La corrélation a pour but d'améliorer la détection des intrusions et de réduire le nombre de fausses alarmes, et cette fonction est réalisée par un centre

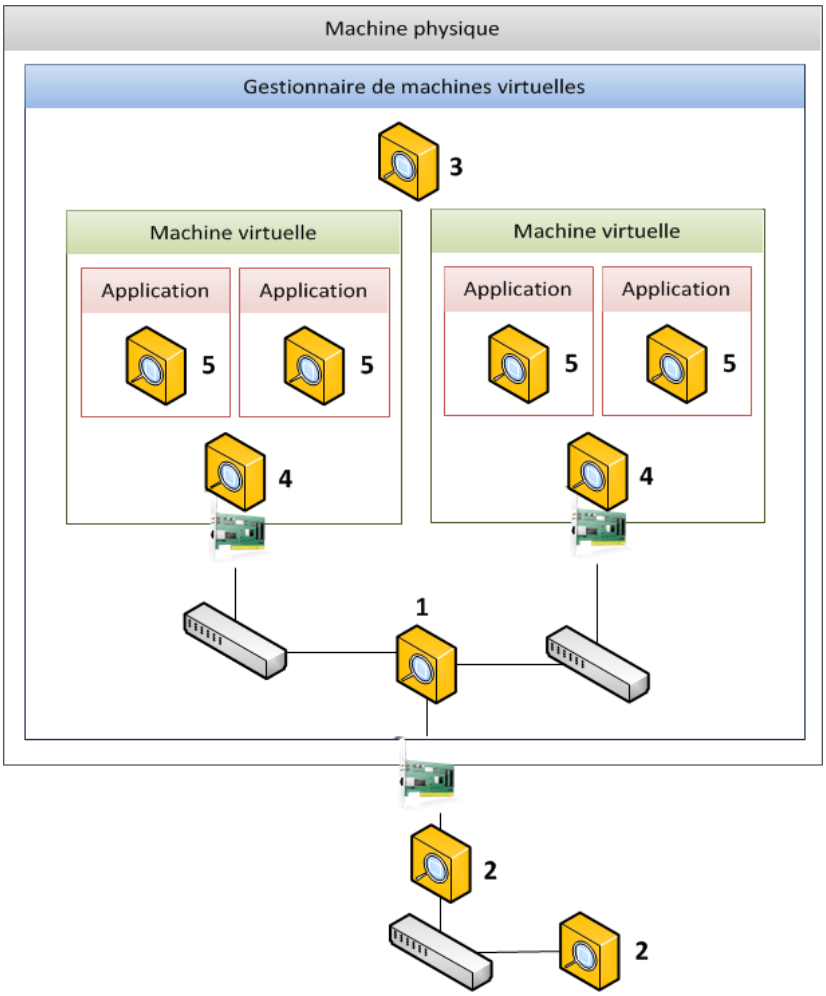


FIGURE 1.7 – Niveaux de déploiement des sondes d'IDS/IPS dans le cloud

de gestion des évènements et informations de sécurité, ou *Security Information and Event Management* (SIEM). Le rôle d'une telle unité est donc de récupérer, analyser et offrir une réponse aux évènements survenant dans les infrastructures cloud. Bien que de nombreux formats différents soient utilisés (syslog, CSV, JSON...), il existe un format d'échange de données standard entre sondes et systèmes de gestion, l'*Intrusion Detection Message Exchange Format* (IDMEF), défini dans la RFC 4765 [65].

La deuxième particularité de certains IDS dans le cloud est leur caractère coopératif. Ceci consiste à informer toutes les autres sondes en cas d'alerte générée par une des sondes. Un vote majoritaire peut alors être effectué pour valider l'alerte et remplir les tables de filtrage (servant à écarter les paquets malveillants) des autres sondes. On notera que la coopération des sondes diffère de la corrélation des alertes, dans le sens où la coopération aide à la prise de décision quant à la validation de la levée d'alertes

Déploiement	Avantages	Inconvénients
Virtuel interne	- Facile à déployer. - Interception de trafic chiffré plus simple.	- Perte des bénéfices du support matériel (ASIC...). - Une surcharge de la sonde impacte rapidement les machines virtuelles où l'hôte physique. - Une compromission de l'hyperviseur compromet la sonde. - Risques de perte de connectivité dans le cas d'un IPS due à l'absence des fonctionnalités matérielles de *fail-open*[2].
Physique externe	- Indépendance vis-à-vis de l'hyperviseur. - Une surcharge de la sonde a moins d'impact sur les machines virtuelles ou l'hôte physique. - Dans le cas d'un IPS, mécanisme de *fail-close*[3] disponible. - Interception de trafic chiffré plus simple	- Dans le cas d'un IDS, tous les commutateurs virtuels ne supportent pas le port mirroring. - Matériel physique dédié requis. - Problèmes possibles de mise à l'échelle.
VMM	- Facile à déployer. - Interception du trafic entre machines virtuelles. - Interception des opérations de l'invité. - Inspection possible des machines virtuelles hors-ligne.	- Perte des bénéfices du support matériel (ASIC...). - Une surcharge de la sonde impacte rapidement les machines virtuelles où l'hôte physique. - Une compromission de l'hyperviseur compromet la sonde.
Système invité	- Facile à déployer. - Interception du trafic entre machines virtuelles. - Interception des opérations de l'invité.	- Perte des bénéfices du support matériel (ASIC...). - Une surcharge de la sonde impacte rapidement les machines virtuelles où l'hôte physique. - Une compromission de l'invité compromet la sonde.
Applicatif	- Possibilité d'utiliser la logique de l'application pour définir plus précisément les tentatives d'intrusion. - Aucun déploiement nécessaire.	- Requiert le support de l'application. - Perte des bénéfices du support matériel (ASIC...). - Une surcharge de la sonde impacte rapidement les machines virtuelles où l'hôte physique. - Une compromission de l'invité compromet la sonde.
Hybride	- Interception du trafic entre machines virtuelles. - Possibilité d'éviter les surcharges en utilisant une sonde physique externe. - La politique d'inspection peut suivre facilement les machines virtuelles en mouvement.	- Une compromission de l'hyperviseur compromet la sonde.

Tableau 1.5 – Avantages et inconvénients des niveaux de déploiement des sondes d'IDS/IPS dans le cloud

et à leur diffusion, tandis que la corrélation aide à l'interprétation des évènements par confrontation de plusieurs alertes. En outre, dans [96], il est précisé qu'un IDS distribué sans coopération est exposé aux problèmes de points individuels de défaillance ou *Single Point of Failure* (SPOF).

Une autre propriété des IDS du cloud est de pouvoir offrir différents niveaux pour la détection, en fonction des besoins et activités des utilisateurs du cloud. C'est le cas par exemple dans [91], où l'utilisation du cloud par les utilisateurs est observée afin de définir un niveau d'anomalie (caractérisé par l'accumulation de points de risque) qui sert ensuite à assigner un niveau de détection à chaque sonde déployée. L'objectif de cet aspect multi-niveaux est donc d'adapter la sécurité en fonction des comportements observés dans le but d'améliorer les performances des IDS, car plus le niveau de détection d'un IDS est élevé (utilisation de nombreuses règles), plus sa consommation de ressources est importante. Par exemple, plus un utilisateur a une activité suspecte, plus son nombre de points de risque augmente et donc plus son niveau de détection requis s'élève. Ce niveau de détection est alors utilisé pour déterminer le type d'IDS

2. Mode de défaillance où la sonde d'IPS physique laisse passer le trafic.
3. Mode de défaillance où la sonde d'IPS physique ne laisse pas passer le trafic.

qui sera appliqué à son infrastructure.

Certaines approches ont proposé un IDS dont le service est offert à l'utilisateur du cloud, comme tout service cloud. On parle alors d'*Intrusion Detection System as a Service* (IDSaaS). Ce principe est notamment détaillé dans [39, 77], où les administrateurs système des clients ont la possibilité de surveiller et réagir aux attaques, ainsi que de dimensionner les composants de l'IDS.

Le Tableau 1.6 synthétise les caractéristiques de fonctionnement de différentes solutions d'IDS pour le cloud proposées dans la littérature.

Auteurs	Déploiement	Distribué	Coopératif	Multi-niveaux	IDSaaS
Vieira *et al.* [144]	Système invité	*	*		
Lee *et al.* [91]	Système invité	*		*	
Dastjerdi *et al.* [64]	Système invité ou Applicatif	*	*		
Bakshi et Yogesh [46]	Virtuel interne				
Roschke *et al.* [126]	Virtuel interne	*		*	
Alharkan et Martin [39]	Virtuel interne		*	*	*
Lo *et al.* [96]	Physique externe	*	*		
Mazziarello et al. [102]	Physique externe				
Hamad et Al-Hoby [77]	Physique externe	*		*	*

Tableau 1.6 – Caractéristiques de fonctionnement d'IDS pour le cloud

1.4 Analyse et évaluation de la sécurité

L'objectif de l'évaluation et de l'analyse d'un système est de révéler des fautes ou de fournir des valeurs permettant de mesurer l'efficacité des mécanismes de protection vis-à-vis de malveillances. L'analyse et l'évaluation de la sécurité participent à l'élimination et la prévision des fautes. C'est un domaine de recherche qu'il convient d'introduire, car dans le chapitre suivant nous détaillons l'état de l'art des méthodes et techniques utilisées pour évaluer et analyser les mécanismes de sécurité réseau qui nous intéressent.

Analyse

Analyser un système consiste à examiner ou étudier ce système afin d'en dégager et comparer les composants. Comme décrit dans [44], l'analyse est issue du domaine de la vérification, qui permet de révéler l'existence de fautes. En effet, la vérification peut être mise en œuvre par des techniques d'analyse. Nous n'avons pas pour but de mener des analyses de codes ou d'instances de logiciels, mais de mener des analyses de mécanismes de sécurité. Cependant, le déploiement d'un mécanisme de sécurité peut être assimilé à l'installation et la configuration de logiciels de sécurité. L'analyse

de mécanismes de sécurité correspond ainsi à l'analyse de la bonne installation et configuration des logiciels de sécurité. Par exemple, on cherche à mettre en évidence la présence de fautes dans l'application d'une politique de sécurité sur un réseau. Ceci peut être lié à des vulnérabilités ou des erreurs de configuration résiduelles. De manière générale, pour la vérification, l'analyse peut être statique : manuelle (revues ou inspections) ou automatique (utilisant des outils informatiques) ; ou dynamique : par exécution symbolique (mener une exécution en fournissant des expressions symboliques en entrées et récupérer de nouveaux symboles en résulats d'exécution) ou test (exécuter un système en fournissant des entrées valuées).

Évaluation

Évaluer un système consiste à estimer ou apprécier ce système par rapport à une mesure. Il existe historiquement deux grandes familles de méthodes d'évaluation de la sécurité : les évaluations qualitatives et les évaluations quantitatives. Comme expliqué dans [141], les approches qualitatives (ITSEC, Critères Communs et normes ISO) ne sont pas conçues pour évaluer régulièrement la sécurité d'un système en opération, ni de comparer des systèmes entre eux ou le même système à différents intervalles de temps. Par conséquent, ces approches ne semblent pas adaptées pour le suivi de l'évolution de la sécurité dans le cloud. En revanche, les approches quantitatives ont pour but de quantifier (rendre mesurable) la sécurité ou les mécanismes de protection associés en mode opérationnel. Nous devons alors appronfondir deux termes : *mesure* et *métrique*. D'après [8], une mesure est une *quantité servant d'unité de base pour cette évaluation*. D'après [90], il s'agit d'une *grandeur prise comme terme de comparaison pour évaluer la durée, l'étendue, la quantité, le poids, etc.* Le mot *métrique* est un adjectif *qui a rapport à la mesure appelée Mètre* [8], ou *qui a rapport au mètre ou aux mesures auxquelles il sert de base* [90]. L'utilisation de métrique en tant que nom commun dans la langue française est un abus lié à son existence en tant que nom commun dans la langue anglaise. Il est donc couramment employé à la place de mesure. Les mesures peuvent être déterminées à partir d'évaluations fondées sur des modèles ou d'évaluations expérimentales.

Les évaluations fondées sur les modèles reposent principalement sur les vulnérabilités présentes dans les systèmes évalués afin d'établir des liens entre elles pour construire des scénarios d'attaque. On peut citer notamment les graphes des privilèges [59, 116], graphes d'attaque [59, 83, 114, 115, 117, 118, 132–134, 139], arbres d'attaque [130, 146], et réseaux de Pétri d'attaque [61, 62, 103, 137, 147]. Ces modèles sont notamment utiles pour analyser les menaces et vulnérabilités d'un système, afin d'aider leur prévention et élimination. Ils ne sont pas conçus pour évaluer l'efficacité en opération des mécanismes de protection réseau comme les contrôles d'accès et la détection d'intrusion.

Les évaluations expérimentales utilisent des données censées reproduire des activités réelles ou issues d'activités réelles, afin de vérifier la sécurité de systèmes d'un point de vue opérationnel : à un moment donné lorsqu'il est en cours d'exécution. Elles sont donc plus adaptées pour vérifier l'efficacité de mécanismes de protection en production, et ainsi déterminer de façon prédictive l'évolution du niveau de sécurité d'un

système. Un exemple répandu d'évaluation expérimentale est le test de pénétration, dont le NIST propose une méthode décomposée en quatre phases [129] :

- Planification : préparation des tests.
- Découverte : identification des réseaux, hôtes, systèmes, services et personnels, et découverte des vulnérabilités.
- Attaque : gain d'accès, escalade de privilèges, parcours de systèmes, installation d'outils.
- Report : présentation des vulnérabilités trouvées, des risques, et apport de conseils pour pallier les faiblesses.

En outre, une des difficultés concerne l'automatisation du processus d'évaluation et la génération de scénarios d'attaque pertinents pour donner des mesures représentatives du comportement des systèmes ciblés.

Dans le cadre de nos travaux, nous avons privilégié principalement les méthodes d'évaluation expérimentale pour mener des audits des mécanismes de sécurité réseau dans le cloud.

1.5 Conclusion

Dans ce chapitre, nous avons présenté les domaines associés au contexte de cette thèse : le cloud computing et la sécurité. Puis, nous avons détaillé les mécanismes de sécurité du destinés à assurer la protection des réseaux dans le cloud. Rendre ces mécanismes efficaces pour la prévention et la détection d'attaques réseau est une tâche nécessaire et complexe. En effet, les produits déployés doivent être maintenus à jour, configurés de manière adéquate et positionnés correctement dans les réseaux. De plus, les environnements cloud sont exposés à de nombreuses menaces et évoluent constamment au cours du temps. Les clients existants peuvent ajouter ou supprimer des machines virtuelles et des réseaux virtuels, ou modifier les configurations dans leurs centres de données virtuels ; de nouveaux clients peuvent souscrire à des services et créer de nouvelles infrastructures virtuelles ; les fournisseurs administrent et modifient également certains composants du cloud. Cette dynamicité peut entraîner des impacts négatifs sur la sécurité réseau du cloud. Une question principale peut alors être soulevée :

Comment donner confiance aux clients et fournisseurs de services dans la sécurité réseau du cloud ?

Nous pourrions répondre succintement à cette question : en permettant la conduite d'évaluations et d'analyses de sécurité réseau. Ainsi, il paraît essentiel de surveiller le niveau de sécurité des infrastructures cloud, afin d'adapter et améliorer le déploiement des outils de sécurité. C'est pourquoi nous avons donc également introduit les thèmes de l'évaluation et de l'analyse de la sécurité à la fin de ce chapitre. Vient alors naturellement une seconde question, à laquelle nous tentons de répondre dans cette thèse :

Comment mener efficacement des évaluations et des analyses des mécanismes de sécurité réseau dans des infrastructures virtuelles de cloud computing ?

Dans le chapitre suivant, nous présentons l'état de l'art concernant l'évaluation et l'analyse des deux mécanismes de sécurité réseau auxquels nous nous intéressons, à savoir le contrôle d'accès et la détection d'intrusion. L'objectif est de comprendre les techniques traditionnellement utilisées dans les domaines de l'analyse d'accessibilité réseau et de l'évaluation des IDS, afin de savoir comment les améliorer et les adapter au contexte du cloud. Nous terminons le chapitre suivant par l'annonce des contributions de cette thèse.

Chapitre 2

État de l'art

Sommaire

2.1 Analyse d'accessibilité réseau 31
2.1.1 Analyse statique 31
2.1.2 Analyse dynamique 33
2.1.3 Comparaison 34
2.2 Évaluation des systèmes de détection d'intrusion 34
2.2.1 Évaluation par description et analyse 35
2.2.2 Évaluation expérimentale 36
2.3 Conclusion et objectifs de la thèse 43

Il y a plusieurs approches pour évaluer la sécurité du cloud de manière générale, et il existe différentes techniques pour conduire des évaluations et des analyses de la sécurité. En revanche, peu d'entre elles se concentrent sur les mécanismes de sécurité réseau dans le cloud. Ce chapitre présente une étude synthétique de l'état de l'art dans les domaines de recherche que sont l'analyse d'accessibilité réseau et de l'évaluation des IDS, qui sont les deux domaines associés à nos contributions principales. En effet, nous sommes intéressés par les approches et outils disponibles pour mener ce type de recherche, à la fois dans les environnements traditionnels et pour les environnements de type cloud.

2.1 Analyse d'accessibilité réseau

L'analyse d'accessibilité consiste à découvrir les différents hôtes d'un système d'information et les connectivités réseau entre eux, en prenant en compte le plan d'adressage, les règles de routage, filtrage et traduction de paquets. Elle peut être menée de manière statique, approche boîte blanche avec des connaissances préalables du réseau analysé, ou dynamique, approche boîte noire sans connaissance préalable. Nous présentons ici ces deux aspects, ainsi que leurs avantages et inconvénients.

2.1.1 Analyse statique

L'analyse statique consiste à analyser les règles de routage, de filtrage et de transformation de paquets à partir de la configuration des équipements réseau. Des algorithmes sont ensuite exécutés pour déduire les accessibilités réseau (*accessibility* ou *reachability*

en anglais) tout en modélisant ces informations dans une matrice d'accessibilité ou de connectivité, ou encore dans un graphe d'accessibilité.

Dans [148], les auteurs formulent rigoureusement le problème de l'accessibilité dans les réseaux IP. Ils définissent une accessibilité entre deux points comme le sous-ensemble de paquets qu'un réseau peut transporter entre ces points. Les auteurs expliquent comment le filtrage, le routage et la transformation de paquets affectent l'accessibilité mise en place dans un réseau. Le travail présenté dans [87], portant sur le calcul de matrices d'accessibilité et l'expression de requêtes d'accessibilité (demandes visant à connaître des accessibilités) au travers d'un langage dédié, comble certains manques de [148].

De nombreuses approches, comme [49, 72, 85, 98, 101, 109, 143, 149], se focalisent spécifiquement sur l'analyse des configurations de pare-feu. Ceci est généralement réalisé pour vérifier la correcte application des politiques de sécurité au sein des pare-feu (détecter les problèmes de configuration : incohérences et redondances dans les règles), pour optimiser le nombre de règles et réduire le temps de traitement des paquets, mais aussi pour pouvoir exprimer des requêtes d'accessibilité. Un des travaux les plus précoces sur l'analyse de configuration de pare-feu est introduit dans [101]. Il s'agit d'un outil permettant de faire des requêtes d'accessibilité pour examiner la conformité de la politique de sécurité en place. Le travail décrit dans [72] présente une méthode d'analyse des règles de filtrage de routeurs Cisco en utilisant la programmation logique par contraintes avec le langage Prolog. Cela permet aux utilisateurs de spécifier des prédicats et ainsi enrichir la base d'informations sur le réseau avec leur expertise. Dans [98], les auteurs présentent un outil qui analyse les règles d'un pare-feu de type *iptables* en comparant la configuration du pare-feu avec un ensemble de requêtes d'accessibilité. À l'aide d'un moteur utilisant un diagramme décisionnel pour représenter les règles de filtrage, ceci a pour but de valider la mise en place de la configuration de pare-feu. L'outil présenté dans [143] aide, quant à lui, à la configuration et l'analyse de pare-feu distribués. Il peut recevoir les requêtes d'utilisateurs concernant des modifications et ainsi analyser les impacts potentiels d'un changement de configuration. De manière similaire, l'outil de [149] analyse des réseaux de pare-feu dans le but de détecter des violations de politiques de sécurité, erreurs de configuration ou incohérences. Pour ce faire, il modélise les règles avec des diagrammes de décisions binaires. Le travail issu de [49] permet la comparaison de la configuration d'un pare-feu avec une politique de sécurité donnée, ainsi que la vérification de la cohérence de la politique de sécurité. Par ailleurs, les auteurs de [85] présentent un modèle pour l'analyse de configuration de pare-feu. Ce modèle permet de détecter le déclenchement de règles et la cyclicité de la configuration (possibilité d'entrer dans une boucle infinie dans le traitement d'un paquet). Enfin, l'outil d'analyse de pare-feu issu de [109] permet également l'analyse de configuration (conflits, redondances, impacts de changements, etc.), mais également d'effectuer des requêtes à différents niveaux (règles, filtres, pare-feu, et réseaux de pare-feu).

Toutes ces approches sont très intéressantes, qu'elles soient centrées sur le calcul d'accessibilité, sur l'analyse en profondeur des configurations de pare-feu ou sur la vérification de politiques de sécurité. Cependant, elles sont toutes conçues pour des

architectures physiques traditionnelles, et ne supportent ainsi pas les caractéristiques spécifiques aux architectures virtuelles ou au cloud (VXLAN, pare-feu en mode pont utilisé conjointement avec des pare-feu en mode hyperviseur, groupements abstraits d'hôtes ou autres objets du réseau sur plusieurs niveaux...).

Certaines approches dédiées aux environnements cloud ont été proposées. La plus connue est probablement celle associée à l'outil d'IBM *Security Audit of Heterogeneous Virtual Environments* (SAVE) [51]. Les auteurs rassemblent des informations de bas niveau sur les composants du cloud et produisent un modèle avec quatre éléments de base : machine physique (hôte, hyperviseur, système d'exploitation et équipements physiques), machine virtuelle, stockage (disque virtuel, ressources physiques pour le stockage) et réseau (interfaces réseaux et commutateurs virtuels). Après avoir effectué les associations entre les composants, une coloration est appliquée en fonction de certaines propriétés pour distinguer les domaines et produire un modèle logique. Ce modèle montre notamment les propriétés d'isolation des ressources. Le même procédé a été adapté pour interroger la configuration du cloud public Amazon [1]. Les groupes de sécurité (ou *security groups* en anglais) rassemblent des machines virtuelles pour appliquer des actions de filtrage sur des groupes de machines du cloud. Le graphe d'accessibilité est obtenu à partir des règles de filtrages issues des configurations des groupes de sécurité. Il est utilisé conjointement avec des scans de vulnérabilité pour construire un graphe d'attaque. Toutefois leur modèle n'inclut qu'un seul pare-feu global (de type pare-feu en mode hyperviseur) et ne gère pas les topologies réseau plus complexes, en analysant les effets additionnés de toutes les règles de traitement de paquets. Du même auteur, les travaux présentés dans [52] s'intéressent à l'analyse de flots dans les infrastructures virtuelles, où les intéractions entre ressources du cloud sont représentées sous forme de graphe. Cependant, les règles de filtrage haut-niveau ne sont pas considérées. Dans [71], les auteurs proposent un système d'audit de sécurité automatisé pour le cloud, avec un langage de définition de politique de sécurité pour le cloud. Le but de ce système est de permettre l'audit automatique de machines virtuelles vis-à-vis de politiques définies par les utilisateurs. Les scénarios définis sont assez exhaustifs et permettent de modéliser les spécifications de sécurité qu'un cloud devrait satisfaire, dont les contrôles d'accès. Cependant, leur solution est très intrusive car requérant l'installation d'agents embarqués dans les éléments clés de l'infrastructure.

2.1.2 Analyse dynamique

L'analyse dynamique a également pour objectif de déterminer les accessibilités réseau. En revanche, contrairement à l'analyse statique, on cherche à réaliser une cartographie du réseau dont on ne dispose peu ou pas d'informations de configuration. Ainsi, des interrogations synthétisées par des envois de paquets sont réalisées pour découvrir les accès existants. Le domaine de l'analyse dynamique d'accessibilité a fait l'objet de peu de travaux de recherches, mais quelques outils et techniques aidant à sa mise en œuvre sont très utilisés. Il peut s'agir de simples requêtes ICMP (à l'aide

d'outils comme `ping` [19], `traceroute` [29]...) afin d'obtenir une cartographie de niveau 3, ou bien de balayages de ports réseau pour obtenir une cartographie de niveau 4 et plus. Elle ne prend en compte que les liens et règles de traitement de paquets courants, sans pouvoir anticiper les changements de topologie. Si les scanners de port (comme Nmap [15]) couvrent souvent un grand nombre de protocoles pour l'interrogation, ils peuvent aussi, de par leur nature agressive, se retrouver inhibés par des mécanismes de sécurité. Il faut aussi noter que la réalisation d'une analyse dynamique de tout le réseau peut être longue voire impossible s'il y a beaucoup trop d'hôtes. Par ailleurs, dans des environnements physiques, le déploiement de scanners dans tous les segments réseau peut se réveler parfois très fastidieux.

2.1.3 Comparaison

Bien qu'ayant le même objectif, l'analyse statique et l'analyse dynamique d'accessibilité réseau présentent des avantages et des inconvénients liés aux différentes techniques et connaissances qu'elles utilisent. Les avantages et inconvénients de chaque méthode sont synthétisés dans le Tableau 2.1.

	Avantages	**Inconvénients**
Analyse statique	- Rapide. - Peut prendre en compte et anticiper les changements de topologie. - Utilisable hors production, dans la phase de conception du réseau, en amont de son déploiement	- Difficulté à gérer la cohérence de l'ensemble de la topologie analysée. - Repose sur l'hypothèse que les mécanismes logiciels de traitement de paquets appliqués sur le réseau sont bien ceux configurés.
Analyse dynamique	- Les accès trouvés sont avérés.	- Traitement pouvant être lent si la topologie est complexe. - Techniques agressives ou intrusives donc inhibées.

Tableau 2.1 – Avantages et inconvénients des types d'analyse d'accessibilité réseau

L'analyse statique est généralement privilégiée dans les cas où on ne souhaite pas être trop intrusif sur le réseau, tandis que l'analyse dynamique est préférée pour les audits utilisant peu de connaissances préalables ou si une faible confiance est placée dans l'application des configurations par les outils qui traitent les paquets.

2.2 Évaluation des systèmes de détection d'intrusion

L'évaluation des IDS permet de valider la conception et la mise en œuvre de ces systèmes. Cela permet également de comparer différents IDS pour faire les bons choix dans leur intégration dans le système à protéger, afin de répondre aux besoins en termes de sécurité. Les systèmes de détection d'intrusion ont, comme tout système, des spécifications à respecter. Dans [84], on différencie les spécifications fonctionnelles et les

spécifications de performances. Les principales spécifications fonctionnelles attendues d'un IDS sont :

- Une continuité du service de surveillance et du rapport d'intrusion.
- Un faible taux de faux positifs pour les systèmes de détection d'anomalies.
- Une possibilité de traiter et fusionner efficacement des données de plusieurs sources distribuées.
- Une production d'informations suffisante en cas d'intrusion.
- Une capacité de réaction contre des attaques distribuées et coordonnées.
- Une tolérance aux fautes : être conçu de façon à pouvoir fonctionner dans un environnement hostile, sensible aux attaques.
- Une facilité de configuration et de modularité pour chaque hôte ou segment réseau afin de dissocier les tests.
- Une évolutivité aisée pour s'enrichir des expériences passées afin d'améliorer la détection.
- Une interopérabilité avec d'autres outils de sécurité (réseau et hôte), forensic (analyse après incident) et traitement de données.

Les principales spécifications de performances sont :

- La détection d'intrusion doit se faire en temps-réel.
- Une capacité de mise à l'échelle pour supporter des charges supplémentaires.
- Le fonctionnement de l'IDS ne doit pas interférer avec le fonctionnement du système/réseau surveillé, à savoir que les agents déployés doivent être conscients des ressources consommées.

Le problème principal dans l'évaluation des IDS est le choix de la méthode, qui est censée fournir un résultat le plus précis et complet possible sur les capacités de ces systèmes. Afin d'évaluer des IDS, deux grandes familles de méthodes se dégagent : 1) l'évaluation par description et analyse, approche plutôt boîte blanche, hors ligne, établie sur des modèles ; 2) l'évaluation expérimentale, approche plutôt boîte noire, en ligne, menée autour du test. Il est à noter que certaines approches sont hybrides. Nous sommes intéressés par l'évaluation de systèmes de détection d'intrusion en production dont nous n'avons pas forcément accès aux spécifications. Bien que nous introduisons l'évaluation par description et analyse, nous présentons surtout les principales approches expérimentales.

2.2.1 Évaluation par description et analyse

Ce type d'évaluation des IDS repose sur la modélisation du système à évaluer quelque soit son stade de développement. À ce jour et à notre connaissance, les travaux les plus complets sur l'évaluation par description et analyse sont sans doute les travaux de thèse exposés dans [38]. L'auteur propose une analyse des IDS en fonction des classes d'attaques à couvrir, dans le but de permettre une meilleure conception de ce type de système. Les attaques sont donc classées principalement en fonction des paramètres des IDS et des alarmes attendues. Un IDS est alors évalué selon ses descriptions (spécifications) et pour chaque classe d'attaque et conditions de levée d'alarmes.

Dans [76], l'auteur propose une évaluation diagnostique visant à identifier les causes de défaillances en passant par l'utilisation d'un modèle de défaillance de l'IDS. Il propose une analyse des modes de défaillances et de leurs effets (pour identifier les défaillances potentielles de chaque composant de l'IDS et donner une idée sur la cause) ainsi qu'une analyse par arbre de fautes (identifier les combinaisons de défaillances des composants qui pourraient conduire à une défaillance totale de l'IDS). Les composants identifiés dans ces études sont la sonde, le préprocesseur, le détecteur et le générateur d'alerte.

2.2.2 Évaluation expérimentale

L'évaluation expérimentale repose sur la génération et l'injection de données vers un système déployé, et l'utilisation de métriques pour mesurer la performance de ce système. Nous présentons d'abord quelques unes des principales approches dans la littérature. La robustesse de l'évaluation expérimentale reposant sur le choix des données utilisées ainsi que sur les métriques d'évaluation employées, nous détaillons ensuite ces deux éléments clés.

2.2.2.1 Principales approches

Un des tout premiers travaux sur l'évaluation expérimentale d'IDS fut introduit en 1996 dans [123]. La méthodologie présentée est un point de départ pour élaborer une stratégie d'évaluation d'IDS. Trois procédures de test sont définies : une pour mesurer la capacité de détection, une pour mesurer l'utilisation de ressources, et une pour vérifier si l'IDS fonctionne correctement dans des conditions de surcharge de trafic. Les travaux d'évaluation d'IDS par expérimentation les plus connus sont sûrement ceux menés conjointement par la *Defense Advanced Research Projects Agency* (DARPA) et le laboratoire Lincoln du MIT [94, 95]. Un grand jeu de données (environ 300 attaques différentes) a été conçu pour être injecté via des scripts. Cependant, les données ne sont plus du tout mises à jour depuis plusieurs années, et ces approches furent critiquées sous plusieurs angles dans [104]. On peut également citer le travail d'IBM Zurich [66] qui a pour but d'évaluer comparativement plusieurs IDS en utilisant une plateforme équipée de plusieurs hôtes clients et serveurs contrôlés par une seule machine. Le rapport disponible ne mentionne malheureusement que peu d'informations sur les données utilisées et les résultats obtenus.

Plus récemment, un travail sur la création de jeux de données a été fourni dans [99], où les auteurs proposent une stratégie pour répondre au manque de jeux de données pour l'évaluation d'IDS. Leur approche permet la génération de traces réseau malveillantes sur une plateforme virtuelle, et le rejeu de ces traces grâce à un outil qui vérifie la capacité de détection de l'IDS évalué. On peut également citer [42], sur la génération de trafic pour les NIDS à détection par signature à partir d'un modèle de génération de charge de travail applicatif. En outre, une méthode d'estimation de capacité de traitement des paquets y est présentée. Cependant, même si les méthodes proposées dans ces deux travaux sont intéressantes, les techniques présentées ne paraissent pas automatisables (car requérant une intervention manuelle trop importante)

et aucun jeu de données n'est publiquement accessible.

Un autre type d'approche intéressante consiste à soumettre les IDS à des attaques construites à partir de signatures d'IDS. C'est le cas dans [107], où les auteurs proposent un outil qui parcourt les signatures du NIDS Snort et génère des attaques aléatoires mais appartenant à un trafic censé être détecté par des IDS. Cependant, l'hypothèse que le trafic généré à partir des signatures d'un IDS sera réaliste et potentiellement détectable par un autre IDS est trop forte. En effet, les signatures de l'IDS doivent être suffisamment génériques, et l'outil de parcours et de génération d'attaques produisant des cas de test aléatoires, il paraît difficile de dériver du trafic malveillant réaliste pour n'importe quel IDS. De plus, une signature étant construite par expertise, il est également possible qu'elle ne couvre pas toutes les variantes d'exécution de l'attaque qu'elle est censée détecter. Ce type de problème est traité dans [100], où les auteurs proposent de modéliser le langage de signature des IDS comme une machine à états où les conditions de transitions sont des expressions logiques utilisant des prédicats, utilisant eux-mêmes des fonctions booléennes pour vérifier les caractéristiques des paquets. L'approche présentée permet de générer des attaques et ainsi vérifier la conformité des moteurs de détection sur trois niveaux : prédicats, expressions logiques, et séquences de transitions.

D'autres approches consistent à créer des attaques à partir de modèles d'exploits modifiés grâce à des variations diverses. C'est notamment le cas de [127, 136, 145], où des mutations et obfuscations sont appliquées au niveau de certains protocoles de niveau réseau ou applicatif. Un des risques est d'exécuter des exploits mutants pouvant ne plus correspondre à des attaques réalistes censées viser des cibles vulnérables.

Dans [67], les auteurs ont mené une évaluation de trois IDS commerciaux couplée à celle de Snort, NIDS en source libre. Cinq campagnes de tests différentes furent effectuées, utilisant différents outils et attaques. Le point à retenir de cette évaluation est que pour la plupart des tests, Snort égale ou dépasse les performances des IDS commerciaux.

On peut également citer le travail issu du [36], mettant en œuvre des scénarios d'attaque ciblant les applications Web, et utilisés dans une plateforme dédiée à l'évaluation d'IDS spécialisés dans la protection de serveurs applicatifs Web.

2.2.2.2 Données expérimentales

Les données expérimentales sont les données qui sont utilisées par des outils de test afin de générer des activités que les IDS évalués doivent traiter. Le choix de ces données s'avère crucial pour que les métriques d'évaluation calculées soient pertinentes. Un autre choix important est la manière de collecter et produire les données. Il peut s'agir de production de trafic synthétique reproduisant un environnement réel, ou de capture et rejeu de trafic issu d'observations réelles. Un exemple de collecte de données issues d'observations réelles sont les pots de miels [37, 60, 111], systèmes rendus volontairement accessibles aux attaquants et vulnérables afin d'enregistrer leurs activités malveillantes. Récemment, certaines travaux [47, 54, 113] ont proposé des pots de miel

déployés dans des clouds publics, afin de définir le profil des attaquants dans le cloud. Ces caractérisations pourraient permettre de déduire des données expérimentales pour le cloud. Malheureusement, à ce jour, ces données restent similaires à celles observées dans les environnements traditionnels et ne sont pas directement exploitables.

Les types de données expérimentales utilisés sont associés à deux catégories d'activités :

- Activités légitimes : activités considérées comme ne violant pas les règles ou politiques de sécurité et correspondant à un comportement normal. Elles servent à vérifier qu'un IDS ne réagit effectivement pas à ces activités.
- Activités malveillantes : activités considérées comme violant les règles ou politiques de sécurité et correspondant à un comportement anormal. Elles servent à vérifier qu'un IDS réagit effectivement à ces activités : c'est-à-dire évaluer ses taux de faux négatif et vrai positif. Il est à noter que de nombreuses techniques d'obfuscation existent afin de rendre plus difficile la détection d'activités malveillantes. Ceci permet donc d'évaluer la résistance des IDS à des attaques ciblant les algorithmes de détection.

De nombreuses bases de données, outils ou distributions sont disponibles pour générer des activités légitimes ou malveillantes à partir de données. La rétro-ingénierie de protocoles est une des méthodes associées les plus connues. Elle s'est développée dans le but d'analyser des activités afin de pouvoir les reproduire à des fins d'évaluation.

Nous exposons d'abord quelques techniques d'obfuscation d'activités malveillantes, puis revenons plus en détail sur la rétro-ingénierie de protocoles.

Techniques d'obfuscation

L'obfuscation d'attaques consiste à rendre légitimes les données manipulées par les IDS tout en conservant l'interprétation négative de ces données par la cible. Par l'utilisation de techniques d'obfuscation d'attaques, un attaquant peut faire en sorte que ses attaques ne soient pas détectées par les IDS, et on parle alors d'évasion d'intrusion [55, 68, 124]. Dans le domaine de l'évaluation expérimentale, cela sert à évaluer particulièrement le taux de faux négatifs, à savoir la proportion d'attaques non détectées. Ces techniques sont couramment embarquées dans des outils de générations d'activités malveillantes. Les plus courantes employées pour évaluer les NIDS sont les suivantes :

- Fragmentation IP : consiste à segmenter l'attaque sur plusieurs paquets afin que la signature de l'attaque ne soit plus valable par un NIDS traitant chaque paquet indépendamment.
- Chiffrement : consiste à chiffrer le contenu des paquets de l'attaque afin que seule la cible soit en mesure d'en lire le contenu pour exécuter l'attaque. Le NIDS n'est pas capable de décrypter le contenu chiffré et donc de détecter l'attaque. L'exemple le plus connu est l'utilisation du protocole HTTPS pour mener des attaques applicatives Web. On peut également citer les très répandus réseaux privés virtuels (VPN) et notamment les VPN SSL.
- Déni de service du NIDS : consiste à surcharger la sonde de détection afin d'altérer

la disponibilité et continuité du service de détection. Rendre une telle attaque distribuée (utilisation de multiples sources attaquantes), tout en usurpant les adresses sources, peut rendre l'investigation et la reprise de service encore plus complexe.

- Génération d'alertes : consiste à causer la génération d'un grand nombre de faux positifs par le NIDS dans le but de générer du "bruit" afin de cacher les vrais positifs.
- Altération du Time To Live (TTL) : consiste à insérer, entre des paquets constituant une attaque et qui ont un TTL suffisamment grand, des paquets bénins avec un TTL suffisamment petit pour atteindre le NIDS mais pas la cible de l'attaque car ils seront rejetés avant d'atteindre la cible. De cette manière, le NIDS n'a pas la même perception que le destinataire sur les données échangées. L'attaque est alors altérée pour le NIDS, qui ne la considère pas comme telle (car le flux est composé de paquets d'attaques et de paquets bénins) et ne la détecte pas bien qu'elle reste intègre pour la cible (car ne recevant que les paquets d'attaques qui ont un TTL suffisamment grand).
- L'envoi de paquets TCP de type RST avec une somme de contrôle invalide peut faire croire à un NIDS à la fin d'une communication avec la cible (qui refuse en fait le paquet de type RST), mais l'attaquant peut bien continuer à envoyer des paquets subséquents.
- L'utilisation de drapeau d'urgence (URG) dans un paquet TCP permet d'indiquer les données à être traitées directement par le destinataire. Certains NIDS ne traitant pas ce paramètre, il peut être utilisé pour ajouter des données inutiles avant la donnée pointée malveillante de manière à éviter la détection du motif malveillant lors de la lecture de l'ensemble par le NIDS.
- Polymorphisme de shellcode : permet de chiffrer des shellcode injectés dans la payload d'un paquet. Une routine de déchiffrement, différente à chaque fois, déchiffre le shellcode sur la machine cible. Le NIDS n'est ainsi pas capable de trouver la signature du shellcode qu'il voit chiffré.

Il existe également des techniques d'obfuscation d'attaques système employées pour évaluer les systèmes de détection d'intrusion système (HIDS), que nous ne détaillons pas ici étant donné que nous sommes intéressés par les mécanismes de sécurité réseau.

Rétro-ingénierie de protocoles

D'après [56], la rétro-ingénierie de protocoles est le processus d'extraction de protocoles applicatifs sans en connaître la spécification. Nous nous intéressons à la rétro-ingénierie de protocoles car une de ses applications possibles est l'évaluation de systèmes de détection d'intrusion. En effet, les protocoles inférés peuvent être utilisés pour rejouer du trafic (qui peut correspondre à des activités légitimes ou malveillantes) et simuler le comportement des applications. Il existe deux manières d'inférer des protocoles, soit en analysant le comportement interne de l'application, soit en analysant son comportement externe. Nous désirons nous focaliser sur la sécurité des réseaux, donc nous nous sommes intéressés aux approches et outils utilisant les traces réseaux pour

inférer puis rejouer des protocoles.

Récent et connu, Netzob [53] est un bon exemple d'outil permettant d'inférer des protocoles de communication à partir de trafic réseau. Il offre également un module de simulation pour générer du trafic à partir des protocoles inférés en utilisant des instances client et serveur fournies automatiquement. Ceci dit, les instances produites ne sont pas légères, ni facilement portables. Précurseur dans ce domaine, Scriptgen [93] infère de manière automatique le vocabulaire de protocoles afin de peupler les scripts utilisés dans le projet Honeyd [7]. À partir de trafic de pot miel, les séquences de messages sont analysées pour en dériver une machine à états, ensuite traduite en script. Cependant, cette approche est dédiée au projet Honeyd, qui n'est plus mis à jour publiquement et ne couvre pas les attaques récentes. De manière similaire à Scriptgen, PRISMA [88] déduit des modèles de Markov de protocoles de Botnet à partir de captures réseau et en utilisant une approche probabiliste. Les travaux de [43] traitent du problème de l'injection d'attaques, de la détermination des spécifications de protocoles jusqu'à la génération puis l'injection d'attaques. Par ailleurs, il est à noter que des outils analysant les comportements applicatifs internes proposent aussi des représentations formelles intéressantes. On peut citer en particulier Replayer [110], qui propose une définition formelle du problème de rejeu de trafic, ainsi qu'une solution utilisant l'analyse dynamique de binaire pour rejouer des dialogues applicatifs.

2.2.2.3 Métriques d'évaluation

Les métriques de bases pour l'évaluation des IDS viennent du domaine de l'apprentissage (ou *machine learning* en anglais), dans lequel les matrices de confusion permettent d'évaluer la performance d'algorithmes de classification par apprentissage. De telles matrices permettent de comparer des résultats attendus avec des résultats obtenus. Dans le domaine de l'évaluation d'IDS, on s'intéresse à comparer les activités opérées avec les réactions de l'IDS. On retrouve quatre métriques principales :

- Vrai négatif ou *True Negative* (TN) : activité légitime reconnue comme telle.
- Vrai positif ou *True Positive* (TP) : activité malveillante reconnue comme telle.
- Faux positif ou *False Positive* (FP) : activité légitime reconnue comme une activité malveillante.
- Faux négatif ou *False Negative* (FN) : activité malveillante reconnue comme une activité légitime.

Le Tableau 2.2 présente la matrice de confusion associée aux IDS.

		Réaction de l'IDS	
		Pas d'alerte	**Alerte**
Activités	**Légitimes**	Vrai négatif	Faux positif
	Malveillantes	Faux négatif	Vrai positif

Tableau 2.2 – Matrice de confusion d'IDS

À partir de ces métriques, plusieurs autres peuvent être déduites [89] :

– Taux de détection (DR) : rapport entre le nombre de vrais positifs et le nombre total de vrais positifs et faux négatifs.

$$DR = \frac{TP}{TP + FN}$$

.

– Taux de fausses alarmes (FAR) : rapport entre le nombre de faux positifs et le nombre total de faux positifs et vrais négatifs.

$$FAR = \frac{FP}{FP + TN}$$

.

– La précision (PR) : rapport entre le nombre de vrais positifs et le nombre total de vrais positifs et faux positifs.

$$PR = \frac{TP}{TP + FP}$$

.

Parfois, la notion temporelle est introduite dans des évaluations, comme c'est le cas dans [94], où les auteurs proposent l'utilisation du taux de fausses alarmes par jour au lieu du taux de fausses alarmes sur l'ensemble de la campagne d'évaluation. Un autre moyen, plus concret, permettant de visualiser l'efficacité d'un IDS, est l'utilisation des courbes de type *Receiver Operating Characteristic* (ROC) [74]. Ces courbes ont pour but de synthétiser, pour un IDS donné, la probabilité de reconnaître des activités malveillantes comme tel par rapport à la probabilité de considérer des activités légitimes comme malveillantes. Il s'agit ainsi d'exprimer, pour une activité donnée, la probabilité de générer un vrai positif en fonction de la probabilité de générer un faux positif. La probabilité de vrais positifs est équivalente au taux de détection (faculté à reconnaître des activités malveillantes comme tel), tandis que la probabilité de faux positifs est équivalente au taux de fausses alarmes (faculté à considérer des activités légitimes comme malveillantes). Ces probabilités sont bien sûr dépendantes de l'environnement de test, ce qui permet de tracer différents points établissant la correspondance entre différentes valeurs de l'un ou l'autre des deux taux associés. La Figure 2.1 montre des exemples de courbes ROC pour quatre solutions d'IDS. Elles ont été générées à partir de plusieurs campagnes d'attaques ciblant différentes classes d'attaques. Chaque point correspond donc aux valeurs du taux de détection et du taux de fausses alarmes pour un IDS donné et une classe d'attaque donnée. Plus les points sont concentrés en haut et à gauche, plus la performance de l'IDS est jugée bonne.

Il est à noter que des métriques liées à la consommation de ressources sont aussi souvent utilisées : consommation de mémoire, de CPU, de bande passante réseau, temps de traitement, etc.

Un autre ensemble de métriques proposées dans [140] est en rapport avec les coûts

d'erreurs, évalués selon trois métriques :

- Coût de réponse lors d'un faux positif : conséquence d'une réponse à une activité légitime.
- Coût de non réponse lors d'un faux négatif : conséquence d'une activité malveillante non détectée.
- Rapport entre coût de non réponse à un faux négatif et coût de réponse à un faux positif : sa valeur permet de mesurer l'impact du choix de l'un des deux paramètres de la courbe ROC.

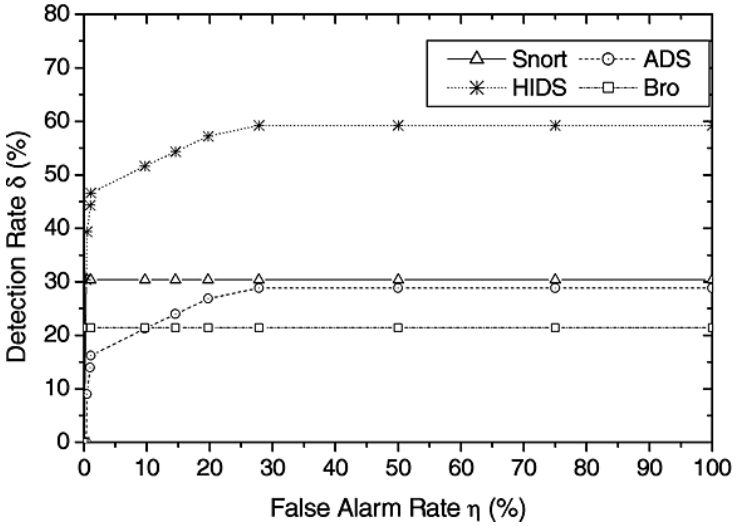


FIGURE 2.1 – Exemples de courbes ROC pour différents IDS (extrait de [80])

En fonction de ces coûts et de la probabilité d'intrusion, on peut déterminer le point opérationnel de l'IDS sur sa courbe ROC. Il s'agit du point (couple probabilité de détection/probabilité de fausses alarmes) pour lequel les coûts d'erreurs sont acceptables pour une probabilité d'intrusion donnée (pourcentage d'activités malveillantes). Par exemple, s'il y a beaucoup d'intrusions (probabilité d'intrusion élevée), il vaudra mieux détecter un maximum d'intrusions (taux de détection élevé) quitte à générer des fausses alarmes (taux de fausses alarmes élevé), car les coûts d'erreur seraient trop importants sinon. Les paramètres variables sont ici la probabilité d'intrusion et les coûts, car dépendants de l'environnement dans lequel évolue l'IDS.

D'une manière générale, les IDS sont souvent évalués en comparant les mesures obtenues avec celles concernant d'autres produits d'IDS. Par exemple, dans le cas du cloud, les solutions d'IDS sont généralement évaluées au moyen d'une comparaison de leurs performances avec les performances d'une sonde IDS seule, sur laquelle repose le système proposé (par exemple une instance de l'outil NIDS Snort). C'est le cas dans [77, 96], où les auteurs soulignent la faible dégradation des performances pour une meilleure optimisation des ressources tout en évitant d'avoir un point unique de défaillance.

Par ailleurs, il est important de dissocier la qualité intrinsèque d'un produit d'IDS de la qualité de son déploiement. En effet, un IDS peut offrir des performances plus ou moins bonnes selon l'environnement dans lequel il est installé. Ceci est à prendre en compte dans le calcul de métriques, et dans l'interprétation de leurs valeurs.

2.3 Conclusion et objectifs de la thèse

Nous avons présenté les principaux travaux associés à la problématique de cette thèse. Ils concernent deux domaines de recherche : l'analyse d'accessibilité réseau et l'évaluation des IDS. Plus particulièrement, nous avons étudié les approches pour l'analyse statique et dynamique des accessibilités, ainsi que les approches pour l'évaluation des IDS de manière expérimentale. De nombreuses contributions ont été proposées depuis plusieurs années, mais elles sont majoritairement appliquées aux infrastructures traditionnelles. En effet, peu de solutions existent à ce jour pour mener des évaluations et des analyses des mécanismes de sécurité réseau dans les infrastructures virtuelles de type cloud. Les quelques approches expérimentales associées ne considèrent que trop peu les particularités du cloud comme les nouvelles technologies proposées en matière de réseau et de sécurité, la dynamicité de tels environnements et l'automatisation des tâches. Elles ne permettent d'ailleurs pas d'évaluer des mécanismes de sécurité déployés en opération dans des infrastructures virtuelles. De plus, aucune des solutions que nous avons étudiée n'associe le contrôle d'accès réseau à la détection d'intrusion réseau, qui sont pourtant deux mécanismes généralement appliqués conjointement pour protéger les réseaux.

Nous avons donc relevé la nécessité de construire une approche globale pour évaluer les mécanismes de sécurité réseau que sont le contrôle d'accès et détection d'intrusion dans les environnements virtuels déployés dans des clouds. Afin de la rendre la plus efficace et automatisée possible, il paraît alors judicieux de concevoir une telle approche en tirant profit des méthodes traditionnelles et en utilisant les bénéfices du cloud et de la virtualisation. Les opérations d'audit étant intrusives, nous avons d'abord identifié le besoin de cloner l'infrastructure dont la sécurité réseau est à évaluer. Ensuite, nous recherchons les accessibilités réseau au sein de l'infrastructure de manière statique à partir des configurations, puis dynamique à partir d'observations réelles reposant sur des envois de paquets réseau. Ceci nous permet d'abord de vérifier la présence de déviances entre les configurations et les observations, puis de se servir des accessibilités trouvées pour exécuter des campagnes d'attaque réseau. Ces campagnes d'attaque ont pour but de vérifier l'efficacité des NIDS déployés pour protéger les infrastructures virtuelles. Ainsi, durant cette thèse, nous avons conçu et mis en œuvre une approche en trois phases permettant l'évaluation et l'analyse des mécanismes de sécurité réseau dans les infrastructures virtuelles de cloud computing. Nous proposons ainsi les quatre contributions suivantes :

– une méthode d'analyse statique des contrôles d'accès réseau dans les infrastructures virtuelles de cloud computing [121],

- une méthode d'analyse dynamique des contrôles d'accès réseau dans les infrastructures virtuelles de cloud computing [121],
- une méthode d'exécution de campagnes d'attaques dans les infrastructures virtuelles de cloud computing [122],
- un prototype expérimental mettant en œuvre les trois méthodes précédentes.

Dans le chapitre suivant, nous donnons une vue d'ensemble de l'approche que nous proposons, dont nous détaillons ensuite les contributions associées dans le reste de ce document.

Chapitre 3

Présentation générale de l'approche

Sommaire

3.1 Hypothèses principales 46
3.1.1 Environnement 46
3.1.2 Contraintes et cas d'utilisation 46
3.1.3 Conformité 47
3.2 Principes 48
3.2.1 Clonage de l'infrastructure virtuelle 49
3.2.2 Analyse des contrôles d'accès réseau 56
3.2.3 Évaluation des systèmes de détection d'intrusion réseau 57
3.3 Conclusion 57

Dans ce chapitre, nous donnons une vue d'ensemble de l'approche que nous avons développée au cours de ces travaux de thèse, et qui a donné lieu à quatre contributions principales, présentées plus en détail dans les chapitres suivants. Cette approche propose une réponse à la problématique émise à la fin du Chapitre 1. Ainsi, afin de prendre en compte diverses contraintes et de permettre une évaluation efficace des mécanismes de sécurité réseau dans les infrastructures virtuelles de cloud computing, nous avons axé notre approche autour de deux phases principales :

- Analyse des contrôles d'accès réseau.
- Évaluation des systèmes de détection d'intrusion réseau.

Ces deux phases nécessitent la réalisation d'opérations d'audit réseau assez intrusives sur les infrastructures virtuelles du clients. Leurs services et données pourraient alors être perturbés, ce qui n'est pas envisageable. C'est pourquoi nous avons décidé d'ajouter une phase préliminaire consistant à cloner l'infrastructure du client avant l'accomplissement des deux phases principales d'audit.

Ainsi, après avoir expliqué dans un premier temps les diverses hypothèses autour desquelles nous avons construit notre approche, nous introduisons son principe en trois phases, en détaillant particulièrement la première phase de clonage. Les deux phases principales de l'approche font l'objet des chapitres suivants, et sont donc décrites de manière plus succinte dans ce chapitre.

3.1 Hypothèses principales

Nous avons émis des hypothèses afin de définir le cadre conceptuel de notre approche. Ces hypothèses concernent l'environnement d'application, les cas d'utilisation et la conformité de notre approche.

3.1.1 Environnement

Notre approche ayant pour but d'évaluer et d'analyser les mécanismes de sécurité réseau déployés dans des infrastructures virtuelles, elle s'applique pour le modèle de service cloud IaaS. Dans ce modèle, les fournisseurs mettent à disposition des infrastructures virtuelles (réseaux, machines et pare-feu virtuels) aux clients.

Nous considérons les pare-feu virtuels de deux types possibles : en mode pont (contrôlés par les clients) ou en mode hyperviseur (contrôlés par les clients et les fournisseurs). De plus, nous faisons l'hypothèse que ce sont des pare-feu à états.

Concernant les NIDS, nous estimons qu'ils sont contrôlés par le fournisseur de service IaaS, et sont par défaut en charge de la détection des intrusions survenant depuis, à destination, ou au sein de tous les réseaux virtuels des clients déployés. Les NIDS doivent être déployés en prenant en compte la dynamicité du cloud afin d'assurer une continuité dans la fonction de détection d'intrusion (que ce soit la modification d'une infrastructure virtuelle existante ou le déploiement d'une nouvelle infrastructure virtuelle). De plus, nous considérons que la détection est menée à base de signatures, pour détecter les attaques connues, car nous exécutons des attaques réseau connues afin d'évaluer les NIDS dans le cadre de notre approche.

3.1.2 Contraintes et cas d'utilisation

De par son caractère expérimental, notre approche est plus adaptée pour mener des audits de sécurité réseau sur des infrastructures virtuelles de petite et moyenne taille.

Les opérations d'audit s'exécutent au nom du fournisseur de service, utilisant donc les droits d'administrateur sur l'infrastructure. Elles doivent être automatisées au maximum, à savoir requérant le minimum d'intervention humaine. Aussi, elles ne doivent pas perturber la production des clients, c'est-à-dire ne pas accéder, altérer ou divulguer les données des clients.

Les rapports d'évaluation et d'analyse fournis comprennent des résultats correspondant à l'état des systèmes évalués au moment de l'audit.

Les cas d'utilisation possibles de l'approche dépendent des contrats de type *Service-Level Agreement* (SLA) négociés entre le fournisseur et les clients. Cependant, en considérant trois acteurs différents possibles (client, fournisseur, auditeur), nous recommandons l'utilisation des fonctionnalités développées dans le cadre de l'approche comme illustré sur la Figure 3.1. Ces fonctionnalités sont l'évaluation des NIDS et l'analyse des contrôles d'accès réseau. Le client peut demander une analyse des contrôles d'accès réseau de son infrastructure virtuelle (ce qui entraîne son clonage). Le fournisseur

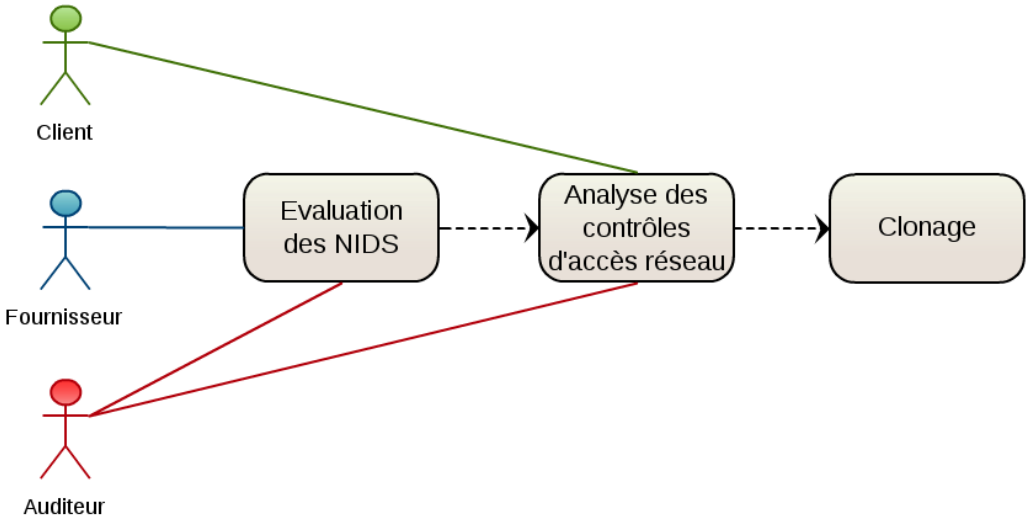


FIGURE 3.1 – Cas d'utilisation

peut conduire une évaluation des NIDS vis-à-vis de l'infrastructure d'un client (ce qui entraîne une analyse des contrôles d'accès réseau de l'infrastructure, et donc son clonage). En tant qu'acteur tiers, l'auditeur peut demander une évaluation des NIDS, ainsi qu'une analyse des contrôles d'accès réseau pour n'importe quelle infrastructure virtuelle.

3.1.3 Conformité

Les fournisseurs de services cloud devant être en mesure d'assurer un service d'évaluation de la sécurité des systèmes et réseaux aux organisations clientes, la CSA a proposé un guide de recommandations en termes d'évaluations de sécurité dans le cloud [41]. Il a pour but de définir une cartographie des domaines à considérer pour l'implémentation des évaluations. Notre approche peut aider à la mise en œuvre des recommandations de ce guide. En effet, parmi les domaines traités dans ce document, notre approche peut être utilisée dans les activités suivantes :

- Évaluation des vulnérabilités système et réseau.
- Évaluation de la conformité sécurité système et réseau.
- Évaluation de la sécurité des applications Web.
- Test de pénétration interne/externe.
- Évaluation des contrôles de sécurité.

Par ailleurs, la norme ISO/IEC 27017 [82] (en cours de rédaction et inspirée de la norme ISO/IEC 27002) vise à apporter des conseils et bonnes pratiques pour l'implémentation de contrôles de sécurité dans les services cloud. Parmi les domaines traités dans ce document, notre approche peut fournir une aide pour les thèmes suivants :

- Politiques de sécurité de l'information.

– Spécification des contrôles d'accès : politique de contrôle d'accès, accès aux réseaux et services réseau.
– Sécurité des opérations : contrôles d'audit des systèmes d'information.
– Gestion de la sécurité réseau : contrôles réseau, sécurité des services réseau, ségrégation dans les réseaux.
– Gestion des informations d'incidents de sécurité et améliorations.
– Continuité de la sécurité de l'information : vérification, revue et évaluation.

3.2 Principes

Notre objectif est de donner une réponse à la problématique énoncée à la fin du Chapitre 1 : *comment mener efficacement des évaluations et des analyses des mécanismes de sécurité réseau dans des infrastructures virtuelles de cloud computing ?* Nous voulons être ainsi capables de mettre en évidence les problèmes engendrés par un déploiement non optimal des mécanismes de sécurité. Nous désirons pouvoir mesurer l'efficacité de la mise en œuvre des contrôles d'accès réseau et de la détection d'intrusion réseau pour des infrastructures virtuelles de cloud computing. Afin de respecter les contraintes émises précedemment concernant le respect des données des clients et de ne pas perturber les services délivrés aux utilisateurs, nous choisissons de cloner l'infrastructure virtuelle impliquée dans l'évaluation. Nous menons ensuite sur ce clone une analyse des contrôles d'accès réseau avant l'évaluation des NIDS. En effet, nous avons besoin de déterminer les canaux de communication, ou accessibilités réseau, pour savoir vers quels hôtes et sur quels services nous pouvons exécuter des campagnes d'attaque afin d'évaluer la réaction des NIDS. Cela permet également de réduire la durée des campagnes d'attaque, en ne cherchant à exécuter que celles qui sont réalisables. Ainsi, notre approche est logiquement décomposée en trois phases, illustrées par la Figure 3.2.

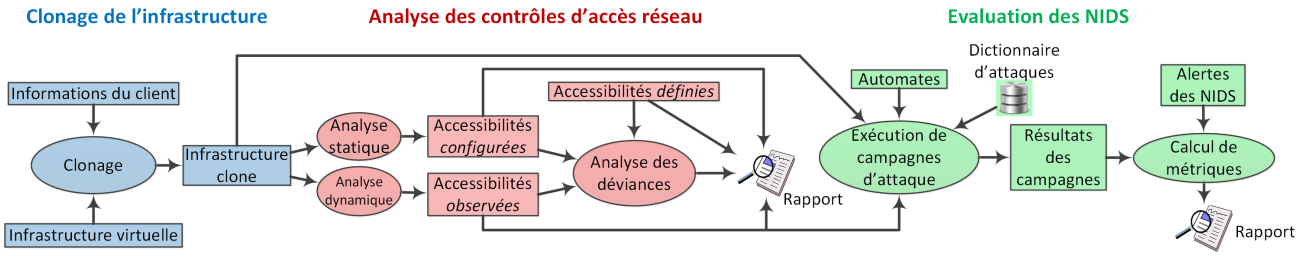


FIGURE 3.2 – Processus global de l'approche d'évaluation et d'analyse

Nous décrivons maintenant ces trois phases, en détaillant particulièrement l'étape de clonage. Les deux dernières phases sont expliquées plus brièvement car les deux chapitres suivants leurs sont dédiés.

3.2.1 Clonage de l'infrastructure virtuelle

Dans le but principal de protéger la production du client associé, l'infrastructure dont la protection réseau est à évaluer n'est pas utilisée directement. En effet, pour réaliser les audits, nous utilisons un clone de l'infrastructure. Cependant, dans le processus de clonage, nous avons choisi de ne pas copier les machines virtuelles du client pour les raisons suivantes :

- Nous ne voulons pas accéder aux données des machines virtuelles du client pour préserver la confidentialité de ses données.
- Nous ne sommes pas intéressés par l'évaluation de la sécurité des machines virtuelles, mais par l'évaluation des mécanismes de sécurité réseau.
- Les machines virtuelles de l'infrastructure clone doivent être équipées des outils nécessaires pour réaliser les opérations d'audit réseau requises par notre approche.
- Nous avons besoin d'un accès aux machines virtuelles de l'infrastructure clone pour les manipuler afin d'effectuer les opérations d'audit réseau.

C'est pourquoi nous avons préféré copier principalement la configuration réseau et les pare-feu virtuels de l'infrastructure initiale. Dans la nouvelle infrastructure clone, les machines virtuelles initiales sont alors remplacées par des machines virtuelles personnalisées, importées d'un modèle [1] de machine virtuelle, ou *virtual machine template*, que nous avons construit au préalable. Il s'agit d'un modèle léger consommant peu de ressources car uniquement équipé des outils d'audit nécessaires. L'utilisation d'un modèle léger nous permet également d'espérer un gain de ressources requises pour le clonage, par rapport à un clonage où on copierait toutes les machines virtuelles de l'infrastructure. Les machines virtuelles importées dans l'infrastructure clone sont configurées en réseau de la même manière que les machines virtuelles initiales. Il est à noter que bien que cela ait pour but de copier une infrastructure virtuelle donnée, la création d'un clone d'infrastructure peut aussi être assimilé à l'arrivée d'un nouveau client IaaS dans le cloud. C'est d'ailleurs pour le compte d'une organisation cliente fictive, créée spécifiquement pour les évaluations de sécurité, que l'infrastructure clone est installée.

L'organisation des composants d'un cloud est structurée hiérarchiquement en suivant la représentation illustrée par la Figure 3.3. Cette représentation est inspirée de la terminologie de VMware en matière de cloud computing [45]. Nous l'avons suivie car elle prend en compte les notions importantes du cloud, ce qui la rend suffisamment générique pour être utilisée dans l'implémentation de prototypes dédiés à d'autres solutions de cloud. Pour réaliser le clonage d'une infrastructure virtuelle, nous devons également procéder de manière hiérarchique, sans quoi certains éléments pourraient ne pas être recréés ou manqueraient d'informations de configuration provenant d'autres éléments censés exister. En effet, il faut recréer dans l'ordre les composants accueillant d'autres composants, en commençant par ceux en accueillant le plus. Le processus de clonage se résume ainsi :

1. Le modèle utilisé dans le prototype est détaillé dans le Chapitre 6.

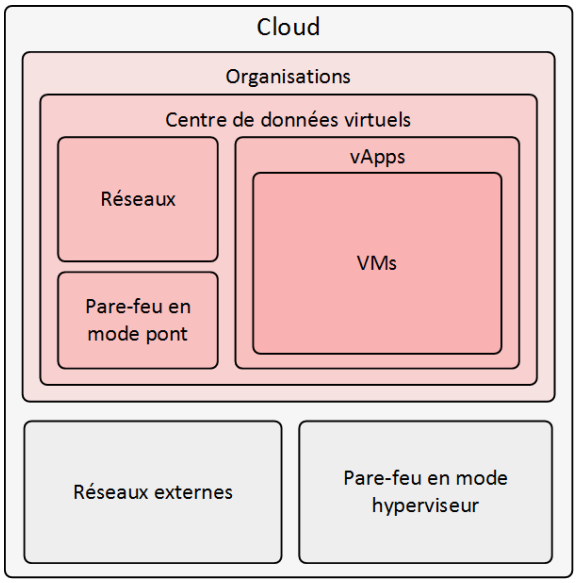


FIGURE 3.3 – Vue hiérarchique des composants d'un cloud

1. Récupération des informations de configuration des centres de données virtuels du client : plan d'adressage, connexions réseau, configuration des pare-feu, etc.
2. À partir des informations de configuration, création des copies de centres de données virtuels, prêts à accueillir des nouveaux réseaux, pare-feu et machines virtuelles.
3. À partir des informations de configuration, création des copies de réseaux virtuels et pare-feu dans les centres de données virtuels.
4. À partir des informations de configuration et de notre modèle, importation des nouvelles machines virtuelles dans les centres de données virtuels.
5. À partir des informations de configuration, mise à jour de la configuration des pare-feu en mode hyperviseur.

Le clonage d'une infrastructure virtuelle comprend donc quatre étapes principales : le clonage des centres de données virtuels, le clonage des réseaux virtuels et pare-feu virtuels en mode pont, l'importation des machines virtuelles et la mise à jour des pare-feu en mode hyperviseur.

3.2.1.1 Clonage des centres de données virtuels

La première étape du processus de clonage consiste à créer de nouveaux centres de données virtuels (vDC) pour l'organisation cliente fictive utilisée pour l'évaluation. Le Tableau 3.1 reprend les principales informations à récupérer auprès du centre de données initial pour recréer un centre de données virtuel clone.

Information	Description
Capacité de stockage	Unité, quantité allouée, limite, quantité utilisée, quantité de surcharge.
Capacité de calcul	Unité, quantité allouée, limite, quantité utilisée, quantité de surcharge.
Capacité de mémoire	Unité, quantité allouée, limite, quantité utilisée, quantité de surcharge.
Quota de NIC	Nombre maximum de cartes réseau virtuelles autorisées dans ce vDC.
Quota réseau	Nombre maximum d'objets réseau autorisés dans ce vDC.
Quota de VM	Nombre maximum de VMs autorisées dans ce vDC.
Activation	vDC activé pour l'utilisation ou non.
Ressources mémoire garanties	Quantité de mémoire garantie dans ce vDC.
Ressources de calcul garanties	Quantité de calcul garantie dans ce vDC.
CPU virtuel par VM	CPU virtuel automatiquement alloué à chaque VM.

Tableau 3.1 – Informations de configuration requises pour le clonage d'un centre de données virtuel

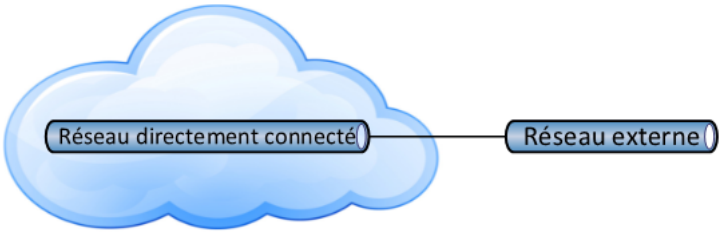
3.2.1.2 Clonage des réseaux virtuels et pare-feu en mode pont

Au sein des centres de données virtuels, on retrouve les réseaux et pare-feu virtuels. Il faut les cloner avant d'importer les machines virtuelles, afin de pouvoir connecter celles-ci correctement. Les réseaux virtuels internes aux centres de données virtuels sont administrés par les clients. Les réseaux dit externes sont les réseaux interconnectant les réseaux virtuels internes des clients, et sont donc administrés par le fournisseur de service IaaS. D'après [45], il y a trois types de réseaux virtuels internes possibles dans les centres de données virtuels :

- Les réseaux directement connectés (Figure 3.4a) : réseaux connectés à un réseau externe par un pont de niveau 2, assurant donc la connectivité avec des hôtes externes. Les hôtes d'un réseau directement connecté sont donc dans le réseau externe.
- Les réseaux isolés (Figure 3.4b) : réseaux non connectés à un pare-feu en mode pont, n'assurant donc une connectivité qu'entre les machines virtuelles de ce réseau.
- Les réseaux routés (Figure 3.4c) : réseaux connectés à un pare-feu en mode pont, lui même connecté à un réseau externe et assurant donc une connectivité avec des hôtes externes.

Clonage des réseaux virtuels directement connectés

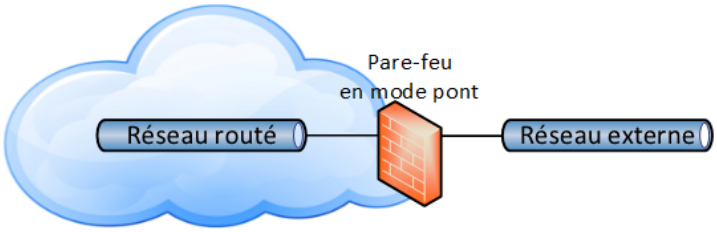
Le clonage de ce type de réseau ne requiert que de récupérer le nom et la configuration du réseau afin d'en recréer un à l'identique. Le réseau étant directement connecté à un réseau externe, il partage les plages d'adresses IP de celui-ci. Cependant, aucun hôte (pare-feu ou machine virtuelle) n'étant cloné dans ce cas, il n'y a pas de conflit d'adresse IP possible dû à l'ajout d'un hôte clone dans le réseau.



(a) Réseau virtuel directement connecté



(b) Réseau virtuel isolé



(c) Réseau virtuel routé

FIGURE 3.4 – Types de réseaux virtuels

Clonage des réseaux virtuels isolés

Le clonage de ce type de réseau ne requiert que de récupérer le nom et la configuration du réseau afin d'en recréer un à l'identique. Le réseau étant isolé et aucun hôte n'étant cloné, il n'y a pas de conflits d'adresses IP possibles avec le clone.

Clonage des réseaux virtuels routés

Le clonage de ce type de réseau est plus complexe. La première étape consiste à récupérer les informations de configuration du réseau, ainsi que celles du pare-feu en mode pont associé. Le pare-feu en mode pont cloné a, tout comme le pare-feu en mode pont original correspondant, au moins une interface réseau virtuelle connectée sur un réseau externe (interfaces de type liaison montante ou *uplink*). Ainsi, une fois cloné, il peut se retrouver en conflit d'adresse IP avec les autres hôtes (notamment les autres pare-feu en mode pont) qui sont déjà connectés sur ce même réseau externe. L'Algorithme 1 a ainsi pour but de rechercher, pour chaque interface de liaison montante et parmi les adresses IP disponibles (non utilisées par d'autres hôtes) dans le même sous-réseau, l'adresse IP dont le numéro d'hôte est le plus proche. Cet algorithme est exécuté jusqu'à ce que les adresses IP trouvées soient utilisables par le clone (un numéro de tentative est à incrémenter et est utilisé dans l'algorithme). Si aucune adresse

Algorithme 1 Recherche d'adresses IP avec évitement de conflits pour la configuration d'un clone de pare-feu en mode pont

Require: $conf$: configuration du pare-feu en mode pont
Require: $reserved_ips$: adresses IP inutilisables
Require: n : numéro de tentative de configuration, incrémenté après chaque échec d'exécution de l'algorithme si conflit
Ensure: $ip_addresses$: dictionnaire contenant pour chaque adresse IP originale du pare-feu en mode pont l'adresse IP utilisable par le clone

```
ip_addresses ← ∅
for int ∈ get_interfaces(conf) do
    if get_type(int) == "uplink" then
        ip_address ← get_ip_address(int)
        netmask ← get_netmask(int)
        gateway_ip_address ← get_gateway_ip_address(int)
        available_ips ← get_ips_of_network(ip_address, netmask)
        available_ips ← available_ips \ ip_address
        available_ips ← available_ips \ gateway_ip_address
        available_ips ← available_ips \ reserved_ip
        if available_ips == ∅ then
            exit()
        else
            available_ips ← sort_by_closest_addresses(available_ips, ip_address)
        end if
        ip_addresses[ip_address] ← get_closest_available_ip(ip_address, available_ips, n)
    end if
end for
return ip_addresses
```

IP n'est disponible, le clonage ne peut être effectué et il faudrait alors libérer une adresse IP le temps de l'audit, pour pouvoir cloner le pare-feu.

La deuxième étape consiste à recréer le réseau virtuel routé en l'associant au pare-feu en mode pont cloné. Enfin, une fois le réseau virtuel routé et son pare-feu en mode pont cloné, il faut mettre à jour la configuration du pare-feu en mode pont. En effet, la configuration ayant été copiée du pare-feu en mode pont original, des références (réseaux, adresses IP...) peuvent désormais être obsolètes pour le pare-feu cloné. Pour ce faire, il suffit de mémoriser tous les changements effectués via l'Algorithme 1, et de répercuter ces changements dans la configuration de chaque pare-feu en mode pont.

3.2.1.3 Importation des machines virtuelles

La dernière étape du clonage consiste à importer les machines virtuelles dans l'infrastructure clone, à partir d'un modèle que nous avons prédéfini. En partant du principe que les machines virtuelles sont regroupées en entités de plus haut niveau (appelées vApp dans la terminologie VMware), l'objectif est alors de recréer la même structure d'entités et de reproduire les mêmes configurations réseau pour les machines virtuelles, tout en remplaçant le contenu des machines virtuelles par celui de notre modèle. Le processus de clonage d'un vApp, illustré par la Figure 3.5 (où les étapes sont identifiées par les mêmes numéros), est le suivant :

1. Récupération de la configuration du vApp original.

2. Récupération de la configuration de chaque machine virtuelle du vApp original.
3. Pour chaque machine virtuelle du vApp original, importation d'une machine virtuelle en tant que vApp à partir du modèle de machine virtuelle (une fois importée, une machine virtuelle seule constitue automatiquement un vApp).
4. Composition d'un nouveau vApp à partir des nouveaux vApp créés, de la configuration du vApp original et de la configuration des machines virtuelles originales.
5. Démarrage des machines virtuelles du nouveau vApp créé.

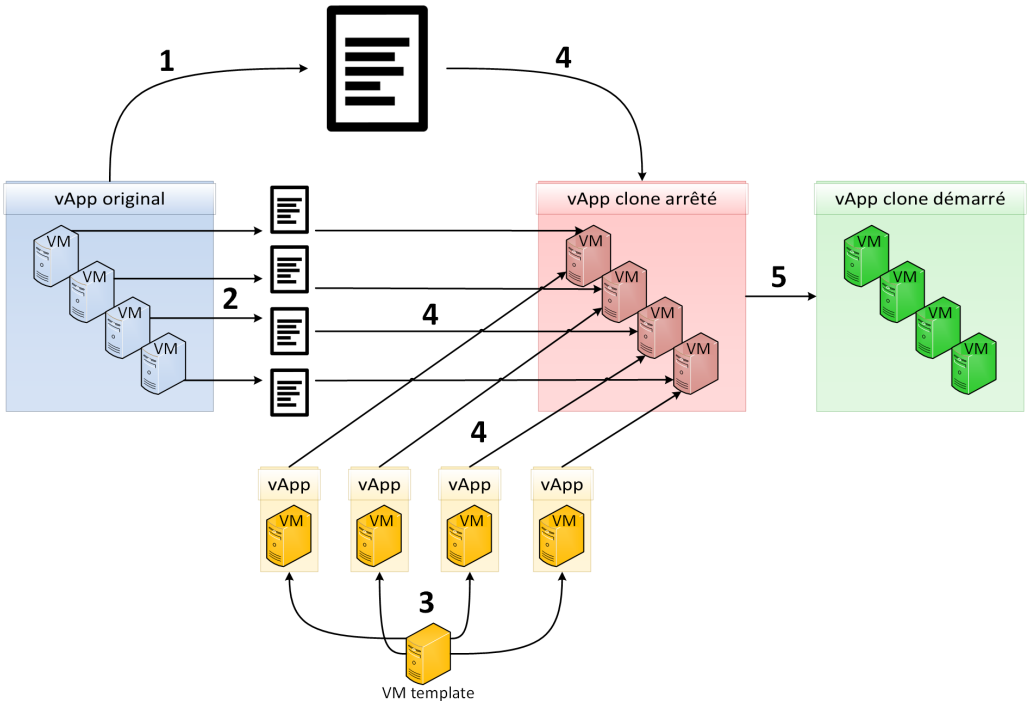


FIGURE 3.5 – Processus d'importation des machines virtuelles

Il est à noter que l'Algorithme 1 est réutilisé durant l'étape 4 du processus de clonage d'un vApp afin d'éviter les conflits d'adresses IP pour les machines virtuelles importées dans des réseaux virtuels directement connectés. En effet, elles peuvent alors être en conflit d'adresse IP dans le réseau externe, et il est donc nécessaire d'appliquer le même procédé de recherche d'adresses IP disponibles, comme lors du clonage des pare-feu en mode pont.

3.2.1.4 Mise à jour des pare-feu en mode hyperviseur

Les pare-feu en mode hypeviseur étant déjà déployés sur les hyperviseurs, il n'est pas nécessaire de les cloner. Il suffit de mettre à jour leur configuration en ajoutant les règles de filtrage globales, et celles spécifiques aux réseaux virtuels clonés. Concrètement il s'agit donc d'une copie de la configuration du filtrage associé aux réseaux

originaux, en tenant compte des éventuels changements d'adresses IP opérés pour éviter des conflits d'adresse IP.

3.2.1.5 Ressources nécessaires pour le clonage

Il est possible d'exprimer la quantité de ressources (CPU, mémoire, disque) nécessaire à allouer pour réaliser le clonage d'une infrastructure virtuelle. Notons N_m le nombre de machines virtuelles dans l'infrastructure à cloner, R_m^i les ressources utilisées par une machine virtuelle i, R_t les ressources utilisées par le déploiement du modèle de machine virtuelle, N_f le nombre de pare-feu en mode pont dans l'infrastructure à cloner, et R_f^i les ressources utilisées par un pare-feu en mode pont i.
La quantité R_a de ressources allouées à une infrastructure virtuelle correspond à la somme des quantités de ressources allouées aux machines virtuelles et pare-feu virtuels en mode pont, soit :

$$R_a = \sum_{0<i<N_m} R_m^i + \sum_{0<i<N_f} R_f^i$$

La quantité R_c de ressources requise pour son clonage correspond à la somme des quantités de ressources allouées aux pare-feu virtuels en mode pont et de la quantité de ressources allouées au déploiement du modèle de machine virtuelle multipliée par le nombre de machines virtuelles, soit :

$$R_c = N_m \times R_t + \sum_{0<i<N_f} R_f^i$$

Ainsi, le gain G de ressources espéré pour le clonage grâce à notre technique utilisant l'importation d'un modèle léger, par rapport à un clonage incluant la copie des machines virtuelles de l'infrastructure, est exprimé par :

$$G = R_c - R_a = N_m \times R_t - \sum_{0<i<N_m} R_m^i$$

A titre d'exemple, considérons une infrastructure virtuelle avec dix machines virtuelles utilisant chacune 2 CPUs, 4 Go de mémoire et 40 Go de disque ; deux pare-feu en mode pont utilisant chacun 1 CPU, 256 Mo de mémoire et 320 Mo de disque ; un pare-feu en mode pont utilisant 2 CPU, 1 Go de mémoire et 320 Mo de disque. Au total, cette infrastructure requiert un total de 24 CPUs, 41,5 Go de mémoire et 400,96 Go de disque. En considérant un modèle de machine virtuelle avec 1 CPU, 1 Go de mémoire et 20 Go de disque, la quantité de ressources requise pour le clonage est de 14 CPUs, 11,5 Go de mémoire et 200,96 Go de disque. Le gain est alors de 10 CPUs, 30 Go de mémoire et 200 Go de disque. En considérant un autre modèle de machine virtuelle avec 1 CPU, 512 Mo de mémoire et 8 Go de disque, la quantité de ressources requise pour le clonage est de 14 CPUs, 6,5 Go de mémoire et 80,96 Go de disque. Le gain est alors de 10 CPUs, 35 Go de mémoire et 320 Go de disque.

A la fin de l'audit, l'infrastructure clone est entièrement supprimée et les ressources utilisées pour le clonage sont ainsi libérées. Le processsus de suppression d'une infra-

structure virtuelle suit l'ordre hiérarchique inverse du processus de clonage :

1. Suppression des vApp des centres de données virtuels.
2. Suppression des réseaux virtuels et pare-feu en mode pont des centres de données virtuels.
3. Suppression des centres de données virtuels.

3.2.2 Analyse des contrôles d'accès réseau

Après avoir cloné l'infrastructure du client, il serait inefficace de lancer directement une campagne d'évaluation des systèmes de détection d'intrusion de façon désordonnée. En effet, il est plus judicieux de chercher en premier lieu les canaux de communication autorisés afin de guider l'exécution des attaques et ainsi ne lancer que celles sur des services réseau associés à des accessibilités. C'est pourquoi la deuxième phase de l'approche consiste à déterminer les accessibilités réseau de bout en bout (entre machines virtuelles de l'infrastructure, et entre machines virtuelles de l'infrastructure et réseaux externes). Des erreurs pouvant survenir dans la définition, la configuration ou l'implémentation des contrôles d'accès réseau, il est intéressant de les examiner à différents niveaux afin de trouver d'éventuelles défaillances dans leur mise en œuvre. Nous avons donc choisi d'analyser de deux manières différentes les contrôles d'accès réseau, en utilisant des techniques différentes et sans la même connaissance initiale. Nous proposons ainsi une méthode d'analyse statique ainsi qu'une méthode d'analyse dynamique, toutes les deux adaptées aux environnements virtuels de cloud computing.

L'analyse statique consiste d'abord à parcourir la configuration du cloud (configuration des machines virtuelles, réseaux et pare-feu) pour transformer les informations de configuration en prédicats. Ces prédicats des configurations sont ensuite utilisés en tant qu'entrées dans un moteur logique exécutant un algorithme déterminant les accessibilités possibles entre toutes les machines virtuelles de l'infrastructure, ainsi que depuis et vers les réseaux externes à cette infrastructure.

L'analyse dynamique consiste à effectuer des échanges de paquets réseau entre les machines virtuelles, en suivant un algorithme permettant d'exécuter autant d'échanges que possible en parallèle. Ceci a pour but de déterminer le plus rapidement possible les accessibilités entre toutes les machines virtuelles de l'infrastructure, ainsi que depuis et vers les réseaux externes à cette infrastructure.

Les accessibilités trouvées par chaque méthode sont modélisées sous forme de matrices, appelées matrices d'accessibilité. Elles sont comparées entre elles, et également avec les accessibilités définies initialement par l'utilisateur [2], dans le but de trouver et reporter d'éventuelles différences dans les résultats. En effet, il est important pour les clients de mettre en évidence les imprévus pouvant survenir lors des différentes étapes de définition et d'implémentation des contrôles d'accès réseau. Ceci doit permettre d'améliorer la définition des politiques de sécurité et l'exploitation des outils de sécurité.

2. La définition et la récupération d'une politique de sécurité réseau sont hors du cadre de nos contributions.

3.2.3 Évaluation des systèmes de détection d'intrusion réseau

La troisième et dernière phase de l'approche concerne l'évaluation des systèmes de détection d'intrusion réseau. À partir des accessibilités réseau découvertes, le but est d'exécuter des campagnes d'attaque et d'étudier la réaction des systèmes de détection d'intrusion vis-à-vis de ces attaques. Cela requiert une démarche méthodique permettant d'organiser les campagnes d'attaque, de les orchestrer de manière automatisée et d'optimiser leur durée. Pour ce faire, nous avons établi une méthode permettant de mener des campagnes d'attaque grâce à un algorithme reposant sur trois éléments :

- Les accessibilités réseau sous forme de matrice d'accessibilité : les flux autorisés dérivés de l'analyse dynamique des contrôles d'accès réseau. Nous prenons ici les résultats de l'analyse dynamique car ils sont issus d'envois de paquets révélant les accessibilités effectives.
- Un dictionnaire d'attaque : une base de données contenant les informations nécessaires sur les attaques à exécuter.
- Un ensemble de données de trafic légitime et malveillant représentant l'utilisation légitime et malveillante d'applications que nous avons modélisées et générées, et que nous sommes capables de rejouer à l'aide de plusieurs outils.

Notre méthode d'exécution de campagnes d'attaque propose plusieurs paramètres, de manière à pouvoir adapter la réalisation de l'audit des NIDS en fonction de l'environnement d'exécution et de critères d'évaluation.

Après l'exécution des campagnes d'attaque, les alertes émises par les NIDS sont analysées et confrontées aux attaques générées afin de calculer des métriques d'évaluation. L'objectif est d'aider à la mise en évidence d'un déploiement inapproprié (positionnement réseau et configuration) possible des systèmes de détection d'intrusion.

3.3 Conclusion

Nous avons présenté une vue d'ensemble de l'approche que nous proposons pour l'évaluation et l'analyse des mécanismes de sécurité réseau dans les infrastructures virtuelles de cloud computing. Cette approche est construite autour de plusieurs hypothèses importantes ayant permis d'établir un cadre de conception rigoureux. Afin de fournir une démarche d'audit méthodique, nous avons décomposé l'approche en trois phases : clonage de l'infrastructure du client, analyse des contrôles d'accès réseau, évaluation des NIDS. L'objectif est de fournir des rapports d'évaluation permettant d'aider les clients et fournisseurs de service à améliorer le déploiement des mécanismes de sécurité réseau. Tout en minimisant l'impact sur la production des clients, l'intégralité du processus d'audit est automatisé.

Nous avons également détaillé la première étape de notre approche, à savoir une technique de clonage d'infrastructures virtuelles, qui est indispensable pour la bonne réalisation des opérations d'audit réseau. Nous nous sommes servis de la terminologie VMware pour mettre au point un processus prenant en compte tous les composants d'une infrastructure cliente. Par le déploiement d'un modèle de machine virtuelle dans

l'infrastructure clone, notre méthode permet de préserver la confidentialité des données du client et d'économiser des ressources par rapport à un clonage intégral des machines virtuelles. Cela permet également de déployer les outils nécessaires aux opérations d'audit au sein des machines virtuelles de l'infrastructure clone.

Enfin, nous avons introduit les deux phases suivantes de notre approche, concernant l'analyse des contrôles d'accès réseau et l'évaluation des NIDS. Dans les deux chapitres suivants, nous détaillons les méthodes développées pour ces étapes.

Chapitre 4

Analyse des contrôles d'accès réseau

Sommaire

4.1 Accessibilités 59
4.2 Analyse statique 61
4.2.1 Parcours des configurations 61
4.2.2 Résolution logique des accessibilités 62
4.3 Analyse dynamique 66
4.3.1 Principes 67
4.3.2 Complexité 69
4.4 Analyse des déviances 70
4.5 Conclusion 71

Ce chapitre présente la deuxième phase de notre approche d'évaluation et d'analyse des mécanismes de sécurité réseau dans le cloud, portée sur l'analyse des contrôles d'accès réseau. Elle est appliquée après la phase de clonage de l'infrastructure virtuelle à évaluer, que nous avons décrite dans le chapitre précédent. A présent, le but est de déterminer les accessibilités réseau à partir des différents moyens à disposition : en analysant la configuration du cloud et en injectant du trafic réseau. La découverte de ces accessibilités est essentielle pour savoir sur quels canaux de communication exécuter des campagnes d'attaque lors de l'évaluation des NIDS. En outre, la comparaison des résultats obtenus permet de souligner d'éventuelles déviances entre les configurations et les observations réelles. La mise en œuvre de cette phase a donné lieu à deux contributions : une méthode d'analyse statique et une méthode d'analyse dynamique des contrôles d'accès réseau. Nous introduisons d'abord les notions importantes sur les accessibilités, puis détaillons ces deux contributions ainsi que l'analyse des déviances.

4.1 Accessibilités

Une accessibilité réseau est définie comme un accès autorisé à un service d'une source vers une destination, où un service est l'association d'un protocole et d'un port réseau. L'ensemble des accessibilités est modélisé par une matrice d'accessibilité. La première dimension contient les machines virtuelles sources (et non une adresse IP, car

d'un point de vue utilisateur lorsqu'un hôte a plusieurs adresses IP, on ne sait pas nécessairement à l'avance quelle sera l'adresse source d'une communication) ; la deuxième dimension contient les adresses IP destinations (et non une machine virtuelle, car les communications ne sont pas nécessairement contrôlées de la même manière à destination d'une machine ayant plusieurs adresses IP). Le Tableau 4.1 montre un exemple de matrice d'accessiblité, où trois machines virtuelles (deux avec une adresse IP et une avec deux adresses IP) et quatre services sont considérés. Une matrice d'accessibilité est aussi la matrice d'incidence d'un graphe d'accessibilité, qui permet de représenter graphiquement les accessiblités réseau.

		Destinations			
		IP A1	**IP B1**	**IP B2**	**IP C1**
Sources	**VM A**				Service W
	VM B	Service X			Service Y
	VM C	Service Z			

Tableau 4.1 – Exemple de matrice d'accessibilité

Ce genre de matrice est habituellement spécifié dans une politique de sécurité afin d'y indiquer les accessibilités. Ces accessibilités sont appelées les accessibilités *définies*, car elles sont issues de la définition de la politique de sécurité. Elles sont ensuite généralement déclinées en matrices des flux autorisés et bloqués pour chaque équipement gérant le filtrage de paquets. Enfin, le contrôle des flux autorisés et bloqués est mis en œuvre sous forme de règles de filtrage de paquets sur les équipements.

Afin de retrouver les accessibilités réseau en place au sein d'une infrastructure, nous avons vu dans le Chapitre 2 qu'il existe deux manières de procéder : statiquement ou dynamiquement. L'analyse statique utilise les configurations des composants pour déterminer les accessibilités. Ces accessibilités sont appelées les accessibilités *configurées*, car elles sont issues de la configuration du cloud. L'analyse dynamique repose sur l'injection de trafic réseau pour trouver les accessibilités. Ces accessibilités sont appelées les accessibilités *observées*, car elles sont issues d'observations réelles dans le cloud. Dans notre approche, nous proposons de mener les deux types d'analyse. En effet, la confrontation des résultats permet de mettre en évidence les déviances pouvant résulter des problèmes, incohérences ou oublis dans les définitions, les configurations ou les implémentations des contrôles d'accès réseau. Une déviance est donc définie comme une accessibilité non présente à la fois dans la matrice des accessibilités *définies*, dans la matrice des accessibilités *configurées* et dans la matrice des accessibilités *observées*.

Il est à noter que l'on se placera toujours au niveau 3 du modèle OSI (couche IP) pour caractériser une accessibilité. Les connexions virtuelles de niveau 2 sont vérifiées dans l'analyse statique en tant que prédicats de connexions, et implicitement vérifiées dans l'analyse dynamique par les envois de paquets IP. En revanche, on ne considère pas le filtrage de niveau 2 (pouvant être réalisé par les commutateurs virtuels par exemple), ni les pare-feu potentiellement présents au sein d'une machine virtuelle. Bien que cela consisterait simplement à étendre nos modèles et algorithmes, ce type

de filtrage ne fait pas partie des mécanismes de sécurité réseau évalués.

4.2 Analyse statique

Pour construire la matrice d'accessibilité de manière statique, il faut prendre en compte tous les composants du cloud qui impactent le traitement des communications réseau. Il est donc naturel de considérer les configurations de tous ces composants. Il faut ainsi déduire des configurations du cloud toutes les accessibilités 1) des machines virtuelles vers les adresses IP des autres machines virtuelles, 2) des machines virtuelles vers les réseaux externes (représentés par l'adresse 0.0.0.0) et 3) des réseaux externes vers les machines virtuelles. Nous modélisons la déduction d'une accessibilité sous forme de prédicat d'accessibilité :

$$accessibility(X, SPORT, Y, DPROTO, DPORT)$$

Ce prédicat signifie qu'il existe une accessibilité de X, sur le port source $SPORT$, vers Y, sur le protocole destination $DPROTO$ et le port destination $DPORT$.

Afin de générer ces prédicats d'accessibilité à partir des configurations du cloud, il faut prendre en compte la topologie réseau ainsi que les actions de routage, modification et filtrage de paquets. Il faut considérer les intéractions au sein d'un pare-feu (une règle pouvant en annuler une autre par exemple) et à travers la topologie (pour mémoriser les actions effectuées sur les paquets). En outre, les concepts de réseau et de sécurité du cloud computing utilisent des notions comme les groupes (d'adresses ou d'objets), les services (association de protocoles et ports) et groupes de services, qui sont à prendre en compte dans l'analyse statique des règles.

La méthode d'analyse statique se décompose alors logiquement en deux étapes :

1. La récupération des informations de configuration.
2. La génération des prédicats d'accessibilité (et de fait la matrice d'accessibilité).

L'outil d'analyse statique que nous avons conçu est donc composé de deux modules principaux : un parser de configuration et un moteur logique. Le parser de configuration extrait les informations des configurations de chaque composant du cloud et les traduit en prédicats des configurations à l'aide de feuilles de styles. Puis, sur soumission de requêtes d'accessibilité, le moteur logique utilise les prédicats des configurations dans des règles logiques afin de générer les prédicats d'accessibilité. La Figure 4.1 montre les intéractions entre les modules.

4.2.1 Parcours des configurations

Le parser de configuration a pour rôle de récupérer les informations de configuration du cloud, dans un format spécifique tel que XML. Il est donc spécifique à un produit cloud puisqu'il en interroge les APIs. Les composants à partir desquels le parser récupère ces informations de configuration sont les modules de gestion du cloud,

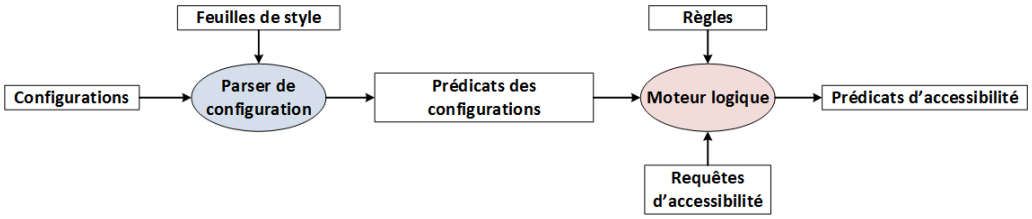


FIGURE 4.1 – Intéractions entre les modules d'analyse statique

les pare-feu et les machines virtuelles. Il est à noter que des routes statiques à destination des réseaux virtuels routés sont ajoutées sur la machine appartenant aux réseaux externes, afin de pouvoir effectuer les tests d'accessibilité (statiquement puis dynamiquement) depuis et vers cette machine. Les informations de configuration récupérées sont ensuite transformées en prédicats Prolog utilisés par un moteur logique Prolog. Nous avons choisi le Prolog car c'est un langage logique bien adapté pour exprimer l'existence de règles, objets, relations, etc. Il permet aussi de répondre facilement à une série de questions binaires dans le but de déterminer une réponse finale d'accessibilité, comme nous allons le voir plus loin. Le Tableau 4.2 répertorie les prédicats pouvant être générés à l'aide du parser de configuration, et le Tableau 4.3 donne des exemples pour chacun de ces prédicats.

La transformation des informations de leur format initial vers du Prolog est réalisée à l'aide de feuilles de style. Une feuille de style permet de décrire, à l'aide d'un langage, la transformation d'un format vers un autre. Ici, on s'intéresse à la transformation de XML vers Prolog. Le langage que nous avons choisi pour faire de telles transformations est *EXstensible Stylesheet Language Transformations* (XSLT), dont les feuilles de styles sont d'ailleurs elles-mêmes écrites en XML. Chaque balise XML des documents contenant les informations de configuration est analysée et leur contenu est récupéré puis inséré dans un prédicat Prolog.

4.2.2 Résolution logique des accessibilités

Après le parcours des configurations et la génération des prédicats de configuration, l'objectif est de mettre en relation ces prédicats afin de déterminer les accessibilités réseau. La détermination d'une accessibilité réseau entre deux hôtes consiste à répondre à la question initiale suivante : *y a-t-il accessibilité entre deux hôtes pour un service donné ?* Cette question est elle-même décomposée en trois questions principales :

1. *Y a-t-il une route réseau entre les deux hôtes ?*
2. *Le trafic associé aux hôtes et service donnés (ou une modification de ce trafic par traduction d'adresse) est-il autorisé par tous les pare-feu en mode pont sur la route entre les deux hôtes ?*
3. *Le trafic associé aux hôtes et service donnés (ou une modification de ce trafic par traduction d'adresse) est-il autorisé par tous les pare-feu en mode hyperviseur ?*

Prédicat	Description
vm(X)	X est une VM.
vapp(X,V)	X appartient au vApp V.
vdc(X,V)	X appartient au vDC V.
network(X,N)	X appartient au réseau N.
ip_address(X)	X est une adresse IP.
network_address(X)	X est une adresse réseau.
internal(X,N)	N est un réseau client interne à X.
route(W,Y,H)	Il existe une route sur W pour aller vers Y en passant par H.
mac(L,X)	X possède une liste L d'adresses MAC.
ip(L,X)	X possède une liste L d'adresses IP.
mac_ip(M,I)	L'adresse MAC M est associée à l'adresse IP I.
ip_mac(I,M)	L'adresse IP I est associée à l'adresse MAC M.
portgroup(X,N)	X appartient au groupe de port N.
vxlan(X,N)	X appartient au VXLAN N.
cidr(X,N)	L'adresse IP X appartient au réseau N.
ipset(L,I)	La liste L d'adresses IP correspond à l'IPSet I.
macset(L,M)	La liste L d'adresses MAC correspond au MACSet M.
security_group(L,S).	La liste L d'objets correspond au groupe de sécurité S.
known_service(PROTO,PORT).	Le protocole PROTO et le port PORT constituent un service connu.
service(PROTO,PORT,S)	Le protocole PROTO et le port PORT constituent le service S.
service_group(L,SG).	La liste L d'objets correspond au groupe de services SG.
allow(W,N,X,SPORT, Y,DPROTO,DPORT) allow(W,N,X,SPORT,Y,S) allow(W,N,X,SPORT,Y,SG)	La règle d'ordre N (plus N est petit, plus la règle est prioritaire) sur W autorise le trafic de X sur le port source SPORT, vers Y sur le protocole destination DPROTO et le port destination DPORT ou le service S ou le groupe de service SG.
block(W,N,X,SPORT, Y,DPROTO,DPORT) block(W,N,X,SPORT,Y,S) block(W,N,X,SPORT,Y,SG)	La règle d'ordre N (plus N est petit, plus la règle est prioritaire) sur W bloque le trafic de X sur le port source SPORT, vers Y sur le protocole destination DPROTO et le port destination DPORT ou le service S ou le groupe de service SG.
snat(W,X,PROTO,SPORT, T,TPORT)	La règle SNAT sur W permet la traduction de l'adresse source X, sur le protocole PROTO, du port source SPORT, vers T et TPORT. Les protocoles et ports ont généralement pour valeur "any".
dnat(W,X,DPROTO,DPORT, T,TPROTO,TPORT)	La règle DNAT sur W permet la traduction de l'adresse destination X, du protocole destination DPROTO, du port destination DPORT, vers T,TPROTO et TPORT.
introspect_network(N)	Il existe un pare-feu en mode hyperviseur avec un ensemble de règles pour le réseau N.
exclude_introspect(L)	La liste L contient les objets exclus du filtrage du pare-feu hyperviseur.
introspect(A)	Il existe un pare-feu A en mode hyperviseur avec un ensemble de règles global (pour les objets appartenant à des réseaux non filtrés par un ensemble de règle du pare-feu en mode hyperviseur).

Tableau 4.2 – Descriptions des prédicats des configurations

Bien entendu, chacune de ces questions comprend d'autres questions intermédiaires. Nous avons modélisé cet ensemble de questions par un algorithme, présenté sous forme d'organigramme dans la Figure 4.2, que nous avons mis en œuvre en Prolog. Cette implémentation est un ensemble de règles dépendantes les unes des autres, où une règle correspond à une question de l'algorithme. La question initiale, que nous appelons *requête d'accessibilité*, est ainsi projetée sur les trois questions principales émises précédemment et auxquelles les règles logiques tentent de répondre afin de fournir une réponse à la question initiale.

C'est un moteur logique Prolog, SWI-Prolog [25], qui a pour rôle d'exécuter le programme sur soumission d'une requête d'accessibilité. Les réponses positives (qui ont pour valeur "Vrai") correspondent aux accessibilités existantes, à savoir les prédicats

Prédicat	Exemples
vm(X)	vm(vm-001)
vapp(X,V)	vapp(vm-200,vapp-mysql)
vdc(X,V)	vdc(vm-001,vdc-04)
network(X,N)	network('10.10.10.254',network1) network(vm45,'10.10.10.0/24')
ip_address(X)	ip_address('10.10.10.12')
network_address(X)	network_address('10.10.10.0/24')
internal(X,N)	internal(fw-001,network3) internal('192.168.1.254','192.168.1.0/24')
route(W,Y,H)	route('192.168.1.4',network2,'172.16.3.254') route(edge-fw-005,'10.10.10.0/24',fw-04) route(vm-01,default,'172.16.3.254') route(edge-270,'172.16.2.0/26','192.168.1.110')
mac(L,X)	mac(['00 :05 :01 :AC :45 :E6','F5 :C1 :02 :1A :33 :4C'],vm-001)
ip(L,X)	ip(['192.168.1.2','10.10.10.10'],vm-001)
mac_ip(M,I)	mac_ip('00 :05 :01 :AC :45 :E6','192.168.1.2')
ip_mac(I,M)	ip_mac('10.10.10.10','F5 :C1 :02 :1A :33 :4C')
portgroup(X,N)	portgroup(vm-001,network1)
vxlan(X,N)	vxlan('172.16.2.4',vxlan001)
cidr(X,N)	cidr('10.10.10.15','10.10.10.0/24')
ipset(L,I)	ipset(['192.168.1.2','10.10.10.10'],ipset001)
macset(L,M)	macset(['00 :05 :01 :AC :45 :E6','F5 :C1 :02 :1A :33 :4C'],macset001)
security_group(L,S).	security_group([ipset-384,'192.168.1.2',vm-002],security-group-25).
known_service(PROTO,PORT).	known_service(udp,161) known_service(icmp,echo-reply)
service(PROTO,PORT,S)	service(tcp,389,application-7) service(icmp,echo-request,application-3141)
service_group(L,SG).	service_group([application-7,application-3141],application-group-1)
allow(W,N,X,SPORT, Y,DPROTO,DPORT) allow(W,N,X,SPORT,Y,S) allow(W,N,X,SPORT,Y,SG)	allow(edge-269,3,ipset-3581,any,internal,[application-18,application-65]) allow(edge-269,8,ipset-3581,any,['192.168.1.0/24','10.10.10.2'],any,any) allow('192.168.1.254',1,external,any,security-group-25,tcp,80) allow('192.168.1.254',3,ipset-3581,any,internal,application-3135)
block(W,N,X,SPORT, Y,DPROTO,DPORT) block(W,N,X,SPORT,Y,S) block(W,N,X,SPORT,Y,SG)	block(edge-269,3,ipset-3581,any,internal,[application-18,application-65]) block(edge-269,8,ipset-3581,any,['192.168.1.0/24','10.10.10.2'],any,any) block('192.168.1.254',1,external,any,security-group-25,tcp,80) block('192.168.1.254',3,ipset-581,any,internal,application-3135)
snat(W,X,PROTO,SPORT, T,TPORT)	snat(edge-32,'192.168.1.0/24',any,any,'65.48.2.1',any) snat('192.168.1.254','192.168.1.5',any,any,'65.48.2.1',any)
dnat(W,X,DPROTO,DPORT, T,TPROTO,TPORT)	dnat(edge-32,'65.48.2.1',tcp,2222,'192.168.1.5',tcp,22) dnat('192.168.1.254',edge-32,tcp,any,'192.168.1.5',tcp,80)
introspect_network(N)	introspect_network(network1) introspect_network(vxlan001)
exclude_introspect(L)	exclude_introspect([vm-595,vm-28,vm-591,vm-480,vm-13,vm-568])
introspect(A)	introspect(app)

Tableau 4.3 – Exemples des prédicats des configurations

d'accessibililité. Par exemple, considérons les requêtes d'accessibilité Prolog suivantes :

- $accessibility(vm\text{-}372, any, \text{'172.16.2.66'}, tcp, DPORT)$.
- $accessibility(\text{'0.0.0.0'}, any, \text{'192.168.1.150'}, icmp, DPORT)$.

Il est demandé au moteur logique de trouver les prédicats d'accessibilité associés à tous les ports TCP de vm-372 vers 172.16.2.66, puis ceux associés à tous les services ICMP de 0.0.0.0 (réseaux externes) vers 192.168.1.150. A partir de la source, l'algorithme vérifie les règles de routage, de traduction d'adresse, et de filtrage sur chaque pare-feu en mode pont présent sur la route de la source vers la destination. Dans notre modèle, un pare-feu en mode pont peut bien sûr agir en tant que simple routeur qui autorise

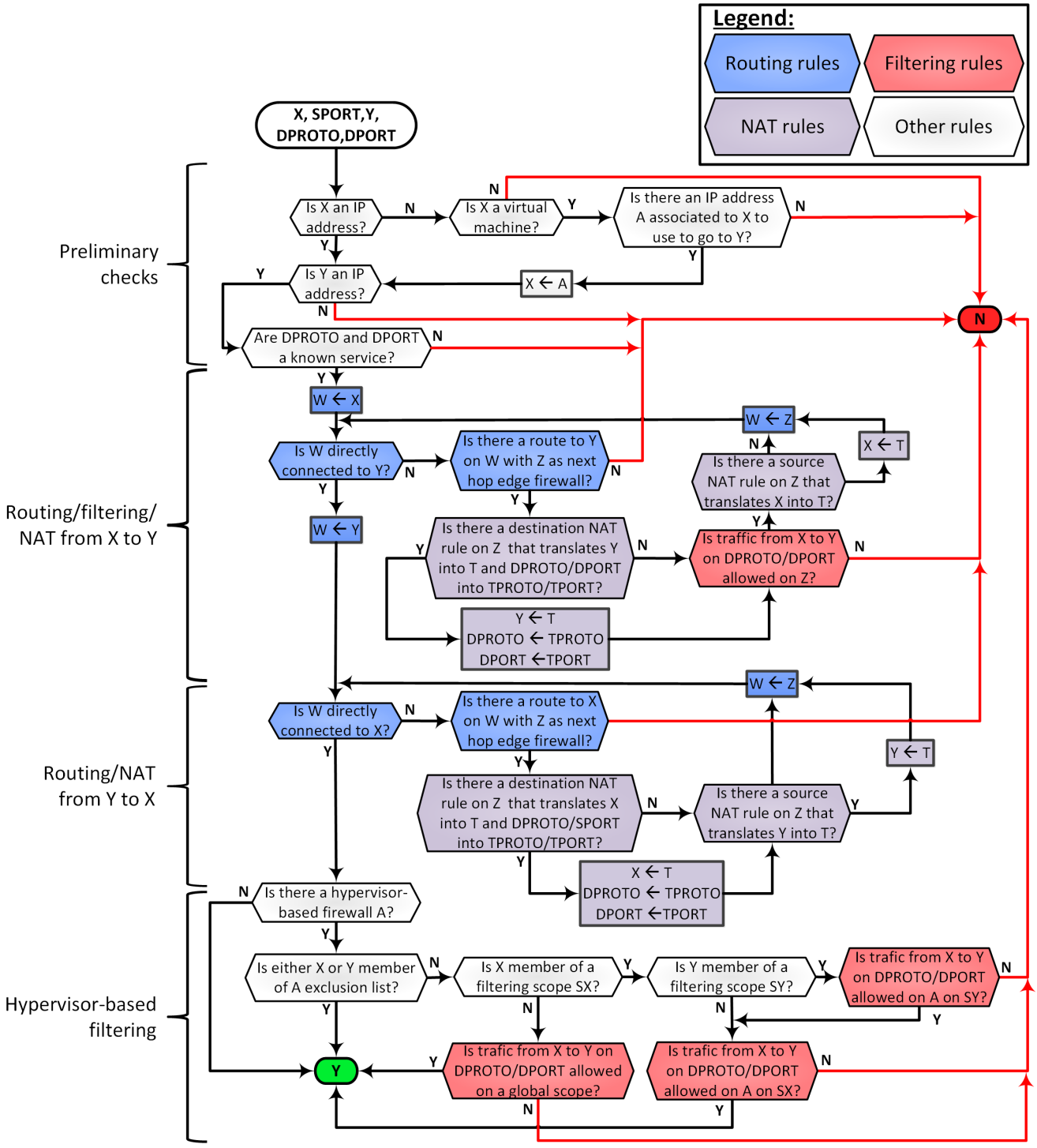


FIGURE 4.2 – Organigramme de l'algorithme de résolution des accessibilités

tout le trafic le traversant. Ensuite, l'algorithme vérifie le routage de la destination vers la source, mais ne vérifie pas le filtrage sur la route retour car nous considérons les pare-feu comme étant des pare-feu à états. Enfin, le filtrage au niveau du pare-feu en mode hyperviseur est examiné (sur les réseaux de la source et de la destination, ou par défaut sur l'ensemble des machines virtuelles).

La vérification de toutes ces conditions entraîne un grand nombre de règles dans l'algorithme, et donc un très grand nombre de tests de ces règles durant la résolution des accessibilités. Par exemple, pour savoir si un trafic est autorisé sur un pare-feu, il faut vérifier qu'il existe une règle l'autorisant, et qu'il n'y a pas de règle l'interdisant avec une priorité plus forte que celle l'autorisant. Les types de valeurs possibles des variables dans ces règles de filtrage sont :

- Pour la source et la destination : sa valeur (adresse IP), une association (adresse MAC) ou un groupe (un numéro de réseau, une plage d'adresses, un VXLAN, un groupe de ports, un vDC, un groupe de sécurité, un ensemble d'adresses), une liste, et les mots-clés "internal", "external", ou "any".
- Pour le service destination : un service, une liste de services, un groupe de services, une liste de groupes de services, les combinaisons protocole et port, protocole et mot-clé "any" pour le port, mot-clé "any" pour le protocole et mot-clé "any" pour le port.

On voit bien que le nombre d'interrogations de règles au sein de l'algorithme devient rapidement très grand. La durée d'exécution d'un tel algorithme logique dépend du nombre de requêtes d'accessibilité à traiter (qui dépend lui-même des services qui doivent être testés [1] et de la taille de l'infrastructure virtuelle) et du nombre de règles nécessaires à intérroger pour chaque requête d'accessibilité. Pour réduire cette durée d'exécution, un maximum de règles Prolog est précompilé (une fois pour toutes) de manière à connaître déjà certaines réponses et donc ne pas reposer plusieurs fois les mêmes questions au moteur logique (par exemple, précompiler tous les trafics autorisés sur chaque pare-feu, *i.e.*, les trafics entre tous les hôtes sur chaque protocole et port). En outre, les requêtes sont parallélisées par l'utilisation du multithreading.

4.3 Analyse dynamique

Nous venons de voir comment nous menons notre analyse pour déterminer statiquement la matrice contenant les accessibilités *configurées*. Pour construire la matrice contenant les accessibilités *observées*, nous procédons de manière dynamique. Des paquets réseaux sont envoyés afin de déterminer les services autorisés à destination des cibles. Généralement, un scanner de port, tentant d'initier ces communications, est utilisé depuis une machine distante contrôlée par l'auditeur. L'inconvénient majeur de ce genre de méthode est le manque de complétude des résultats, lié au fait que la machine distante doit être placée dans tous les segments réseaux pour déterminer toutes

1. Par défaut, on teste les services réseau connus fournis par le fichier `/etc/services` dans les distributions GNU/Linux.

les accessibilités existantes. De plus, cette méthode repose sur la capacité à interpréter certaines réponses pour déterminer les accessibilités, ce qui est difficile à réaliser pour du trafic orienté sans connexion comme UDP (l'emetteur ne sait pas si le récepteur a bien reçu un message). Nous traitons les problèmes cités, parce que nous sommes capables de contrôler toutes les machines faisant partie de l'infrastructure virtuelle clone, et donc de surveiller le trafic effectivement reçu sur les hôtes destinations. Nous proposons un algorithme d'analyse dynamique des contrôles d'accès réseau, puis nous étudions sa complexité.

4.3.1 Principes

L'algorithme proposé (Algorithme 2) repose sur l'élaboration de sessions pair-à-pair, en testant toutes les paires possibles parmi les machines virtuelles, et entre machines virtuelles et réseaux externes (représentés par une machine virtuelle externe à l'infrastructure du client et que nous contrôlons aussi) et vice-versa. Une session comprend un serveur et un client. Le client envoie des paquets TCP, UDP et ICMP à destination de l'adresse IP du serveur (si ce dernier possède plusieurs adresses IP, alors d'autres sessions seront utilisées) sur les services à tester [2] avec une payload spécifique de 4 octets que nous introduisons dans les paquets [3]. Le serveur écoute le trafic qu'il reçoit et applique deux filtres : un pour détecter la payload ; et un pour capturer seulement les paquets qui lui sont adressés. En effet, une machine serveur peut être aussi client dans une autre session. En revanche, elle ne peut pas être serveur de deux clients en même temps, ce qui permet un nombre maximum de serveurs par session égal au nombre total de machines virtuelles.

Il est à noter que nous appliquons une pause ($t1$) entre le lancement du serveur et celui du client, pour nous assurer que le serveur est prêt à recevoir les paquets. Puisque nous importons les machines virtuelles dans l'infrastructure, les programmes du client et du serveur sont déjà déployés sur les hôtes et prêts à être utilisés pour l'exécution de l'algorithme. Ils sont bien sûr également déployés sur la machine externe. Un tableau (M) à trois dimensions (1ère dimension : les clients ; 2ème dimension : les serveurs ; 3ème dimension : les adresses IP des serveurs) est utilisé pour reporter les sessions réalisées ou non tout au long de l'exécution de l'algorithme. Toutes les valeurs de ce tableau sont donc initialisées à "Faux", sauf quand le client et le serveur d'une session sont les mêmes (car on ne teste pas les sessions locales). A la fin de chaque itération de l'algorithme, les clients sont surveillés pour savoir s'ils ont fini d'envoyer leurs paquets afin de pouvoir arrêter les serveurs et récupérer les résultats (les accessibilités découvertes pour chaque session). L'algorithme commence par réaliser les sessions entre machines virtuelles de l'infrastructure (*l. 1-32*), puis celles des réseaux externes vers les machines virtuelles (*l. 33-46*, parallélisées, car la machine virtuelle externe peut être client de toutes les machines virtuelles serveurs à la fois), et enfin des machines virtuelles vers

2. Par défaut, on teste les mêmes services que lors de l'analyse statique.
3. Cette payload, comprenant la suite de caractères "PASS", sert simplement à valider le transfert de données.

Algorithme 2 Algorithme d'analyse dynamique

Require: V : ensemble des VMs et leurs adresses IP
Require: M : tableau de sessions
Require: *this* : machine d'audit externe
Require: *this_ip* : adresse IP de la machine d'audit
Ensure: AM : matrice d'accessibilité

```
still_sessions ← True
while still_sessions do
    still_sessions ← False
    for n ∈ 1...Card(V) do
        session_found ← False
        S ← ∅
        vm1 ← 1
        while vm1 ≤ Card(V) and !session_found do
            vm2 ← 1
            while vm2 ≤ Card(V) and !session_found do
                ip ← 1
                while M[vm1][vm2][ip] do ip++ end while
                if !M[vm1][vm2][ip] and {V[vm1],V[vm2][ip]} ∉ S then
                    still_sessions ← True
                    run_server(V[vm2])
                    sleep(t1)
                    run_client(V[vm1])
                    S ← S ∪ {V[vm1],V[vm2],V[vm2][ip]}
                    M[vm1][vm2][ip] ← True
                    session_found ← True
                end if
                vm2++
            end while
            vm1++
        end while
    end for
    for session ∈ S do
        monitor_session(session)
        sleep(t2)
        AM[session[0]][session[2]] ← get_accessibility_results(session[1])
    end for
end while
S ← ∅
for vm ∈ 1...Card(V) do
    for ip ∈ 1...Card(vm[ip]) do
        run_server(V[vm])
        sleep(t1)
        run_client(this)
        S ← S ∪ {this,V[vm],V[vm][ip]}
    end for
end for
for session ∈ S do
    monitor_session(session)
    sleep(t2)
    AM[session[0]][session[2]] ← get_accessibility_results(session[1])
end for
for vm ∈ 1...Card(V) do
    run_server(this)
    sleep(t1)
    run_client(V[vm])
    session ← {V[vm],this,this_ip}
    monitor_session(session)
    sleep(t2)
    AM[session[0]][session[2]] ← get_accessibility_results(session[1])
end for
return AM
```

les réseaux externes (*l. 47-55*, séquentiellement, car la machine virtuelle externe ne peut être serveur que d'un seul client à la fois).

4.3.2 Complexité

Il est intéressant d'analyser la complexité de l'algorithme proposé. Notons :

- N_V le nombre de machines virtuelles de l'infrastructure.
- N_{IP}^i le nombre total d'adresses IP d'une machine virtuelle i.
- N_S^i le nombre de sessions avec des machines virtuelles de l'infrastructure pour une machine virtuelle i de l'infrastructure en tant que serveur.

$$N_S^i = \sum_{\substack{0<j<N_V \\ i \neq j}} N_{IP}^i = N_{IP}^i \times (N_V - 1)$$

- $N_V + \sum_{0<i<N_V} N_{IP}^i$ le nombre de sessions entre machines virtuelles de l'infrastructure et la machine externe.
- N_S le nombre total de sessions (sessions entre machines virtuelles de l'infrastructure et sessions avec les réseaux externes).

$$N_S = \sum_{0<i<N_V} N_S^i + N_V + \sum_{0<i<N_V} N_{IP}^i = \sum_{0<i<N_V} (N_S^i + 1 + N_{IP}^i)$$

$$= \sum_{0<i<N_V} (N_{IP}^i \times N_V + 1)$$

- d le délai entre chaque envoi de paquet.
- N_P le nombre de paquets à envoyer au cours de chaque session.

Une itération de l'algorithme consiste à exécuter plusieurs sessions en parallèle, et le temps requis pour exécuter une session vaut $N_P \times d + t1$ ($t1$ est la durée de la pause entre le lancement du serveur et du client). Quand une machine virtuelle i n'a qu'une seule adresse IP, $N_S^i = N_V - 1$, et c'est également le nombre d'itérations pour les échanges entre machines virtuelles de l'infrastructure (chaque machine virtuelle est serveur d'une session à chaque itération). Cependant, les machines virtuelles peuvent avoir plusieurs adresses IP (aussi appelé *multihoming*), et une machine virtuelle ne peut pas être serveur de plus d'un client au cours d'une même itération. Ainsi, pour un serveur i, le nombre de sessions additionnelles entre machines virtuelles dues au *multihoming* est noté $N_A^i = N_S^i - (N_V - 1)$. Le *multihoming* entraîne $\max\{N_A^i\}$ itérations supplémentaires (où les sessions additionnelles sont exécutées en parallèle toujours en suivant l'algorithme). Ainsi, on peut déduire le nombre d'itérations pour les sessions entre machines virtuelles de l'infrastructure comme étant $N_V - 1 + \max\{N_A^i\}$. Les sessions de la machine externe vers les adresses IP des machines virtuelles de l'infrastructure sont lancées en parallèle ($\max\{N_{IP}^i\}$ itérations), et les sessions des machines virtuelles de l'infrastructure vers la machine externe sont exécutées séquentiellement (N_V itérations). En prenant également en compte la pause $t2$ assurant que les résultats

de chaque session aient bien été générés sur un serveur avant de les récupérer, on peut donner une estimation du temps d'exécution :

$$T = (2N_V - 1 + \max\{N_A^i\} + \max\{N_{IP}^i\}) \times (N_P \times d + t1) + N_S \times t2$$

Dans le cas particulier où toutes les machines n'ont qu'une adresse IP ($\max\{N_A^i\} = 0$, $\max\{N_{IP}^i\} = 1$), le temps d'exécution devient :

$$T = 2N_V \times (N_P \times d + t1) + N_S \times t2$$

Dans cette formule, nous ne prenons pas en compte le temps ajouté par les durées d'exécution des instructions, de l'utilisation des APIs du cloud et du temps requis pour récupérer les résultats d'accessibilité et les écrire sur le disque. Notre algorithme a une durée d'exécution linéraire qui dépend du nombre de machines virtuelles et d'adresses IP par VM. Un algorithme qui exécuterait simplement chaque session de manière séquentielle aurait une durée d'exécution exponentielle dépendant du nombre de sessions (bien sûr, la durée des pauses seraient également à considérer).

4.4 Analyse des déviances

A partir des résultats d'analyse des contrôles d'accès, il est intéressant de comparer les matrices obtenues entre elles afin de mettre en évidence des déviances dans la définition, la configuration ou l'observation des accessibilités réseau. Si on considère les matrices suivantes :

- AM_d : matrice des accessibilités *définies*, générée à partir de la politique de sécurité du client.
- AM_c : matrice des accessibilités *configurées*, générée à partir de l'analyse statique.
- AM_o : matrice des accessibilités *observées*, générée à partir de l'analyse dynamique.

Les déviances dans l'analyse des contrôles d'accès réseau incluent toute accessibilité a telle que :

- $a \in AM_d$ et $a \notin AM_c$, a est *définie* mais n'est pas *configurée*.
- $a \notin AM_d$ et $a \in AM_c$, a n'est pas *définie* mais est *configurée*.
- $a \in AM_d$ et $a \notin AM_o$, a est *définie* mais n'est pas *observée*.
- $a \notin AM_d$ et $a \in AM_o$, a n'est pas *définie* mais est *observée*.
- $a \in AM_c$ et $a \notin AM_o$, a est *configurée* mais n'est pas *observée*.
- $a \notin AM_c$ et $a \in AM_o$, a n'est pas *configurée* mais est *observée*.

Idéalement, les trois matrices devraient être identiques et aucune déviance ne devrait être observée. Puisque nous pouvons comparer deux par deux les trois matrices entre elles, une déviance peut être idéntifiée à six niveaux différents. Ceci permet à un administrateur de savoir s'il s'agit d'une erreur humaine dans la définition ou configuration des contrôles d'accès, ou d'une erreur logicielle dans l'application des contrôles d'accès configurés.

Dans le prochain chapitre, nous considérons les accessibilités *observées* (matrice AM_o) comme références, car elles sont les plus proches de la réalité de par le caractère dynamique de la méthode utilisée pour les déterminer. Une autre raison de ce choix est le caractère commun de la méthode d'analyse dynamique des contrôles d'accès et de la méthode d'exécution de campagnes d'attaque. En effet, ces deux méthodes ont un caractère expérimental qui comprend l'envoi de trafic réseau.

4.5 Conclusion

Dans ce chapitre, nous avons présenté les méthodes d'analyse statique et dynamique des contrôles d'accès réseau dans les infrastructures virtuelles de cloud computing. La méthode d'analyse statique présentée consiste à récupérer les informations de configuration du cloud mises à disposition, puis de s'en servir en tant que prédicats dans la résolution logique des accessibilités, également exprimées sous forme de prédicats puis de matrice d'accessibilité. L'analyse dynamique a le même objectif, à savoir la génération d'une matrice d'accessibilité, mais une autre méthode est utilisée. Elle consiste à réaliser des sessions d'envois de paquets réseau entre les machines de l'infrastructure analysée, en parallélisant au maximum ces sessions. Au final, les résultats peuvent être comparés entre eux, ainsi qu'avec les accessibilités définies dans la politique de sécurité, si disponibles. L'objectif de cette comparaison est de mettre en évidence d'éventuelles déviances dans l'implémentation de la politique de sécurité, où dans l'application des règles par les équipements de filtrage réseau.

Dans le chapitre suivant, nous présentons la troisième et dernière phase du processus d'évaluation et d'analyse des mécanismes de sécurité réseau. Cette phase vient logiquement après celle présentée dans ce chapitre. En effet, les accessibilités trouvées y sont utilisées afin de mener des campagnes d'attaque réseau pour analyser la réaction des systèmes de détection d'intrusion réseau.

Chapitre 5

Évaluation des systèmes de détection d'intrusion réseau

Sommaire

5.1 Trafic d'évaluation 73
5.1.1 Modélisation sous forme d'automates 74
5.1.2 Génération des automates 75
5.1.3 Rejeu des automates 76
5.2 Campagnes d'attaque 80
5.2.1 Dictionnaire d'attaques 80
5.2.2 Sessions d'attaque 81
5.2.3 Exécution des campagnes d'attaque 82
5.2.4 Calcul des métriques d'évaluation 84
5.2.5 Analyse des incohérences de détection 85
5.3 Conclusion 85

Ce chapitre présente la troisième phase de notre approche d'évaluation et d'analyse des mécanismes de sécurité réseau dans le cloud, et porte sur l'évaluation des systèmes de détection d'intrusion réseau. Elle intervient après l'analyse des contrôles d'accès réseau, dont les méthodes ont été présentées dans le chapitre précédent. L'objectif de cette phase est de fournir des métriques d'évaluation des NIDS en analysant leur réaction face à des campagnes d'attaque. La réalisation de campagnes d'attaque requiert l'envoi de trafic d'évaluation spécifique. Nous présentons donc d'abord comment nous avons modélisé, généré et rejoué ce trafic d'évaluation, qui est utilisé par notre méthode d'exécution de campagnes d'attaque, décrite en détail par la suite. Nous expliquons comment nous structurons les informations relatives aux attaques sous forme de dictionnaire d'attaques, puis exposons notre algorithme d'exécution de campagnes d'attaque. Enfin, nous décrivons comment nous calculons les métriques d'évaluation et sommes capables de mettre en évidence des incohérences dans la détection des attaques pour chaque NIDS évalué.

5.1 Trafic d'évaluation

Il est nécessaire de bien caractériser le trafic d'évaluation envoyé durant les campagnes d'attaque. Puisque nous voulons évaluer les systèmes de détection d'intrusion

réseau, nous nous intéressons aux attaques réseau les plus courantes, à savoir celles ciblant les applications utilisant les protocoles de transport TCP et UDP. Les approches à base de modèles sont bien adaptées pour représenter le trafic et ainsi en utiliser une abstraction pour automatiser le processus de rejeu. En effet, cela nous permet, en amont des audits, de générer des modèles d'automates représentant le trafic d'évaluation, puis de rejouer ces automates dans des contextes différents durant les procédures d'audit.

5.1.1 Modélisation sous forme d'automates

Un trafic est défini par une séquence de paquets réseaux malveillants ou légitimes échangés, et généralement regroupés pour former des requêtes et des réponses. Pour mener des campagnes d'attaque comme expliqué plus tard dans ce chapitre, nous prévoyons d'envoyer du trafic malveillant et légitime à destination d'applications vulnérables. Par conséquent, les machines virtuelles de l'infrastructure clone doivent être en mesure d'envoyer et recevoir un tel trafic. Plusieurs contraintes sont alors à prendre en compte. En premier lieu, beaucoup d'applications vulnérables disponibles sont destinées au système d'exploitation propriétaire Windows, ce qui nécessiterait l'obtention de licences Windows pour chacune des machines virtuelles importées durant le processus de clonage de l'infrastructure virtuelle. Or, ceci n'est pas envisageable quand le nombre de machines virtuelles est trop important. De plus, le processus d'installation, lancement et arrêt d'applications est fastidieux et difficile à automatiser (notamment sous Windows). Nous avons donc besoin d'un système ouvert, comme Linux, pour pouvoir automatiser facilement le rejeu de certains comportements des applications dans les machines virtuelles lors de nos opérations d'audit, afin de se passer du déploiement des applications. Par ailleurs, on ne peut pas prédire quelles applications pourraient être déployées par les clients. Ainsi, on ne peut pas prédire quelles attaques pourraient être réellement exécutées (hors du cadre de l'évaluation) et devraient être détectées par les NIDS. De ce fait, nous avons besoin d'une solution permettant la simulation de comportements réseau d'un large ensemble d'applications vulnérables.

Pour ces raisons, nous avons décidé de modéliser des comportements réseau légitimes et malveillants d'applications, et de développer un outil permettant d'utiliser les modèles pour simuler ces comportements. Nous ne sommes pas intéressés par la modélisation de tous les comportements réseau possibles de l'application, mais seulement celle de certains flux : l'utilisation malveillante de l'application ainsi que certains usages légitimes. La littérature montre que presque tous les protocoles peuvent être modélisés avec un langage de type régulier [1]. Par conséquent, notre choix s'est naturellement porté sur les automates. L'essentiel est que le rejeu de trafic soit assez réaliste pour accréditer nos campagnes d'évaluation. Nous supposons avoir atteint cette situation lorsqu'un outil d'attaquant réagit de la même manière face aux réponses issues de l'automate que face aux réponses issues de l'application.

Ainsi, nous modélisons les échanges de paquets entre l'attaquant et l'application

1. En faisant abstraction des valeurs et de la sémantique.

cible par un automate fini, où chaque état représente un paquet par un quintuplet incluant :

- La direction du paquet : reçu ou envoyé par l'application.
- Le protocole : TCP ou UDP.
- Le numéro de port.
- L'identifiant de l'application.
- La payload.

L'état initial d'un automate contient des valeurs de quintuplet vides ou nulles. La Figure 5.1 montre un exemple d'automate représentant le comportement réseau d'un serveur FTP avec une utilisation malveillante et une utilisation légitime.

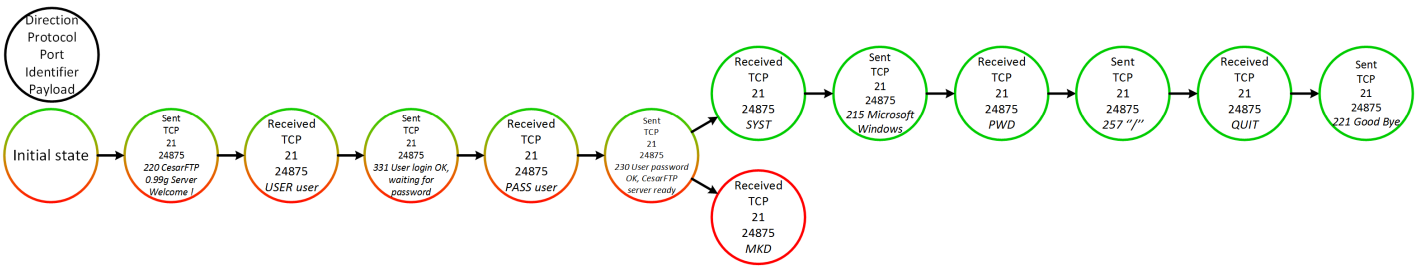


FIGURE 5.1 – Exemple d'automate modélisant les échanges réseau d'un usage légitime (en vert) et d'un usage malveillant (en rouge) d'un serveur FTP

L'automate est en fait le regroupement de deux automates (l'automate de l'utilisation malveillante et celui de l'utilisation légitime). Cette réunion est possible car ils sont associés à la même application, et partagent des états communs avant de diverger. Les états communs correspondent ici à l'authentification sur le serveur FTP, puis les états légitimes suivants sont des commandes FTP de base, tandis que l'état malveillant correspond à une commande de débordement de tampon (associée à la CVE 2006-2961).

5.1.2 Génération des automates

Bien que cela requiert un effort d'intervention humaine, le processus de génération d'un automate est bien moins contraignant que le déploiement d'applications à la volée durant le processus d'évaluation. En effet, la génération d'un automate d'une application, qui requiert bien sûr l'installation puis l'utilisation de l'application, est à faire une seule fois en amont des procédures d'audit. Pour générer nos automates, nous avons utilisé une maquette avec deux machines virtuelles connectées directement dans le même réseau IP. La machine cible héberge les applications vulnérables et capture le trafic réseau émis. La machine attaquante est responsable de la génération des requêtes (légitimes et malveillantes) vers les applications vulnérables. La machine cible exécute un système Windows ou Linux, en fonction des applications à héberger, ainsi que Wireshark [34] pour capturer le trafic réseau. La machine attaquante exécute un

système Linux, ainsi que Metasploit [13] pour envoyer des requêtes malveillantes. Nous avons choisi Metasploit car c'est un outil reconnu et très utilisé par la communauté, qui propose un très large ensemble d'exploits couvrant diverses vulnérabilités applicatives. De plus, c'est un logiciel en source libre facilement automatisable. La machine attaquante possède aussi des outils en ligne de commande pour générer des requêtes réseau légitimes :

- `wget` et `curl` en tant que clients HTTP.
- `ftp` en tant que client FTP.
- `telnet` en tant que client SMTP et IMAP.

Pour les applications utilisant des protocoles réseau non intéractifs comme HTTP, nous utilisons des scripts Shell pour générer des requêtes avec `wget` ou `curl`. Pour les protocoles réseau intéractifs comme FTP, SMTP et IMAP, nous utilisons *autoexpect* [2] pour écouter les sessions réalisées avec `ftp` ou `telnet` et générer automatiquement des scripts Expect qui pourront, comme les scripts Shell, être réutilisés dans d'autres contextes.

Le processus de génération d'un automate avec notre maquette, illustré par la Figure 5.2, est le suivant :

1. Installation et lancement de l'application vulnérable sur la cible.
2. Lancement de Wireshark avec filtrage sur les addresses IP, protocoles et ports considérés.
3. Exécution de l'exploit avec Metasploit depuis l'attaquant.
4. Sauvegarde de la capture réseau de Wireshark et utilisation pour la génération d'un automate stocké sous forme de fichier pickle [2] à l'aide de notre outil Python de génération d'automates (appelé *automata generator*).
5. Redémarrage de l'application si besoin, et redémarrage de Wireshark avec filtrage sur les addresses IP, protocoles et ports donnés.
6. Exécution des programmes en ligne de commande légitimes (avec création et exécution d'un script Shell pour les protocoles non intéractifs, ou enregistrement des sessions avec *autoexpect* pour les protocoles intéractifs).
7. Sauvegarde de la capture réseau de Wireshark et utilisation pour la génération d'un automate stocké sous forme de fichier pickle à l'aide de l'*automata generator*.

L'outil *automata generator*, que nous avons développé pour la génération d'automates, s'appuie sur la librairie Python de l'outil Scapy [24] pour lire les paquets dans les fichiers PCAP générés par Wireshark. Les automates sont mis en œuvre en Python sous forme de listes sérialisées en objets pickle et stockées dans des fichiers afin de pouvoir être réutilisées par la suite.

5.1.3 Rejeu des automates

Notre but étant de se passer de l'installation et de la configuration des applications vulnérables durant le processus d'audit, on cherche à rejouer les attaques et usages lé-

2. Mécanisme de sérialisation d'objets en Python.

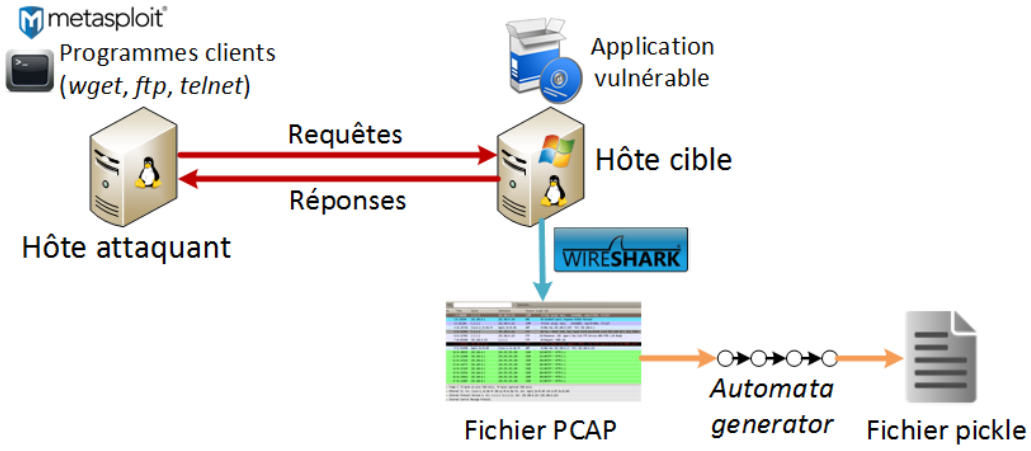


FIGURE 5.2 – Génération d'un automate

gitimes déjà exécutés dans le processus de génération des automates. Notre objectif est que les outils émettant des requêtes vers les applications aient le même comportement lors de l'utilisation d'automates à la place des applications. Pour ce faire, nous avons conçu et mis en œuvre un autre outil en Python (appelé *automata player*) dédié au rejeu des automates. Cet outil léger est utilisé dans nos machines virtuelles importées dans l'infrastructure clone. Son travail consiste à écouter le trafic entrant afin d'envoyer les réponses appropriées aux requêtes reçues, en fonction des automates dont il dispose.

A son lancement, l'*automata player* lit les fichiers pickle contenant les automates associés à l'application dont on veut simuler des comportements réseau. Il les regroupe, en fusionnant les états équivalents (au minimum, l'état initial devient commun). Ensuite, il lance deux threads :

- Le premier crée un socket TCP pour écouter les connexions entrantes, et lance un nouveau thread à chaque fois qu'une connexion TCP est initiée. Ces threads écoutent les paquets TCP entrant au sein de leur connexion.
- Le second crée un socket UDP et écoute les paquets UDP entrants.

Le principe de sélection des réponses à envoyer, appliqué lors de la réception d'une requête par appel de la fonction *select_responses*() (*l. 36-51*), est illustré par l'Algorithme 3. Une requête ou une réponse est composée d'un ou plusieurs paquets réseau contenant des payloads (données utiles relatives à l'application) et transmis dans la même direction. L'état courant de l'automate est initialisé à son état initial, puis mis à jour durant l'exécution de l'algorithme. Ce dernier consiste, pour une payload reçue, à trouver la payload la plus ressemblante dans l'automate depuis à la fois l'état courant et l'état initial (fonction *find_payload*(), *l. 4-23*), et à trouver la payload suivante à envoyer en tant que réponse (fonction *find_payload_to_send*(), *l. 24-35*).

Algorithme 3 Sélection des réponses

Require: *initial_state* : pointeur sur l'état initial de l'automate.
Require: *current_state* : pointeur sur l'était courant de l'automate, initialisé à l'état initial.

```
1: function PAYLOAD_MATCH(state, prev_match_score, payload1, payload2)
2:     return (levenshtein(payload1, payload2) > prev_match_score ? levenshtein(payload1, payload2) : False)
3: end function
4: function FIND_PAYLOAD(state, payload, previous_match_score, proto)
5:     score ← previous_match_score
6:     for s ∈ state["children"] do
7:         if s["received"] and s["proto"] == proto then
8:             concat_payloads ← s["payload"]
9:             last_state ← s
10:            if s["proto"] == "TCP" then
11:                (last_state, concat_payloads) ← concat_tcp_payloads(s, concat_payloads)
12:            end if
13:            match_score ← payload_match(score, concat_payloads, payload)
14:            if match_score then
15:                score ← match_score
16:                current_state ← last_state
17:            end if
18:        else
19:            score ← find_payload(s, payload, score, proto)
20:        end if
21:    end for
22:    return score
23: end function
24: function FIND_PAYLOAD_TO_SEND(state, prev_selected_payload, proto)
25:    p_to_send ← prev_selected_payload
26:    for s ∈ state["children"] do
27:        if s["proto"] == proto and !s["received"] then
28:            payload_to_send ← s["payload"]
29:            current_state ← s
30:        else if s["proto"] ≠ proto then
31:            p_to_send ← find_payload_to_send(s, p_to_send, proto)
32:        end if
33:    end for
34:    return p_to_send
35: end function
36: function SELECT_RESPONSES(received_payload, proto)
37:    score ← 0 ; responses ← ∅ ; next_payload ← ""
38:    score ← find_payload(current_state, received_payload, score, proto)
39:    score ← find_payload(initial_state, received_payload, score, proto)
40:    if !automaton_finished(current_state) then
41:        next_p ← find_payload_to_send(current_state, next_p, proto)
42:        while next_p do
43:            responses ← responses ∪ next_p
44:            next_p = ""
45:            if !automaton_finished(current_state) then
46:                next_p ← find_payload_to_send(current_state, next_p, proto)
47:            end if
48:        end while
49:    end if
50:    return responses
51: end function
```

Il est à noter que la recherche de payload reçue est effectuée pour les états fils de l'état courant afin de pouvoir suivre l'évolution des échanges de paquets, mais aussi pour les états fils de l'état initial afin de pouvoir répondre à un éventuel nouvel usage de l'application. Ainsi, on n'a pas besoin de recharger les automates des applications à chaque usage différent (légitime ou malveillant) d'une même application.

Une payload reçue est délivrée par un socket, et est issue d'un paquet UDP, ou bien d'un ou plusieurs paquets TCP. En effet, dans le cas de TCP, la payload peut être répartie sur plusieurs paquets, car contrairement à un protocole sans connexion orienté message comme UDP, TCP est un protocole avec connexion orienté flux d'octets qui ne respecte pas la frontière initiale des messages. Ainsi, nous avons dissocié deux traitements différents pour chercher la réponse sur réception d'une payload :

- Dans le cas d'UDP, la payload reçue est comparée (à l'aide d'une fonction de comparaison de chaînes de caractères) avec les payloads des fils de l'état courant et des fils de l'état initial. Puis, une fois qu'on a trouvé la payload la plus semblable, chacune des payloads des états suivant celui contenant la payload la plus semblable et dont la direction reste opposée (paquet envoyé par l'application) est sélectionnée et envoyée en tant que réponse dans un paquet UDP.
- Dans le cas de TCP, tant que la direction associée est la même (paquet reçu par l'application), les payloads contiguës des états suivant l'état courant et celles suivant l'état initial sont concaténées, puis ces concaténations sont comparées à la payload reçue. Ensuite, une fois qu'on a trouvé la concaténation de payloads la plus semblable, à partir de l'état contenant la dernière payload concaténée, les payloads des états suivant sont aussi concaténées tant que la direction est la même (paquet envoyé par l'application), puis envoyées en tant que réponse dans un flux TCP.

Enfin, une fois que la réponse TCP ou UDP est envoyée, l'état courant de l'automate est mis à jour, et il est vérifié si l'automate a atteint son état final. Il est à noter que pour effectuer la comparaison de payloads, nous avons utilisé une implémentation de la distance de Levenshtein [22].

En plus d'être portable et exécutable en ligne de commande, l'*automata player* développé répond à nos besoins. Un algorithme de calcul de distance mathématique de chaînes de caractères (comme la distance de Levenshtein) est suffisant pour comparer et identifier les payloads même si certains éléments des payloads reçues diffèrent des payloads à trouver dans l'automate (*i.e.*, la similarité n'est pas parfaite). En ce qui concerne les protocoles binaires, un algorithme de calcul de distance de chaînes de caractères appliqué aux données binaires donne de bonnes performances à la vue de nos expérimentations. De plus, la reconnaissance d'une payload est assez rapide et très peu propice à une erreur étant donné le nombre d'états limité des automates et étant donné que nous réduisons les comparaisons depuis à la fois l'état courant et l'état initial. Aussi, nos études sur les codes sources d'exploits et les traces réseau envoyées nous ont montré que les programmes d'exploit ne font pas une analyse poussée du contenu des réponses reçues. C'est pourquoi nous n'effectuons pas une analyse et modification des payloads envoyées en tant que réponse en fonction des contextes.

Concernant les systèmes de détection d'intrusion réseau, les signatures sont généralement déclenchées par les requêtes. L'essentiel est donc d'assurer l'envoi de réponses pour permettre l'émission d'autant de requêtes malveillantes que si l'application était vraiment déployée pour assurer les réponses. Dans le cas où les réponses lèveraient les alertes, nous avons constaté que les paramètres variables (timestamps, adresses IP, noms d'hôtes, numéro de versions, etc.) n'affectent pas la manière de déclencher les alertes. Ceci est cohérent, puisque les signatures ne peuvent pas dépendre de contextes si spécifiques, auquel cas elles seraient très difficilement déclenchées.

5.2 Campagnes d'attaque

Une campagne d'attaque comprend un ensemble de sessions d'attaque. Une session d'attaque est l'ensemble des attaques réseau pouvant être exécutées entre deux hôtes. Afin d'évaluer la réaction des systèmes de détection d'intrusion réseau face à des usages malveillants et légitimes d'applications vulnérables, nous associons toujours une activité légitime à chaque attaque réseau exécutée dans une session d'attaque. Pour pouvoir choisir ces attaques, nous avons besoin de regrouper les informations relatives dans un dictionnaire. Après exécution des attaques et activités légitimes, des métriques d'évaluation sont calculées pour évaluer l'efficacité des NIDS vis-à-vis du trafic d'évaluation envoyé.

5.2.1 Dictionnaire d'attaques

Nous avons besoin d'une structure de données pour stocker les informations nécessaires relatives aux attaques que nous sommes capables d'exécuter dans les campagnes. Ainsi, une attaque est caractérisée par l'exploit utilisé, sa date, la vulnérabilité associée, le protocole et le numéro de port associés, et sa description. Nous avons implémenté le dictionnaire d'attaques en tant que base de données SQLite avec une table, définie par ses attributs dans le Tableau 5.1.

Nom	Type	Description
exploit_id	Entier	Identifiant de l'exploit
cve	Chaîne de caractères	Identifiant de la vulnerabilité
proto	Chaîne de caractères	Protocole réseau associé
port	Entier	Port réseau associé
date	Date	Date de l'exploit
description	Chaîne de caractères	Description de la vulnerabilité
automata	Booléen	Indique si des automates sont disponibles

Tableau 5.1 – Attributs du dictionnaire d'attaques

Pour insérer les données dans le dictionnaire, nous avons exploré la base accessible sur le site Web Exploit Database [9], qui nous permet de récupérer les informations dont

nous avons besoin, avec parfois même l'application vulnérable. Le champ *automata* est le seul que nous mettons à jour manuellement, après génération de l'automate (cf. Section 5.1.2). Il est à noter que la valeur du champ *exploit_id* est utilisée dans le nom des fichiers pickle contenant les automates (pour les identifier facilement).

5.2.2 Sessions d'attaque

Une attaque réseau est la tentative d'exploitation d'une vulnérabilité sur un service donné par un hôte vers un autre hôte. L'attaque est envisageable s'il existe une accessibilité entre les deux hôtes, et qu'un automate d'application vulnérable ainsi que le programme d'exploit associés sont disponibles. Les sessions d'attaque sont donc guidées par les accessibilités, pour qu'aucune attaque ne soit tentée si le contrôle d'accès réseau ne le permet pas. L'ordre des étapes de notre approche d'évaluation et d'analyse est ainsi essentiel, car la découverte des accessibilités est un préalable fondamental aux campagnes d'attaque. En effet, la connaissance des accessibilités permet de ne tenter que des attaques sur les services réseaux autorisés, et de ne pas perdre de temps à tenter des attaques impossibles (dont le trafic réseau associé est bloqué par les pare-feu), ce qui contribue ainsi à limiter drastiquement la durée totale d'exécution d'une session d'attaque.

Notre objectif est d'évaluer la réaction des NIDS face à des sessions d'attaque dont les attaques doivent pouvoir s'exécuter et varier en nombre et singularité. Ainsi, nous avons dissocié et caractérisé les *attaques standards* et les *attaques non standards* :

- Une *attaque standard* est une attaque exploitant une vulnérabilité sur un service sur lequel l'application vulnérable est censée s'exécuter, comme par exemple une attaque Web ciblant un serveur Web sur le protocole TCP et le port 80.
- Une *attaque non standard* est une attaque exploitant une vulnérabilité sur un service sur lequel l'application vulnérable n'est pas censée s'exécuter, comme par exemple une attaque Web ciblant un serveur Web sur le protocole TCP et le port 21.

Nous considérons plusieurs stratégies d'exécution des sessions d'attaque, en permettant la définition des proportions d'attaques des deux catégories : *attaques standards* et *attaques non standards*. Ainsi, trois modes d'attaque sont offerts pour choisir le nombre d'attaques à lancer pour chaque catégorie :

1. Sur chaque accessibilité, lancer toutes les attaques disponibles de la catégorie pour le service de l'accessibilité.
2. Sur chaque accessibilité, lancer les n plus récentes (par rapport aux dates des vulnérabilités associées) attaques de la catégorie pour le service de l'accessibilité.
3. Sur chaque accessibilité, lancer n attaques choisies aléatoirement dans la catégorie pour le service de l'accessibilité.

A titre d'exemple, si on choisit le mode d'attaque 1 pour la catégorie d'*attaques standards* et le mode d'attaque 3 avec $n = 2$ pour la catégorie d'*attaques non standards*, il sera lancé pour chacune des accessibilités :

- Toutes les *attaques standards* disponibles pour le service de l'accessibilité.
- Deux *attaques non standards* choisies aléatoirement pour le service de l'accessibilité.

Il est à noter qu'à chaque *attaque standard* ou *attaque non standard* exécutée est toujours ajoutée une utilisation légitime de l'application associée.

Une session d'attaque entre deux machines virtuelles (contrôlées par l'API du cloud) est réalisée en exécutant, pour chaque accessibilité et en suivant les modes d'attaque choisis, les attaques et activités légitimes (standards et non standards) entre l'attaquant et la cible. Pour ce faire, les machines virtuelles sont synchronisées afin d'assurer le bon déroulement des attaques. L'outil Metasploit est démarré sur l'attaquant, puis lorsqu'il est prêt à exécuter un exploit et les scripts légitimes, l'outil *automata player* est démarré sur la cible et charge l'automate associé. Puis, un script (écrit en Ruby, car Metasploit offre une API en Ruby) est démarré sur l'attaquant. Ce script a pour rôle de :

1. Sélectionner dans le dictionnaire d'attaques les attaques à lancer en fonction du protocole, du port et des modes d'attaque.
2. S'interfacer avec Metasploit pour lancer l'exécution de chaque exploit, et exécuter les scripts légitimes.
3. Synchroniser le déroulement des sessions d'attaque.

Ce dernier point consiste à attendre un signal avant l'exécution de chaque couple attaque et activité légitime et notifier de la fin de cette exécution. L'API de l'hyperviseur du cloud nous permet là encore de lancer les programmes dans les machines virtuelles et de récupérer les résultats des attaques, indiquant quelles attaques ont abouti (dont les automates associés ont atteint l'état final).

5.2.3 Exécution des campagnes d'attaque

Afin de réaliser des campagnes d'attaque (*i.e.*, un ensemble de sessions d'attaque), nous avons conçu un algorithme d'exécution de campagnes d'attaque, détaillé par l'Algorithme 4. Des threads sont lancés (en fonction du nombre de cœurs disponibles) et exécutent l'algorithme, c'est-à-dire qu'ils parcourent la matrice d'accessibilité et exécutent les sessions d'attaque possibles (représenté par la fonction *run_attacks*(), *l. 31*). L'objectif est de réaliser autant de sessions d'attaque en parallèle que possible en suivant la matrice d'accessibilité tout en assurant la bonne exécution des sessions d'attaque. Concrètement, les contraintes de développement associées sont proches de celles liées à l'algorithme d'analyse dynamique des contrôles d'accès, présenté dans le Chapitre 4, où on cherchait également à réaliser un maximum d'envois de paquets en parallèle.

Algorithme 4 Exécution de campagnes d'attaque

Require: AM : matrice d'accessibilité.
Require: $nb_sessions$: nombre total de sessions d'attaque issu de la matrice d'accessibilité.
Require: $attack_modes$: modes d'attaque.
Require: $sessions$: variable partagée, initialisée à $\emptyset$, pour mémoriser les sessions effectuées, à réaliser, et en cours.
Require: S : sémaphore binaire partagé, initilisé à 1, et manipulé avec les opérations traditionnelles $P()$ et $V()$.

```
1: still_sessions ← True
2: while still_sessions do
3:     still_sessions ← False
4:     P(S)
5:     if sessions.length == nb_sessions then still_sessions ← True end if
6:     V(S)
7:     n ← 0
8:     for attacker ∈ AM.keys() do
9:         for target ∈ AM[attacker].keys() do
10:            for ip ∈ AM[attacker][target].keys() do
11:                session_available = False
12:                for proto ∈ AM[attacker][target][ip].keys() do
13:                    if AM[attacker][target][ip][proto] then
14:                        session_available ← True ; n ← n + 1
15:                    end if
16:                end for
17:                if session_available then
18:                    P(S)
19:                    for s ∈ sessions do
20:                        if s["n"] ≠ n or (s["in_progr"] and ((attacker or target) ∈ (s["attacker"] or s["target"]))
                           then
21:                            session_available ← False
22:                        end if
23:                    end for
24:                    V(S)
25:                    if session_available then
26:                        P(S)
27:                        sessions ← sessions ∪ {"num" : n, "attacker" : attacker, "target" : target}
28:                        V(S)
29:                        for proto ∈ AM[attacker][target][ip].keys() do
30:                            for port ∈ AM[attacker][target][ip][proto] do
31:                                run_attacks(attacker, target, proto, port, attack_modes)
32:                            end for
33:                        end for
34:                        P(S)
35:                        for s ∈ sessions do
36:                            if s["n"] == n then s["in_progr"] ← False end if
37:                        end for
38:                        V(S)
39:                    end if
40:                end if
41:            end for
42:        end for
43:    end for
44: end while
```

Une machine virtuelle ne peut faire partie que de l'exécution d'une session d'attaque à la fois (*i.e.*, attaquant ou cible), sinon les outils embarqués exécutés se perturberaient mutuellement. Si une session d'attaque n'est pas réalisable à cause de l'indisponibilité d'un des deux hôtes (déjà utilisé dans une autre session), elle est reportée et sera donc traitée plus tard, car les threads parcourent la matrice d'accessibilité tant qu'il y a des sessions à exécuter.

5.2.4 Calcul des métriques d'évaluation

La détermination automatique de mesures précises de l'efficacité des NIDS est une tâche assez complexe à mener en pratique. Nous reprenons les métriques traditionnelles évoquées dans le Chapitre 2, à savoir vrai positif (TP), faux positif (FP), vrai négatif (TN), faux négatif (FN), taux de détection (DR) et précision (PR). Nous avons décidé de faire en sorte que notre système soit capable de construire des métriques en menant une analyse automatique la plus proche possible de ce qu'un administrateur ferait en menant une analyse manuelle. Nous connaissons les attaques que nous avons exécutées, et nous disposons des alertes générées par les NIDS. Cependant, associer automatiquement une alerte à l'attaque correspondante (*i.e.*, trouver un TP) n'est pas aisé, et ce, même manuellement. D'après les alertes que nous avons étudiées pour différents produits de NIDS, les informations fournies sont souvent composées : du niveau de sévérité de l'alerte, de la date de l'alerte, des adresses IP, protocoles et ports impliqués, de l'identifiant de la signature associée et d'une description textuelle. Nous associons une alerte à une attaque (et donc validons un TP) si les conditions suivantes sont satisfaites :

- En considérant les horloges synchronisées, le délai entre la date de l'alerte et la date de l'attaque (moment où l'état final de l'automate est atteint) est plus petit qu'une certaine valeur. Cette valeur est appelée la fenêtre de détection (notée W).
- Les adresses IP, protocoles et ports impliqués dans l'alerte sont également impliqués dans l'attaque.

De manière optionnelle, nous ajoutons deux conditions (désactivées par défaut) pouvant être prises en compte dans la validation d'un TP :

- L'alerte est une alerte sévère.
- Les CVEs référencées dans la signature de l'alerte incluent la CVE associée à l'attaque.

En conséquent, le nombre de TP dépend fortement de la bonne synchronisation des horloges et de la fenêtre de détection autorisée. Les deux conditions optionnelles dépendent des caractéristiques des NIDS déployés, notamment de leur capacité à attribuer le bon niveau de sévérité aux alertes et à bien référencer les CVEs associées. Elles dépendent aussi des besoins de l'évaluateur, qui peut ne vouloir considérer que les alertes sévères, ou bien reposer sur des références pour valider la bonne détection des attaques. Néanmoins, nous avons remarqué que de nombreuses signatures ne référencent pas les mêmes CVEs que celles présentes dans notre dictionnaire d'attaques (dont les données

sont peuplées à partir de [9], où la plupart des références CVE proviennent de [4]). Potentiellement, plusieurs alertes peuvent être associées à une attaque. Les FP sont déduits en prenant en compte les alertes qui ne sont pas un TP et dont les adresses IP appartiennent à des hôtes faisant partie de l'infrastructure virtuelle ou appartenant à la machine externe. Les FN sont déterminés en vérifiant les attaques associées à aucun TP.

5.2.5 Analyse des incohérences de détection

A partir des attaques détectées et des attaques non détectées, nous sommes en mesure de préciser les incohérences dans la détection ou non détection de certaines attaques. Une incohérence de détection est une attaque non détectée mais dont l'exploit a été utilisé dans une autre attaque détectée (pour des hôtes ou sur un service différents). Pour chaque incohérence, nous sommes ainsi en mesure de donner les adresses IP, protocoles, ports et identifiants des signatures associées.

5.3 Conclusion

Dans ce chapitre, nous avons présenté la dernière phase du processus d'évaluation et d'analyse des mécanismes de sécurité réseau. Elle concerne l'évaluation des NIDS, guidée par la méthode d'exécution de campagnes d'attaque que nous avons présentée dans son ensemble. Elle comprend l'utilisation de trafic d'évaluation modélisé par des automates qui permettent de simuler le trafic nécessaire pour réaliser des sessions d'attaque et analyser ensuite la réaction des systèmes de détection d'intrusion par un calcul de métriques d'évaluation.

Dans le chapitre suivant, nous présentons la plateforme de maquettage et d'expérimentation qui nous a permis de valider le développement d'un prototype associé à l'approche présentée jusqu'ici. Nous décrivons également ce prototype, utilisé pour mener des expérimentations sur la plateforme afin de démontrer la faisabilité et l'efficacité des méthodes que nous proposons. Pour finir, nous expliquons nos démarches expérimentales, puis exposons et discutons les résultats d'expérimentation obtenus.

Chapitre 6

Prototype et expérimentations

Sommaire

6.1 Plateforme de maquettage et d'expérimentation 87
6.2 Prototype 89
6.2.1 Architecture logicielle 89
6.2.2 Intégration 94
6.3 Expérimentations et résultats 94
6.3.1 Scénario expérimental 94
6.3.2 Résultats expérimentaux 98
6.4 Conclusion 104

Dans ce chapitre, nous présentons d'abord la plateforme de maquettage et d'expérimentation qui a été le support pour le prototypage de notre approche. Puis, nous décrivons le prototype développé pour mettre en œuvre les différentes méthodes d'évaluation et d'analyse présentées dans les chapitres précédents. Enfin, nous présentons nos démarches expérimentales puis exposons et discutons les résultats obtenus après l'utilisation du prototype sur la plateforme pour un scénario donné.

6.1 Plateforme de maquettage et d'expérimentation

Pour pouvoir tester expérimentalement et valider notre prototype, nous avons mis en place une plateforme de cloud privé IaaS reposant sur la solution VMware vCloud Suite [30]. Le choix s'est porté sur cette technologie car, bien que propriétaire et soumise à licences payantes, elle est une des plus complètes notamment en termes de fonctionnalités offertes pour le réseau et la sécurité. C'était particulièrement le cas au début de ces travaux de thèse, même si des solutions alternatives intéressantes ont émergé ces derniers mois. En outre, les spécifications techniques du projet [1] dans lequel ces travaux de thèse s'inscrivent requéraient l'utilisation de la technologie VMware.

La plateforme est composée de quatre machines physiques :

– Un serveur Dell PowerEdge R320 muni d'un CPU Intel Xeon E5-2407, 8 Go de mémoire et deux disques durs de 2 To. Il est utilisé en tant que serveur NFS pour le stockage des disques durs virtuels des machines virtuelles du cloud.

1. Projet français Investissements d'Avenir Secured Virtual Cloud (SVC) (2012-2015)

– Deux serveurs Dell PowerEdge R620 munis chacun de deux CPUs Intel Xeon E5-2660, 64 Go de mémoire et deux disques durs de 146 Go. Ils exécutent l'hyperviseur VMware ESXi et hébergent les composants de la solution VMware vCloud Suite sous forme de machines virtuelles.
– Un commutateur réseau HP 5120-24G EI. Il interconnecte les trois serveurs.

La solution VMware vCloud Suite comprend VMware vCenter, vCloud Director et vShield Manager. vCenter permet de gérer l'infrastructure de virtualisation contenant les hyperviseurs, les commutateurs virtuels et les machines virtuelles. vCloud Director offre la possibilité de gérer des clients du cloud IaaS et leur donne accès à une interface Web pour paramétrer leur infrastructure virtuelle. Les informations d'administration des clients sont stockées dans un serveur de base de données Microsoft SQL Server 2008. Enfin, vShield Manager apporte des fonctionnalités de sécurité avec un pare-feu en mode hyperviseur (appelé vShield App par VMware), un antivirus (appelé Endpoint par VMware) et un antispoofing (appelé SpoofGuard par VMware).

Un commutateur virtuel distribué (c'est-à-dire réparti sur les deux hyperviseurs) connecte toutes les machines virtuelles de la plateforme. Il est indirectement connecté au commutateur physique par un lien *trunk*. Ce lien est composé d'un segment virtuel reliant le commutateur virtuel aux interfaces réseau des deux hyperviseurs, et d'un segment physique entre ces interfaces et le commutateur physique. Les réseaux virtuels sont représentés sous forme de groupes de ports sur le commutateur virtuel distribué. Il y a donc un groupe de port par réseau virtuel de clients. Un réseau virtuel de gestion, associé à un VLAN spécifique, interconnecte toutes les machines virtuelles de gestion du cloud et tous les réseaux virtuels des clients. Il s'agit donc du réseau externe, dans lequel on retrouve également les passerelles éventuelles (pare-feu en mode pont, appelés vShield Edge par VMware) des réseaux virtuels des clients ainsi que la machine virtuelle d'audit. De plus, deux solutions de NIDS sont déployées dans ce réseau : Suricata 2.0.5 et Snort 2.9.2.2. Chacun de ces NIDS est installé dans une machine virtuelle munie de 4 CPUs et 8 Go de mémoire. Nous avons conservé la configuration par défaut de ces NIDS, en leur fournissant le dernier jeu de signatures issu de [26]. Un mécanisme de mise en miroir des ports ou *port mirroring* est appliqué sur le commutateur virtuel distribué, de manière à envoyer une copie de tout le trafic des machines virtuelles du cloud vers les deux NIDS.

La Figure 6.1 illustre l'infrastructure physique et virtuelle de la plateforme d'expérimentation. Nous y avons représenté les quatre machines physiques (à savoir les trois serveurs et le commutateur), les solutions de gestion du cloud, ainsi que les liens physiques (reliant le commutateur physique aux serveurs physiques) et virtuels (reliant le commutateur vituel distribué aux machines virtuelles et aux serveurs physiques). Les liens physiques sont au nombre de deux pour chaque serveur physique hébergeant l'hyperviseur ESXi, où un des liens d'accès est utilisé pour administrer directement le serveur (qui est en fait également dans le réseau de gestion associé au VLAN dédié), et l'autre est le lien *trunk* reliant le commutateur physique au commutateur virtuel distribué (hébergé de manière identique sur l'hyperviseur de chaque serveur). Il est à noter que le lien série représenté sert à administrer le commutateur physique.

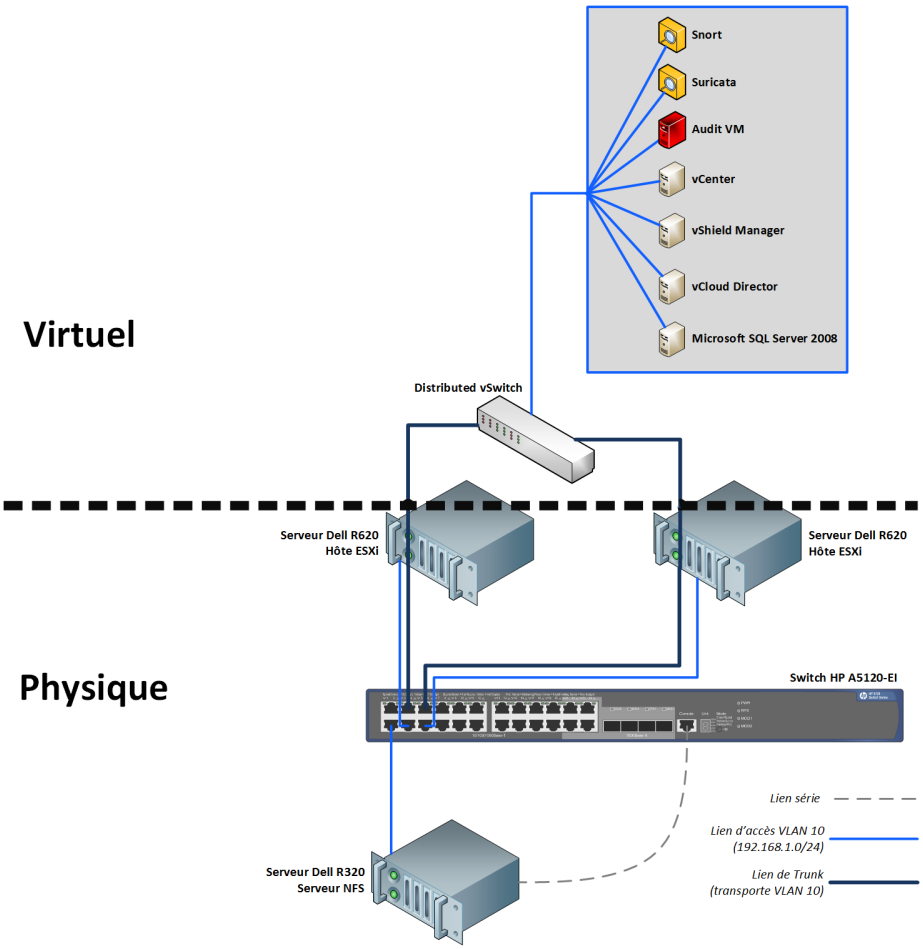


FIGURE 6.1 – Infrastructure physique et virtuelle de la plateforme d'expérimentation

6.2 Prototype

Le prototype correspond à une implémentation des méthodes présentées dans le cadre de l'approche pour l'analyse des contrôles d'accès réseaux et l'évaluation des NIDS dans des infrastructures virtuelles de cloud computing.

6.2.1 Architecture logicielle

Le prototype développé est compatible avec des clouds utilisant la solution IaaS VMware vCloud Suite. Cependant, il a été développé de manière à être extensible à d'autres solutions IaaS. Il est composé de deux éléments principaux :

– La machine d'audit qui consiste à orchestrer les opérations d'audit et sur laquelle s'exécutent les méthodes d'évaluation et d'analyses présentées. Elle est livrée en tant qu'archive *Open Virtualization Archive* (OVA) au format *Open Virtualiza-*

tion Format (OVF) [75], de manière à pouvoir être déployée en tant que machine virtuelle sur n'importe quel hyperviseur.
– Le modèle de machine virtuelle utilisé pour l'importation de machines virtuelles durant la phase de clonage (comme expliqué dans le Chapitre 3) et sur lequel sont installés les outils pour l'injection de trafic. Il est livré en tant qu'archive OVA au format OVF, de manière à pouvoir être déployé en tant que machine virtuelle sur n'importe quel hyperviseur lors de son importation.

6.2.1.1 Machine d'audit

La machine d'audit est munie de 4 CPUs, 4 Go de mémoire et 30 Go de disque, et héberge le système GNU/Linux Debian. Elle permet l'interaction avec l'utilisateur du système d'audit pour récupérer les paramètres d'entrée nécessaires à chaque phase de l'audit. Ces paramètres sont les suivants :

- Pour la connexion au cloud :
  - Les adresses IP de vCloud Director, vShield Manager et vCenter.
  - Les identifiants de connexion administrateur vCloud Director, vShield Manager et vCenter.
- Pour le clonage :
  - Le nom ou identifiant du client.
  - Les adresses IP réservées ne pouvant pas être utilisées pendant le clonage.
- Pour l'analyse des contrôles d'accès réseau :
  - Le nom ou identifiant du client.
  - La liste des services TCP, UDP, ICMP à tester.
  - Le nombre de fois où l'algorithme d'analyse dynamique doit être effectué (une fois par défaut) [2].
  - Le délai entre chaque envoi de paquet lors de l'analyse dynamique (10ms par défaut).
- Pour l'évaluation des NIDS :
  - Le nom ou identifiant du client.
  - Les modes d'attaque (toutes les *attaques standards* disponibles et deux *attaques non standards* aléatoires par accessibilité par défaut).
  - La fenêtre de détection autorisée (15s par défaut).
  - Les URLs du fichier d'alertes de chaque NIDS évalué.

La machine d'audit intéragit avec les APIs du cloud pour mener le processus d'audit. Puis, elle génère en sortie les rapports d'évaluation et d'analyse. Ces rapports comprennent :

- Pour l'analyse des contrôles d'accès réseau :
  - La durée du parcours des configurations.
  - La durée de la résolution logique des accessibilités.

2. Nous laissons la possibilité de consolider les résultats d'accessibilité, car des campagnes de test comprenant de très nombreux envois de paquets peuvent très occasionnellement engendrer des problèmes de congestion sur des pare-feu de petite capacité.

  - La liste des accessibilités *configurées* sous forme de prédicats d'accessibilité.
  - La liste des accessibilités *observées* sous forme de prédicats d'accessibilité.
  - Le graphe des accessibilités *configurées*.
  - Le graphe des accessibilités *observées*.
  - La liste des déviances.
- Pour l'évaluation des NIDS :
  - Le nombre de sessions d'attaque.
  - Le nombre d'accessibilités.
  - Le nombre d'attaques lancées.
  - Les dates de début et de fin de la campagne d'attaque.
  - La durée de la campagne d'attaque.
  - Le nombre d'attaques réussies.
  - Pour chaque NIDS évalué, le nombre de TP, FN et FP, le DR, le PR, et la liste des incohérences de détection.

Il y a dix modules principaux pour orchestrer les opérations d'évaluation et d'analyse depuis la machine d'audit. Leurs interactions sont représentées sur la Figure 6.2.

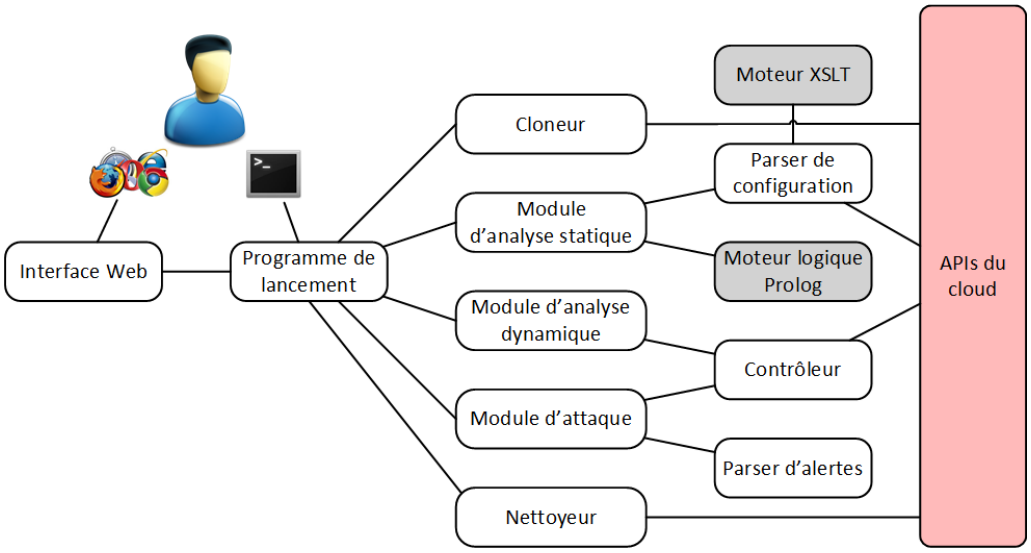


FIGURE 6.2 – Interactions entre les composants du prototype

Contrôleur

Le contrôleur permet de contrôler les machines virtuelles de l'infrastructure clone afin d'y effectuer les opérations suivantes : connexion, inventaire, envoi, récupération et suppression de fichiers, démarrage, terminaison et surveillance de l'état de processus. Le contrôleur constitue en fait une surcharge des méthodes offertes par les APIs cloud. Le but est que ces méthodes soient les mêmes quelle que soit la technologie utilisée, afin de ne pas à avoir à modifier le code des modules appelant ces méthodes lorsqu'on change de technologie cloud. Nous avons implémenté un contrôleur pour vCloud en

nous appuyant sur l'API Python *pysphere* [21].

Cloneur

Le cloneur (implémentant des concepts présentés dans le Chapitre 3) a pour rôle de cloner les composants de l'infrastructure virtuelle d'un client vers une infrastructure virtuelle dédiée aux audits de sécurité. Il intéragit directement avec les APIs vCloud Director pour le clonage des vDCs, des réseaux virtuels, des pare-feu virtuels et des vApp ; avec vShield pour la mise à jour des pare-feu en mode hyperviseur ; avec *pysphere* pour l'importation des machines virtuelles durant le clonage des vApp.

Parser de configuration

Le parser de configuration (implémentant des concepts présentés dans le Chapitre 4) parcourt la configuration du cloud en interrogeant les APIs du cloud afin de générer les prédicats de configuration. Nous avons implémenté un parser pour vCloud en nous appuyant sur l'API vCloud Director, vShield et *pysphere*. Il utilise principalement le processeur XSLT *xsltproc* [28] pour transformer les informations XML en prédicats Prolog. Lorsque les informations de configuration sont suffisamment concises, elles sont directement transformées par manipulation de chaînes de caractères en Python.

Module d'analyse statique

Le module d'analyse statique (implémentant des concepts présentés dans le Chapitre 4) s'appuie sur le parser de configuration pour générer un script Prolog contenant : les prédicats des configurations, les règles correspondant à l'algorithme de résolution logique des accessibilités, et les requêtes d'accessibilités réparties sur plusieurs threads pour être parallélisées. Les accessibilités calculées avec ce script sont stockées dans des fichiers sur le disque, parcourus pour générer une matrice (sous forme de dictionnaire Python) et un graphe d'accessibilité (sous forme de graphe DOT à l'aide de la librairie *pydot* [20]).

Module d'analyse dynamique

Le module d'analyse dynamique (implémentant des concepts présentés dans le Chapitre 4) exécute l'algorithme d'envoi de paquets réseau en utilisant le contrôleur pour lancer les programmes de client et serveur dans les machines virtuelles. Les accessibilités *observées* sont stockées dans des fichiers sur le disque, eux-même parcourus pour générer une matrice (sous forme de dictionnaire Python) et un graphe d'accessibilité (sous forme de graphe DOT à l'aide de la librairie *pydot*).

Parser d'alerte

Le parser d'alerte (implémentant des concepts présentés dans le Chapitre 5) parcourt les fichiers contenant les alertes générées par les NIDS et récupère les valeurs des champs des alertes pour créer des alertes sous forme d'objets Python. Nous avons implémenté un parser JSON pour Suricata, un parser CSV pour Snort, ainsi qu'un parser IDMEF générique.

Module d'attaque

Le module d'attaque (implémentant des concepts présentés dans le Chapitre 5) exécute l'algorithme d'exécution des campagnes d'attaque en utilisant le contrôleur pour lancer les programmes d'exploit et les clients en ligne de commande, ainsi que l'*automata player* dans les machines virtuelles. Les attaques réussies sont confrontées aux alertes récupérées à l'aide du parser d'alerte afin de calculer les métriques d'évaluation des NIDS.

Nettoyeur

Le nettoyeur (implémentant des concepts présentés dans le Chapitre 3) a pour rôle de supprimer les composants de l'infrastructure virtuelle clone après les opérations d'audit. Il intéragit directement avec les APIs vCloud Director pour la suppression des vDCs, des réseaux virtuels, et des vApp ; avec vShield pour la suppression des pare-feu en mode pont.

Interface Web et programme de lancement

L'interface Web, développée avec le framework Python *web2py* [33], offre un formulaire pour fournir les paramètres d'entrée pour le clonage, l'analyse statique et dynamique des contrôles d'accès, et l'évaluation des NIDS. Les autres paramètres relatifs à l'accès aux APIs et aux alertes des NIDS sont spécifiés dans un fichier de configuration. Sur soumission du formulaire, un programme Python de lancement de l'audit est exécuté. Ce programme est également directement utilisable en ligne de commande depuis la machine d'audit.

6.2.1.2 Modèle de machine virtuelle

Le modèle de machine virtuelle est configuré avec 1 CPU, 1 Go de mémoire et 20 Go de disque, et héberge le système GNU/Linux Debian. Il est utilisé lors du clonage de l'infrastructure virtuelle afin de remplacer les machines virtuelles de l'infrastructure clone par des machines virtuelles instanciées à partir de ce modèle. Plusieurs éléments sont installés dans ce modèle :

- Le programme Python du client pour envoyer les paquets lors de l'analyse dynamique.
- Le programme Python du serveur pour recevoir les paquets lors de l'analyse dynamique.
- La liste des services TCP, UDP et ICMP à tester pour l'analyse dynamique.
- Le framework de test de pénétration Metasploit.
- Le script Ruby d'exécution des exploits qui s'interface avec Metasploit.
- Les scripts Expect et Shell pour l'exécution des requêtes légitimes.
- Le dictionnaire d'attaques sous forme de base de données SQLite3.
- Le programme Python *automata player* pour rejouer les comportements réseau des applications.

6.2.2 Intégration

Pour intégrer le prototype dans un environnement de cloud de type IaaS, il suffit de procéder aux deux étapes principales suivantes :

1. Télécharger les archives OVA de la machine d'audit et du modèle de machine virtuelle sur un espace de stockage accessible par les solutions de gestion du cloud.
2. Déployer à partir de leur archive OVA la machine virtuelle d'audit et le modèle de machine virtuelle dans un réseau externe aux réseaux virtuels des clients.

Ensuite, il y a deux manières d'utiliser le prototype :

- En se connectant sur la machine d'audit et en utilisant le programme en ligne de commande pour lancer les audits.
- En se rendant sur l'interface Web (à l'adresse IP de la machine d'audit) et en utilisant le formulaire d'audit.

6.3 Expérimentations et résultats

Nous décrivons d'abord le scénario expérimental que nous avons mis en place sur notre plateforme, puis nous présentons les expérimentations que nous avons réalisées ainsi que les résultats qui en découlent.

6.3.1 Scénario expérimental

Nous avons déployé une infrastructure virtuelle fictive de petite taille, qui pourrait convenir à un client ayant des besoins limités. Elle est constituée d'un centre de données virtuel composé de quatre réseaux virtuels, qui comprennent chacun un pare-feu en mode pont et deux machines virtuelles ayant chacune une adresse IP. On a donc un total de quatre pare-feu en mode pont et huit machines virtuelles. Ce déploiement est représenté par la Figure 6.3.

Le Tableau 6.1 constitue la matrice d'accessibilité définie pour ce scénario.

		Destinations								
		172.16.2.1	172.16.2.3	172.16.2.66	172.16.2.67	172.16.2.129	172.16.2.130	172.16.2.193	172.16.2.194	0.0.0.0
Sources	vm-5793		TCP 80, 8080 UDP 67, 68	TCP 25, 143						
	vm-5796	TCP 21 UDP 67, 68								
	vm-5792				UDP 67, 68 ICMP echo-reply					TCP 25
	vm-5804	TCP 21		TCP 25 UDP 67, 68 ICMP echo-request						
	vm-5821		TCP 80						TCP 80	TCP 80
	vm-5811			TCP 25, 143						
	vm-5816	TCP 21								TCP 21
	vm-5815						TCP 25			
	0.0.0.0		TCP 80, 8080	TCP 143					TCP 80	

Tableau 6.1 – Matrice d'accessibilité définie pour le scénario expérimental

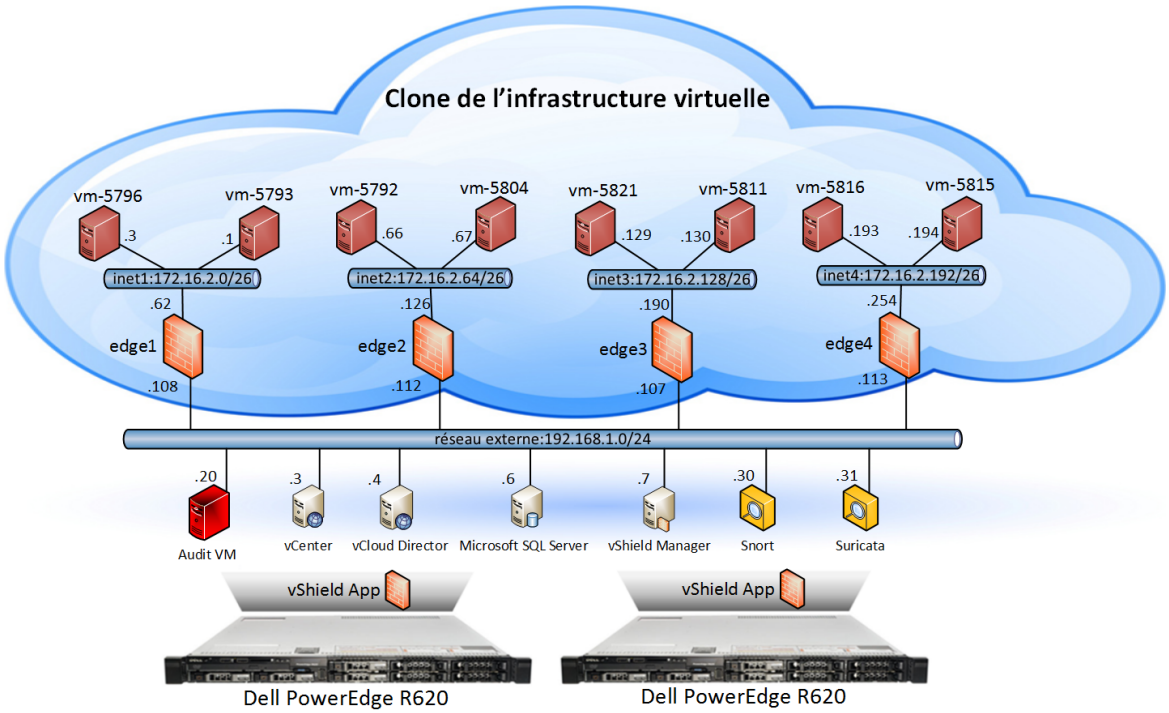


FIGURE 6.3 – Scénario déployé sur la plateforme d'expérimentation

Nous avons mis en œuvre cette matrice sous forme de règles de filtrage sur les pare-feu en mode pont et les pare-feu en mode hyperviseur. Par ailleurs, nous avons configuré des règles de routage sur chaque pare-feu en mode pont de manière à permettre le routage des paquets vers tous les autres réseaux virtuels. Les Figures 6.4, 6.5, 6.6, et 6.7 montrent les règles de filtrage des pare-feu en mode pont. Les Figures 6.8, 6.9, 6.10, 6.11 et 6.12 présentent les règles de filtrage des pare-feu hyperviseur appliquées sur les quatre réseaux virtuels et sur le réseau externe.

Enable firewall

Default action ◉ Deny ○ Allow ☐ Log

Applicable to traffic that does not match the rules in the list.

Rule Id	Name	Source	Destination	Protocol	Action	Log	Enabled
4	HTTP in	external:Any	internal:80	TCP	Allow	-	✔
5	HTTP-ALT in	external:Any	internal:8080	TCP	Allow	-	✔
2	FTP in 1	172.16.2.64/26:Any	172.16.2.0/26:21	TCP	Allow	-	✔
6	FTP in 2	172.16.2.192/26:Any	172.16.2.0/26:21	TCP	Allow	-	✔
1	SMTP out	internal:Any	172.16.2.64/26:25	TCP	Allow	-	✔
3	IMAP out	internal:Any	172.16.2.64/26:143	TCP	Allow	-	✔

FIGURE 6.4 – Règles de filtrage du pare-feu en mode pont edge-1



Enable firewall

Default action Deny Allow Log

Applicable to traffic that does not match the rules in the list.

Rule Id	Name	Source	Destination	Protocol	Action	Log	Enabled
1	SMTP in 1	172.16.2.0/26:Any	172.16.2.64/26:25	TCP	Allow	-	✔
5	SMTP in 2	172.16.2.128/26:Any	172.16.2.64/26:25	TCP	Allow	-	✔
3	IMAP in	external:Any	172.16.2.64/26:143	TCP	Allow	-	✔
2	SMTP out	internal:Any	Any:Any	TCP	Allow	-	✔
4	FTP out	internal:Any	172.16.2.0/26:21	TCP	Allow	-	✔

FIGURE 6.5 – Règles de filtrage du pare-feu en mode pont edge-2



Enable firewall

Default action Deny Allow Log

Applicable to traffic that does not match the rules in the list.

Rule Id	Name	Source	Destination	Protocol	Action	Log	Enabled
1	HTTP out	172.16.2.129:Any	external:80	TCP	Allow	-	✔
2	SMTP out	172.16.2.130:Any	172.16.2.64/26:25	TCP	Allow	-	✔
3	IMAP out	172.16.2.130:Any	172.16.2.64/26:143	TCP	Allow	-	✔
4	SMTP in	172.16.2.192/26:Any	internal:25	TCP	Allow	-	✔

FIGURE 6.6 – Règles de filtrage du pare-feu en mode pont edge-3



Enable firewall

Default action Deny Allow Log

Applicable to traffic that does not match the rules in the list.

Rule Id	Name	Source	Destination	Protocol	Action	Log	Enabled
3	HTTP in	external:Any	172.16.2.194:80	TCP	Allow	-	✔
1	SMTP out	172.16.2.194:Any	172.16.2.128/26:25	TCP	Allow	-	✔
2	FTP out	172.16.2.193:Any	external:21	TCP	Allow	-	✔

FIGURE 6.7 – Règles de filtrage du pare-feu en mode pont edge-4

General | Ethernet

Search

No.	Name	Source	Destination	Service	Action
1	DHCP	172.16.2.0/26 0.0.0.0	255.255.255.255 172.16.2.0/26	DHCP-Server DHCP-Client	Allow
2	WEB in	* any	172.16.2.3	WEB	Allow
3	FTP in	172.16.2.64/26 172.16.2.3 172.16.2.192/26	172.16.2.1	FTP	Allow
4	MAIL out	172.16.2.1	172.16.2.64/26	SMTP IMAP	Allow
5	Default Rule	* any	* any	* any	Block

FIGURE 6.8 – Règles de filtrage du pare-feu en mode hyperviseur pour le réseau inet1

General | Ethernet

Search

No.	Name	Source	Destination	Service	Action
1	DHCP	172.16.2.64/26 0.0.0.0	255.255.255.255 172.16.2.64/26	DHCP-Server DHCP-Client	Allow
2	ECHO-REQUEST	172.16.2.67	172.16.2.66	ECHO-REQUEST	Allow
3	ECHO-REPLY	172.16.2.66	172.16.2.67	ECHO-REPLY	Allow
4	IMAP in restrict	172.16.2.67	172.16.2.66	IMAP	Block
5	MAIL in	* any	172.16.2.66	SMTP IMAP	Allow
6	SMTP out restrict	172.16.2.66	172.16.2.67	SMTP	Block
7	SMTP out	172.16.2.66	* any	SMTP	Allow
8	FTP out	172.16.2.67	172.16.2.0/26	FTP	Allow
9	Default Rule	* any	* any	* any	Block

FIGURE 6.9 – Règles de filtrage du pare-feu en mode hyperviseur pour le réseau inet2

General | Ethernet

Search

No.	Name	Source	Destination	Service	Action
1	SMTP in	172.16.2.192/26	172.16.2.130	SMTP	Allow
2	HTTP in restrict	* any	172.16.2.128/26	HTTP	Block
3	HTTP out	172.16.2.129	* any	HTTP	Allow
4	MAIL out	172.16.2.130	172.16.2.64/26	MAIL	Allow
5	Default Rule	* any	* any	* any	Block

FIGURE 6.10 – Règles de filtrage du pare-feu en mode hyperviseur pour le réseau inet3

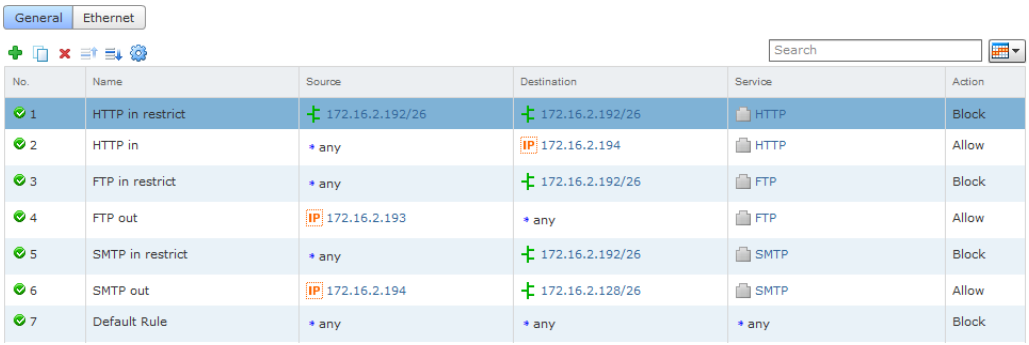


FIGURE 6.11 – Règles de filtrage du pare-feu en mode hyperviseur pour le réseau inet4

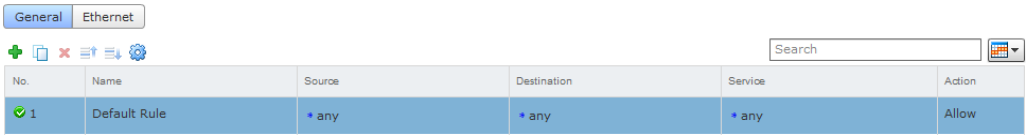


FIGURE 6.12 – Règles de filtrage du pare-feu en mode hyperviseur pour le réseau externe

6.3.2 Résultats expérimentaux

Après la phase de clonage de l'infrastructure, nous avons effectué, sur l'infrastructure clone, une analyse statique et une analyse dynamique des contrôles d'accès, puis une évaluation des NIDS. Nous présentons et discutons les résultats obtenus à l'issue de chacune de ces deux étapes.

6.3.2.1 Analyse des contrôles d'accès réseau

Notre objectif ici est de fournir une analyse de l'implémentation des règles de contrôles d'accès au sein de l'infrastructure virtuelle considérée.

Analyse statique

Nous avons considéré les 587 services TCP, UDP et ICMP connus utilisés par défaut. Le parcours des configurations a duré 2mn28s. La résolution logique des accessibilités a nécessité l'exécution de 42264 requêtes d'accessibilité, et a duré 2mn14s. La Figure 6.13 montre le graphe d'accessibilité obtenu. Au total, l'analyse statique a duré 4mn42s.

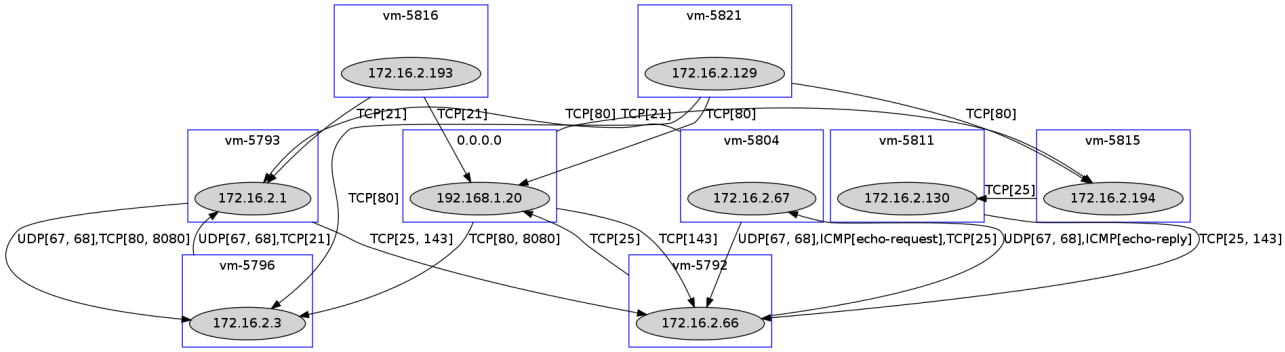


FIGURE 6.13 – Graphe d'accessibilité généré par l'analyse statique

Analyse dynamique

Nous avons considéré les mêmes services TCP, UDP et ICMP que lors de l'analyse statique et un délai de 10ms entre chaque envoi de paquet. L'exécution de l'algorithme d'analyse dynamique a nécessité l'envoi de 42264 paquets réseau, et a duré 7mn08s. Étant donné la taille de l'infrastructure virtuelle, nous avons choisi de l'exécuter deux fois afin de ne conserver que les accessibilités trouvées les deux fois. En effet, au cours de nos tests incluant des balayages de ports à grande échelle comme ici, il est arrivé, rarement et de façon aléatoire, que quelques paquets non autorisés parviennent à atteindre leur destination. Ceci peut être dû à des problèmes de congestion sur les pare-feu en mode pont de petite capacité, qui agissent alors en *fail-open* pour les paquets non analysés. La deuxième exécution de l'algorithme a duré 7mn06s. La Figure 6.14 montre le graphe d'accessibilité obtenu. Au total, l'analyse dynamique a duré 14mn14s.

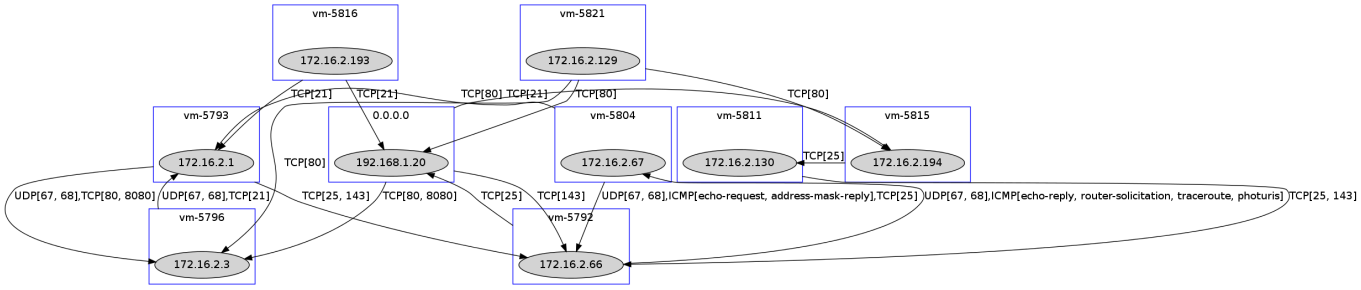


FIGURE 6.14 – Graphe d'accessibilité généré par l'analyse dynamique

Analyse des déviances

En comparant les accessibilités *définies*, *configurées* et *observées*, on constate les déviances suivantes :

- Le trafic ICMP router-solicitation, traceroute et photuris entre vm-5792 et 172.16.2.67 n'est pas *configuré* mais est *observé*.
- Le trafic ICMP address-mask-reply entre vm-5804 et 172.16.2.66 n'est pas *configuré* mais est *observé*.

Pour essayer de comprendre le phénomène, nous avons eu le réflèxe qu'aurait eu un administrateur sécurité, à savoir que nous avons explicitement ajouté sur le pare-feu hyperviseur une règle qui interdit tout trafic ICMP address-mask-reply pour les réseaux virtuels concernés. Cette règle est munie d'une priorité plus forte que celle autorisant le trafic ICMP echo-request. Les accessibilités obtenues par l'analyse statique sont les mêmes, ce qui est logique. Les accessibilités obtenues par l'analyse dynamique n'ont pas changé non plus, ce qui est inattendu, et les déviances découvertes sont donc les mêmes malgré l'ajout de cette règle bloquante. Nous constatons donc que les pare-feu virtuels de VMware laissent passer plus de trafic de gestion ICMP que configuré. Ceci est probablement dû à l'utilisation d'un mécanisme de suivi d'états trop permissif pour les flux ICMP, dont l'application laisse ainsi passer trop de paquets à la fois. La présence de telles déviances montre l'intérêt de notre approche, qui vise à confronter les matrices d'accessibilité d'une même infrastructure obtenues à partir de connaissances et techniques différentes.

6.3.2.2 Évaluation des systèmes de détection d'intrusion réseau

Notre objectif ici n'est pas de fournir une évaluation de type benchmarking de produits de NIDS. En effet, nous souhaitons évaluer l'efficacité des NIDS déployés pour protéger l'infrastructure virtuelle considérée.

Trafic d'évaluation

Notre prototype cible quatre des services les plus courants d'Internet : HTTP, FTP, SMTP, IMAP. Nous avons choisi quinze applications vulnérables issues de [9] et dont les exploits correspondants sont disponibles dans Metasploit. Cette sélection couvre un ensemble de vulnérabilités publiées entre 2002 et 2014. Nous avons généré les automates comme expliqué dans le Chapitre 5, de même que les scripts de requêtes légitimes. Nous avons manuellement testé le fonctionnement de ces automates pour être sûrs de pouvoir les rejouer de manière automatisée. Le Tableau 6.2 montre les entrées associées au dictionnaire d'attaques, ainsi que les nombres d'états des automates légitimes (noté N_{sl}) et des automates malveillants (noté N_{sm}). Il est à noter que le nombre d'états des automates est assez limité (inférieur ou égal à treize).

Campagnes d'attaque

Nous avons exécuté plusieurs campagnes d'attaque, en choisissant différents modes d'attaque. Pour chaque campagne, nous avons choisi le mode d'attaque 1 pour la catégorie d'*attaques standards* et le mode d'attaque 3 avec une valeur différente de n pour la catégorie d'*attaques non standards*. Ainsi, il sera lancé pour chacune des accessibilités :

- Toutes les *attaques standards* et activités légitimes associées disponibles pour le service de l'accessibilité.
- n *attaques non standards* et activités légitimes associées choisies aléatoirement pour le service de l'accessibilité.

exploit_id	*cve*	*proto*	*port*	*date*	*description*	N_{sl}	N_{sm}
35660	2014-9567	TCP	80	2014-12-02	ProjectSend Arbitrary File Upload	3	3
34926	2014-6287	TCP	80	2014-09-11	Rejetto HttpFileServer Remote Command Execution	12	12
33790	N/A	TCP	80	2014-05-20	Easy File Management Web Server Stack Buffer Overflow	7	8
25775	2013-2028	TCP	80	2013-05-07	Nginx HTTP Server 1.3.9-1.4.0 - Chuncked Encoding Stack Buffer Overflow	3	8
16970	2002-2268	TCP	80	2010-12-26	Kolibri 2.0 - HTTP Server HEAD Buffer Overflow	2	2
16806	2007-6377	TCP	80	2007-12-10	BadBlue 2.72b PassThru Buffer Overflow	9	3
28681	N/A	TCP	21	2013-08-20	freeFTPd PASS Command Buffer Overflow	11	4
24875	N/A	TCP	21	2013-02-27	Sami FTP Server LIST Command Buffer Overflow	11	3
17355	2006-6576	TCP	21	2011-01-23	GoldenFTP 4.70 PASS Stack Buffer Overflow	13	7
16742	2006-3952	TCP	21	2006-07-31	Easy File Sharing FTP Server 2.0 PASS Overflow	11	6
16713	2006-2961	TCP	21	2006-06-12	Cesar FTP 0.99g MKD Command Buffer Overflow	11	6
16821	2007-4440	TCP	25	2007-08-18	Mercury Mail SMTP AUTH CRAM-MD5 Buffer Overflow	9	8
16822	2004-1638	TCP	25	2004-10-26	TABS MailCarrier 2.51 - SMTP EHLO Overflow	6	6
16476	2006-1255	TCP	143	2006-03-17	Mercur 5.0 - IMAP SP3 SELECT Buffer Overflow	7	4
16474	2005-4267	TCP	143	2005-12-20	Qualcomm WorldMail 3.0 IMAPD LIST Buffer Overflow	2	2

Tableau 6.2 – Entrées du dictionnaire d'attaques et nombre d'états des automates

Les paramètres des campagnes d'attaque sont les suivants :

- W : la fenêtre de détection.
- S : indique s'il faut aussi calculer les métriques d'évaluation en ne traitant que les alertes sévères.
- n : le nombre de paires d'*attaques non standards* et activités légitimes associées choisies aléatoirement pour chaque accessibilité.
- N_s : le nombre total de paires d'*attaques standards* et activités légitimes associées lancées.
- N_n : le nombre total de paires d'*attaques non standards* et activités légitimes associées lancées.
- N_l : le nombre total de paires d'attaques et activités légitimes associées lancées.

Nous sommes intéressés par l'efficacité du prototype, caractérisée par :

- N_c : le nombre total de paires d'attaques et activités légitimes associées réussies, celles dont les automates associés ont atteint leur état final.
- C : le rapport de complétude, donné par $\frac{N_c}{N_l}$.
- La durée de l'exécution de chaque campagne d'attaque.

Nous avons exécuté sept campagnes d'attaque, en faisant varier n de 0 à 6. La valeur de N_s reste constante car nous lançons toujours toutes les *attaques standards* disponibles pour chaque accessibilité. Les valeurs de N_n, N_l et N_c augmentent en fonction de n.

W dépend de l'environnement des expérimentations. Plus W est petit, plus le risque d'omettre des alertes est grand. Plus W est grand, plus le risque d'augmenter le nombre de fausses alarmes est grand. Dans notre environnement, étant donné le temps requis pour la mise en miroir de port et la transmission vers les NIDS, ainsi que le temps nécessaire aux NIDS pour traiter chaque paquet et rapporter les alertes, nous avons choisi 15s pour W. C'est assez grand pour qu'aucune alerte ne soit oubliée, et pas trop grand pour éviter que de fausses alertes additionnelles soient prises en compte. Nous n'avons pas appliqué la vérification de références CVE pour ne pas être trop restrictif dans le traitement des alertes. Cependant, pour chaque campagne d'attaque, nous avons appliqué le paramètre S pour traiter seulement les alertes sévères d'une part, en plus de traiter toutes les alertes d'autre part. L'IDS Snort ne fournissant pas un indicateur de sévérité dans ses alertes, elles sont toutes considérées au même niveau. Les résultats des campagnes d'attaque sont présentés dans le Tableau 6.3.

Paramètres														
n	0		1		2		3		4		5		6	
N_s	72		72		72		72		72		72		72	
N_n	0		20		40		60		80		100		120	
N_l	72		92		112		132		152		172		192	
Efficacité														
N_c	69		89		109		128		146		165		185	
C	95,83%		96,74%		97,32%		96,97%		96,05%		95,93%		96,35%	
Durée	14mn58s		17mn47s		20mn40s		23mn02s		27mn21s		31mn35s		33mn15s	
Suricata														
S	no	yes	no	yes	no	yes	no	yes	no	yes	no	yes	no	yes
# de TP	20	20	30	28	37	36	42	40	47	44	54	45	64	54
# de FN	49	49	59	61	72	73	86	88	99	102	111	120	121	131
# de FP	5	5	9	8	12	9	17	13	22	16	29	21	23	15
DR	28,99%	28,99%	33,71%	31,47%	33,94%	33,03%	32,81%	31,25%	32,19%	30,14%	32,73%	27,27%	34,59%	29,19%
PR	80%	80%	76,92%	77,78%	75,51%	80%	71,19%	75,47%	67,12%	73,33%	65,06%	68,18%	73,56%	78,26%
Snort														
# de TP	43		62		75		79		99		108		116	
# de FN	26		27		34		49		47		57		69	
# de FP	31		36		53		85		74		129		79	
DR	62,32%		69,66%		68,81%		61,72%		67,81%		65,45%		62,70%	
PR	58,11%		63,27%		58,59%		48,17%		57,23%		45,57%		59,49%	

Tableau 6.3 – Résultats des campagnes d'attaque

Discussion des résultats

A partir des résultats, on constate que notre prototype est capable de mener des campagnes d'attaque avec une bonne efficacité et une durée acceptable. En effet, C est toujours au moins supérieur à $95,83\%$, et donc seulement moins de $4,17\%$ des attaques ont échoué. Au travers des campagnes d'attaque, les quelques attaques ayant échoué furent toujours des attaques différentes. En effet, les quelques échecs sont dus à des bugs logiciels aléatoires, venant soit de l'API de VMware (intérrogée assez intensément), soit du framework Metasploit. Concernant la durée d'exécution, elle a une évolution linéaire dépendant de N_c et on note que notre prototype peut exécuter plus de cinq attaques par minute.

Globalement, on note que pour les deux NIDS, le taux de détection reste stable lorsqu'on augmente n. En revanche, la précision a tendance à diminuer, car le nombre de faux positifs augmente plus rapidement que le nombre de vrais positifs au fil des campagnes. Il est à noter que les faux positifs peuvent être générés par le trafic légitime que nous exécutons, mais aussi par un éventuel trafic de fond que nous ne maîtrisons

pas. Pour Suricata, lorsqu'on ne considère que les alertes sévères, le nombre de vrais positifs diminue, ce qui rend le taux de détection légèrement moins élevé que dans le cas où on considère toutes les alertes. En revanche, la précision est améliorée car le nombre de faux positifs est réduit. Dans l'ensemble, par rapport aux campagnes d'attaque que nous avons menées, on constate que dans notre environnement Suricata offre une moins bonne capacité de détection, mais est plus précis. Ces observations sont illustrées dans les Figures 6.15 et 6.16. Bien que notre objectif ne soit pas de fournir une évaluation complète des produits Snort et Suricata, nous avons été surpris par certaines valeurs de DR et PR. Après avoir manuellement exécuté les attaques individuellement et vérifié les alertes levées par les NIDS, nous nous sommes rendus compte que ceci est principalement dû au manque de certaines signatures et à la présence de nombreuses signatures génériques se déclenchant assez facilement. Il est à noter que de faibles taux de détection et précision pour des produits de NIDS, incluant Snort, ont déjà été reportés dans d'autres travaux (*e.g.*, [67, 99, 108]).

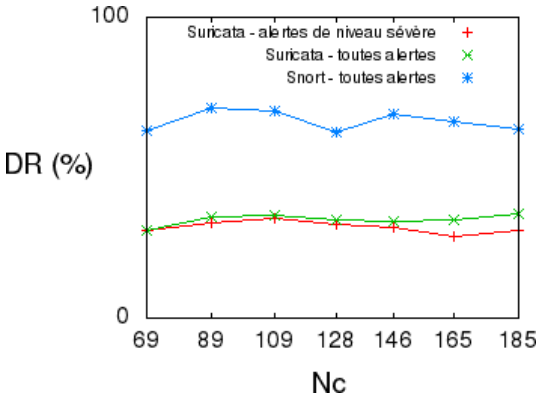


FIGURE 6.15 – Taux de détection des NIDS

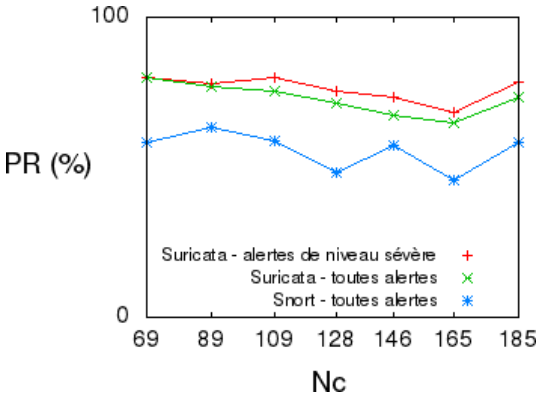


FIGURE 6.16 – Précision des NIDS

Enfin, concernant l'analyse des incohérences de détection, introduite en fin de Chapitre 5, nous ne présentons pas toutes les incohérences identifiées pour toutes les campagnes, mais seulement quelques exemples pour la dernière campagne d'attaque (où n vaut 6) :

- L'attaque utilisant l'exploit 16970 n'a pas été détectée de vm-5793 vers 172.16.2.66 sur TCP port 25, mais a été détectée de vm-5821 vers 192.168.1.20 sur TCP port 80.
- L'attaque utilisant l'exploit 16822 n'a pas été détectée de 192.168.1.20 vers 172.16.2.3 sur TCP port 8080, mais a été détectée de vm-5792 vers 192.168.1.20 sur TCP port 25.
- L'attaque utilisant l'exploit 28681 n'a pas été détectée de vm-5811 vers 172.16.2.66 sur TCP port 143, mais a été détectée de vm-5804 vers 172.16.2.1 sur TCP port 21.

Ces incohérences de détection sont simplement dues au fait que les NIDS ne sont pas configurés pour surveiller toutes les *attaques non standards*.

6.3.2.3 Synthèse des résultats

Les résultats expérimentaux que nous avons obtenus sont intéressants. Ils démontrent que notre prototype implémente comme attendu les méthodes conceptuelles élaborées dans le cadre de notre approche d'évaluation et d'analyse des mécanismes de sécurité réseau dans le cloud. De plus, pour un premier prototype, les durées d'exécution sont acceptables. Par ailleurs, le contenu des rapports que nous avons pu générer témoigne de l'intérêt de notre approche. En effet, nous avons pu mettre en évidence la présence de déviances dans l'implémentation des contrôles d'accès et d'incohérences dans le déploiement des systèmes de détection d'intrusion. Ceci peut alors être pris en compte par les administrateurs sécurité en charge de gérer l'infrastructure auditée.

6.4 Conclusion

Dans ce dernier chapitre, nous avons décrit la plateforme de maquettage et d'expérimentation que nous avons mise en place dans le cadre de ces travaux de thèse. Elle a permis de valider le prototypage de notre approche d'évaluation et d'analyse des mécanismes de sécurité réseau dans des clouds IaaS. Le prototype associé a ensuite été présenté en détails d'un point de vue architectural. Enfin, nous avons exposé les expérimentations que nous avons réalisées, puis présenté les résultats obtenus.

Ces derniers permettent de valider la faisabilité des méthodes présentées au cours des chapitres précédents, et sont encourageants dans l'extension envisageable du prototype associé. Ce dernier est efficace dans l'analyse de l'application des contrôles d'accès réseau par VMware sur des infrastructures virtuelles de petite et moyenne taille. Si l'analyse statique est exécutable sur de grandes infrastructures, l'analyse dynamique requiert un travail d'optimisation pour ces cas. Il faudrait éviter de tester toutes les sessions possibles mais seulement les plus représentatives pour ainsi gagner du temps à

l'exécution dans le cas de grandes infrastructures. Concernant l'évaluation des déploiements de NIDS, nous avons démontré l'efficacité de la méthode proposée pour mettre en évidence les faiblesses de ces déploiements. Nous étant concentrés sur deux produits, nous pouvons être encore plus exhaustifs sur la diversité des solutions évaluables. De plus, le nombre de vulnérabilités embarquées dans le système d'audit reste limité, et pourrait être augmenté afin d'être plus complets dans la recherche des faiblesses de déploiements de NIDS.

Conclusion générale

Bilan

L'essor du cloud computing ces dernières années a entraîné le développement d'une multitude de services associés. Ces services concernent l'utilisation de ressources informatiques à distance : applications, plateformes de développement et d'exécution, infrastructures. Le modèle IaaS permet d'offrir aux clients du cloud des infrastructures virtuelles, généralement hébergées chez les fournisseurs de services. La dynamicité des environnements cloud, due au nombre de changements de configuration possibles par les clients et fournisseurs, peut impliquer des conséquences négatives sur la sécurité réseau du cloud. Pour se prémunir des menaces, ces infrastructures virtuelles sont protégées par des mécanismes de sécurité réseau comme les pare-feu virtuels et les systèmes de détection d'intrusion réseau. Les pare-feu ont pour rôle de gérer le contrôle d'accès réseau, tandis que les systèmes de détection d'intrusion doivent détecter les attaques survenant sur le réseau.

Dans ce contexte, nous avons proposé une approche en trois phases pour l'évaluation et l'analyse des mécanismes de sécurité réseau dans les infrastructures virtuelles de cloud computing. La première phase consiste à cloner l'infrastructure virtuelle du client dont la sécurité réseau est à évaluer, afin de mener les opérations d'audit suivantes sans pertuber la production du client. La deuxième phase concerne l'analyse des contrôles d'accès réseau sur l'infrastructure clone. Ceci est réalisé de manière statique, en analysant la configuration du cloud, et dynamique, en injectant du trafic réseau. L'objectif est de déterminer les communications possibles, ou accessibilités réseau, et de comparer les résultats obtenus par les différentes méthodes à la recherche d'éventuelles déviances dans l'établissement ou l'application de la politique de sécurité et des règles de filtrage. Enfin, la dernière phase de notre approche comprend l'exécution de campagnes d'attaque en fonction des accessibilités trouvées et du paramétrage des attaques à lancer. Des métriques d'évaluation sont calculées pour estimer la capacité des systèmes de détection d'intrusion réseau en place à protéger l'environnement cloud. La conception de cette approche a été suivie par la réalisation d'un prototype pour des clouds de type VMware. Ce prototype a permis de mener des expérimentations qui illustrent nos contributions. Les résultats obtenus sont encourageants. Ils témoignent de l'intérêt et de la faisabilité de notre approche, et ouvrent la voie à plusieurs perspectives à nos travaux.

Perspectives

Les premières pespectives importantes concernent l'extension de notre prototype à d'autres solutions de cloud IaaS et d'autres systèmes de détection d'intrusion. Il convient alors de développer les composants qui sont en intéraction directe avec les

APIs du cloud, à savoir de nouveaux cloneurs, parsers de configuration et contrôleurs. Par exemple, l'extension à la solution en source libre OpenStack [16] paraît tout à fait envisageable, compte tenu du fait qu'elle offre des fonctionnalités similaires accessibles via des APIs documentées. En outre, pour pouvoir évaluer d'autres systèmes de détection d'intrusion, il suffit d'implémenter de nouveaux parsers d'alerte pour les produits voulus.

Concernant l'évaluation des systèmes de détection d'intrusion, un effort humain, pouvant devenir communautaire, est à réaliser quant à la génération d'automates pour modéliser les comportements réseau d'applications vulnérables. Un grand nombre d'automates permet d'exécuter plus d'attaques réseau, et ainsi de couvrir plus de vulnérabilités et être plus exhaustif dans le calcul des métriques d'évaluation. On peut aussi imaginer l'élaboration de scénarios d'attaque complexes pour évaluer des NIDS à détection d'anomalies, ou les comportements observés prennent plus d'importance dans la détection des attaques que des signatures. A cela peuvent être ajoutées des techniques d'obfuscation d'attaques, afin de mesurer la capacité des systèmes à détecter des attaques masquées. Un autre travail intéressant porte sur le calcul des métriques, où il serait intéressant de vérifier quelle requête ou réponse a déclenché une alerte. Ceci permettrait de vérifier la réactivité des systèmes de détection de manière très précise, voire de définir de nouvelles métriques d'évaluation plus adaptées aux attaques actuelles, de plus en plus sophistiquées.

On peut également envisager l'adaptation du prototype à des infrastructures virtuelles de grande taille. Le module d'analyse statique des contrôles d'accès reste utilisable sur de grandes infrastructures. En revanche, les méthodes d'analyse dynamique des contrôles d'accès et d'exécution de campagnes d'attaque ont un caractère expérimental qui ne peut être extensible sans considérer une adaptation. Il peut s'agir par exemple de regrouper les machines ou réseaux similaires, afin d'éviter de mener de nombreuses injections de trafic sur des segments réseaux surveillés de manière très similaire par les pare-feu et systèmes de détection d'intrusion. On pourrait alors envisager d'ajouter une étape préliminaire de modélisation de la topologie réseau sous forme de graphe. Une abstraction de ce graphe pourrait alors être utilisée pour guider le clonage, afin d'économiser des ressources requises pour cette étape et réduire la complexité de l'infrastructure clone évaluée. Ceci permettrait de limiter la durée des phases expérimentales de l'audit.

Par ailleurs, une piste de travail importante à exploiter s'attacherait à automatiser et optimiser le déclenchement des audits. Dans l'état actuel, les fonctionnalités du système sont utilisables à la demande, via une interface Web ou un programme en ligne de commande. Un autre cas plus adapté au cloud serait l'automatisation des lancements des audits tout en optimisant leur exécution. En effet, le caractère dynamique qui caractérise le cloud est un facteur à prendre en compte pour décider quand mener des évaluations de sécurité. Le principe pouvant être adopté se rapproche de la boucle MAPE (*Monitor*, *Analyze*, *Plan*, *Execute*) issue de domaine de l'informatique autonome ou *autonomic computing*. Cela consiste à surveiller les changements

survenus sur une infrastructure, les analyser pour pouvoir planifier l'audit puis l'exécuter. Des exemples de changements à surveiller sont l'ajout d'une machine virtuelle, la connexion d'une interface réseau de machine virtuelle à un réseau, un changement dans le filtrage d'un pare-feu (suppression ou désactivation de la fonction de filtrage, ajout ou activation d'une règle autorisant du trafic, désactivation ou suppression d'une règle refusant du trafic), un changement dans le routage d'un pare-feu (ajout ou activation d'une règle), l'ajout ou la suppression d'une sonde de détection d'intrusion. L'analyse des changements pourrait être liée à trois critères qui valideraient le besoin d'un audit d'une infrastructure : la déviation, la stabilité et la disponibilité. Le critère de déviation considère que l'infrastructure a suffisamment changé pour devoir lancer un audit. Le critère de stabilité considère qu'une infrastructure a fini d'évoluer et est stable pendant une période suffisamment grande pour pouvoir lancer un audit. Le critère de disponibilité considère qu'il y a suffisamment de ressources informatiques disponibles pour effectuer le clonage de l'infrastructure nécessaire à un audit.

Bibliographie

[1] Amazon Web Services (AWS) - Services de cloud computing. http://aws.amazon.com. Consulté le 30/06/15.

[2] AUTOEXPECT (1) manual page. http://expect.sourceforge.net/example/autoexpect.man.html. Consulté le 30/06/15.

[3] Cisco Systems - Scalable Cloud Networking with Cisco Nexus 1000V Series Switches and VXLAN. http://www.cisco.com/c/en/us/products/collateral/switches/nexus-1000v-switch-vmware-vsphere/white_paper_c11-685115.html. Consulté le 30/06/15.

[4] Common Vulnerabilities and Exposures (CVE). http://cve.mitre.org. Consulté le 30/06/15.

[5] CVE Details - The ultimate security vulnerability datasource. http://www.cvedetails.com/browse-by-date.php. Consulté le 30/06/15.

[6] Description of Windows Virtual PC. https://support.microsoft.com/kb/958559. Consulté le 30/06/15.

[7] Developments of the Honeyd Virtual Honeypot. http://www.honeyd.org. Consulté le 30/06/15.

[8] Dictionnaire de l'Académie française, neuvième édition. http://atilf.atilf.fr/academie9.htm. Consulté le 30/06/15.

[9] Exploit Database by Offensive Security. http://www.exploit-db.com. Consulté le 30/06/15.

[10] Heartbleed Bug. http://heartbleed.com. Consulté le 30/06/15.

[11] Internet World Stats - World Internet Users Statistics and 2014 World Population Stats. http://www.internetworldstats.com/stats.htm. Consulté le 30/06/15.

[12] KVM. http://www.linux-kvm.org. Consulté le 30/06/15.

[13] Metasploit - Penetration Testing Software. http://www.metasploit.com/. Consulté le 30/06/15.

[14] Netmedia - Enjeux IT : Où en sont les DSI en région en 2014 ? http://www.vmwaretour2014.com/IMG/pdf/pleniere_vmware_tour_2014.pdf. Consulté le 30/06/15.

[15] Nmap - Free Security Scanner For Network Exploration & Security Audits. https://nmap.org. Consulté le 30/06/15.

[16] OpenStack - Open Source Cloud Computing Software. https://www.openstack.org. Consulté le 30/06/15.

[17] Oracle VM VirtualBox. https://www.virtualbox.org. Consulté le 30/06/15.

[18] PAC CloudIndex - Le niveau de maturité Cloud des organisations françaises a franchi un palier - Quatrième édition, Décembre 2014. `http://www.cloudindex.fr/content/tous-les-résultats`. Consulté le 30/06/15.

[19] ping(8) - Linux man page. `http://linux.die.net/man/8/ping`. Consulté le 30/06/15.

[20] pydot - Python interface to Graphviz's Dot language. `https://github.com/erocarrera/pydot`. Consulté le 30/06/15.

[21] pysphere - vSphere SDK for python - Google Project Hosting. `https://code.google.com/p/pysphere/`. Consulté le 30/06/15.

[22] python-Levenshtein 0.12.0 : Python Package Index. `https://pypi.python.org/pypi/python-Levenshtein/0.12.0`. Consulté le 30/06/15.

[23] QEMU. `http://www.qemu.org`. Consulté le 30/06/15.

[24] Scapy. `http://www.secdev.org/projects/scapy/`. Consulté le 30/06/15.

[25] SWI-Prolog. `www.swi-prolog.org`. Consulté le 30/06/15.

[26] The Emerging Threats Open Source Community. `http://www.emergingthreats.net/open-source/open-source-community`. Consulté le 30/06/15.

[27] The Xen Project, the powerful open source industry standard for virtualization. `http://www.xenproject.org`. Consulté le 30/06/15.

[28] The XSLT C library for GNOME. `http://xmlsoft.org/XSLT/`. Consulté le 30/06/15.

[29] traceroute(8) - Linux man page. `http://linux.die.net/man/8/traceroute`. Consulté le 30/06/15.

[30] vCloud Suite, vSphere-Based Private Cloud Features : VMware. `http://www.vmware.com/products/vcloud-suite`. Consulté le 30/06/15.

[31] VMware - vSphere ESXi Bare-Metal Hypervisor. `http://www.vmware.com/products/esxi-and-esx/overview`. Consulté le 30/06/15.

[32] Vue d'Ensemble d'Hyper-V. `https://technet.microsoft.com/library/hh831531.aspx`. Consulté le 30/06/15.

[33] web2py Web Framework. `http://www.web2py.com`. Consulté le 30/06/15.

[34] Wireshark. `https://www.wireshark.org`. Consulté le 30/06/15.

[35] Alert Logic Cloud Security Report - Research on the Evolving State of Cloud Security - Spring 2014, 2014.

[36] Rim Akrout : *Analyse de vulnérabilités et évaluation de systèmes de détection d'intrusions pour les applications Web.* Thèse de doctorat, INSA de Toulouse, 2012.

[37] Eric Alata : *Observation, caractérisation et modélisation de processus d'attaques sur Internet.* Thèse de doctorat, INSA de Toulouse, 2007.

[38] Dominique Alessandri : *Attack-Class-Based Analysis of Intrusion Detection Systems*. Thèse de doctorat, University of Newcastle upon Tyne, School of Computing Science, 2004.

[39] Turki Alharkan et Patrick Martin : IDSaaS : Intrusion Detection System as a Service in Public Clouds. *In Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, pages 686–687. IEEE, 2012.

[40] Cloud Security Alliance : Security Guidance for Critical Areas of Focus in Cloud Computing v3.0. 2011.

[41] Cloud Security Alliance : SecaaS Implementation Guidance : Security Assessments. 2012.

[42] Spyros Antonatos, Kostas G. Anagnostakis et Evangelos P. Markatos : Generating Realistic Workloads for Network Intrusion Detection Systems. *In ACM SIGSOFT Software Engineering Notes*, volume 29, pages 207–215. ACM, 2004.

[43] João Antunes : *Network Attack Injection*. Thèse de doctorat, Universidade de Lisboa, 2012.

[44] Jean Arlat, Alain Costes, Jean-Paul Blanquart et Jean-Claude Laprie : *Guide de la sûreté de fonctionnement*. Cépaduès-éditions, 1996.

[45] John Arrasjid : vCloud Architecture Toolkit 3.0 Overview.

[46] Aman Bakshi et B. Yogesh : Securing Cloud from DDOS Attacks Using Intrusion Detection System in Virtual Machine. *In 2nd International Conference on Communication Software and Networks. ICCSN'10*, pages 260–264. IEEE, 2010.

[47] M. Balamurugan et B. Sri Chitra Poornima : Honeypot as a Service in Cloud. *In* International Journal of Computer Applications, éditeur : *Proceedings of the 2011 International Conference on Web Services Computing*, 2011.

[48] Eric Bauer et Randee Adams : *Reliability and Availability of Cloud Computing*. John Wiley & Sons, 2012.

[49] Nihel Ben Youssef, Adel Bouhoula et Florent Jacquemard : Automatic Verification of Conformance of Firewall Configurations to Security Policies. *In Symposium on Computers and Communications. ISCC 2009*, pages 526–531. IEEE, 2009.

[50] Matt Bishop : Analysis of the ILOVEYOU Worm. 2000.

[51] Sören Bleikertz : Automated Security Analysis of Infrastructure Clouds. Master. *Norwegian University of Science and Technology*, 2010.

[52] Sören Bleikertz, Thomas Gross, Matthias Schunter et Konrad Eriksson : Automated Information Flow Analysis of Virtualized Infrastructures. *In Computer Security–ESORICS 2011*, pages 392–415. Springer, 2011.

[53] Georges Bossert, Guillaume Hiet et Thibaut Henin : Modelling to Simulate Botnet Command and Control Protocols for the Evaluation of Network Intrusion

Detection Systems. *In 2011 Conference on Network and Information Systems Security (SAR-SSI)*, pages 1–8. IEEE, 2011.

[54] Stephen Brown, Rebecca Lam, Shishir Prasad, Sivasubramanian Ramasubramanian et Josh Slauson : Honeypots in the Cloud. University of Wisconsin - Madison. 2012.

[55] Vít Bukac : IDS System Evasion Techniques. Master. *Masarykova Univerzita*, 2010.

[56] Juan Caballero, Heng Yin, Zhenkai Liang et Dawn Song : Polyglot : Automatic Extraction of Protocol Message Format using Dynamic Binary Analysis. *In Proceedings of the 14th ACM conference on Computer and communications security*, pages 317–329. ACM, 2007.

[57] Dawn M. Cappelli, Andrew P. Moore et Randall F. Trzeciak : *The CERT Guide to Insider Threats : How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)*. Addison-Wesley Professional, 2012.

[58] William R. Claycomb et Alex Nicoll : Insider Threats to Cloud Computing : Directions for New Research Challenges. *In 36th Annual Computer Software and Applications Conference (COMPSAC)*, pages 387–394. IEEE, 2012.

[59] Marc Dacier : *Vers une évaluation quantitative de la sécurité informatique*. Thèse de doctorat, Institut National Polytechnique de Toulouse, 1994.

[60] Marc Dacier, Fabien Pouget et Hervé Debar : Honeypots : Practical Means to Validate Malicious Fault Assumptions. *In Proceedings of 10th Pacific Rim International Symposium on Dependable Computing.*, pages 383–388. IEEE, 2004.

[61] Ole Martin Dahl : Using Coloured Petri Nets in Penetration Testing. *Department of Com puter Science and Media Technology. Gjøvik, Gjøvik University College. Master*, 1:89, 2005.

[62] George C. Dalton, Robert F. Mills, John M. Colombi et Richard A. Raines : Analyzing Attack Trees using Generalized Stochastic Petri Nets. *In Information Assurance Workshop*, pages 116–123. IEEE, 2006.

[63] Sajal K. Das, Krishna Kant et Nan Zhang : *Handbook on Securing Cyber-Physical Critical Infrastructure*. Accès en ligne via Elsevier, 2012.

[64] Amir Vahid Dastjerdi, Kamalrulnizam Abu Bakar et Sayed Gholam Hassan Tabatabaei : Distributed Intrusion Detection in Clouds using Mobile Agents. *In 3rd International Conference on Advanced Engineering Computing and Applications in Sciences. ADVCOMP'09.*, pages 175–180. IEEE, 2009.

[65] Hervé Debar, D. Curry et B. Feinstein : RFC4765 : The Intrusion Detection Message Exchange Format (IDMEF), 2007.

[66] Hervé Debar, Marc Dacier, Andreas Wespi et Stefan Lampart : *An Experimentation Workbench for Intrusion Detection Systems*. IBM TJ Watson Research Center, 1998.

[67] Hervé Debar et Benjamin Morin : Evaluation of the Diagnostic Capabilities of Commercial Intrusion Detection Systems. *In Recent Advances in Intrusion Detection*, pages 177–198. Springer, 2002.

[68] Corbin Del Carlo : Intrusion Detection Evasion : How Attackers Get Past the Burglar Alarm. *SANS Great Lakes, Chicago Illinois*, 2003.

[69] Yves Deswarte et Ludovic Mé : *Sécurité des réseaux et systèmes répartis*. Hermès science publications, 2003.

[70] Georg Disterer : ISO 20000 for IT. *Business & Information Systems Engineering*, 1(6):463–467, 2009.

[71] Frank Doelitzscher, Thomas Ruebsamen, Tina Karbe, Martin Knahl, Christoph Reich et Nathan Clarke : Sun Behind Clouds-On Automatic Cloud Security Audits and a Cloud Audit Policy Language. *International Journal On Advances in Networks and Services*, 6(1 and 2):1–16, 2013.

[72] Pasi Eronen et Jukka Zitting : An Expert System for Analyzing Firewall rules. *In Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, pages 100–107, 2001.

[73] Eucalyptus : Cloud Roles. `http://web.archive.org/web/20140209200258/https://www.eucalyptus.com/resources/cloud-overview/cloud-roles`. Consulté le 30/06/15.

[74] Tom Fawcett : An Introduction to ROC Analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[75] Distributed Management Task Force : Open Virtualization Format Specification. *vol. DSP0243*, 1.1.1, 2013.

[76] Mohammed El-Sayed Gadelrab : *Évaluation des Systèmes de Détection d'Intrusion*. Thèse de doctorat, Université Toulouse III-Paul Sabatier, 2008.

[77] Hatem Hamad et Mahmoud Al-Hoby : Managing Intrusion Detection as a Service in Cloud Networks. *International Journal of Computer Applications*, 41(1):35–40, 2012.

[78] Simon Hansman et Ray Hunt : A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, 2005.

[79] Dan Hubbard et Michael Sutton : Top Threats to Cloud Computing V1. 0. *Cloud Security Alliance*, 2010.

[80] Kai Hwang, Min Cai, Ying Chen et Min Qin : Hybrid Intrusion Detection with Weighted Signature Generation over Anomalous Internet Episodes. *IEEE Transactions on Dependable and Secure Computing*, 4(1):41–55, 2007.

[81] Vinay Igure et Ronald Williams : Taxonomies of Attacks and Vulnerabilities in Computer Systems. *Communications Surveys & Tutorials, IEEE*, 10(1):6–19, 2008.

[82] ISO/IEC : ISO/IEC 27017 - Information technology - Security techniques - Code of practice for information security controls based on ISO/IEC 27002 for cloud

services (DRAFT). `http://www.iso.org/iso/catalogue_detail?csnumber=43757`, 2015. Consulté le 30/06/15.

[83] Sushil Jajodia, Steven Noel et Brian O'Berry : Topological Analysis of Network Attack Vulnerability. *In Managing Cyber Threats*, pages 247–266. Springer, 2005.

[84] Wayne A. Jansen, Peter Mell, Tom Karygiannis et Don Marks : *Applying Mobile Agents to Intrusion Detection and Response*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1999.

[85] Alan Jeffrey et Taghrid Samak : Model Checking Firewall Policy Configurations. *In International Symposium on Policies for Distributed Systems and Networks. POLICY 2009*, pages 60–67. IEEE, 2009.

[86] Bernhard Kauer, Paulo Verissimo et Alysson Bessani : Recursive Virtual Machines for Advanced Security Mechanisms. *In 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 117–122. IEEE, 2011.

[87] Amir R. Khakpour et Alex X. Liu : Quarnet : A Tool for Quantifying Static Network Reachability. *Michigan State University, East Lansing, Michigan, Technical Report MSU-CSE-09-2*, 2009.

[88] Tammo Krueger, Hugo Gascon, Nicole Krämer et Konrad Rieck : Learning Stateful Models for Network Honeypots. *In Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 37–48. ACM, 2012.

[89] Gulshan Kumar : Evaluation Metrics for Intrusion Detection Systems - A Study. *Evaluation*, 2(11), 2014.

[90] Larousse : Dictionnaire français. `http://www.larousse.fr/dictionnaires/francais`. Consulté le 30/06/15.

[91] Jun-Ho Lee, Min-Woo Park, Jung-Ho Eom et Tai-Myoung Chung : Multi-level Intrusion Detection System and Log Management in Cloud Computing. *In 13th International Conference on Advanced Communication Technology (ICACT)*, pages 552–555. IEEE, 2011.

[92] Stefanie Leimeister, Markus Böhm, Christoph Riedl et Helmut Krcmar : The Business Perspective of Cloud Computing : Actors, Roles and Value Networks. *18th European Conference on Information Systems*, 2010.

[93] Corrado Leita, Ken Mermoud et Marc Dacier : Scriptgen : an Automated Script Generation Tool for Honeyd. *In 21st Annual Computer Security Applications Conference*, pages 12–pp. IEEE, 2005.

[94] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba et Kumar Das : The 1999 DARPA Off-Line Intrusion Detection Evaluation. *Computer networks*, 34(4):579–595, 2000.

[95] Richard P. Lippmann, David J. Fried, Isaac Graf, Joshua W. Haines, Kristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham et Marc A. Zissman : Evaluating Intrusion Detection Systems : The 1998 DARPA Off-Line Intrusion Detection Evaluation. *In Proceedings of DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, pages 12–26. IEEE, 2000.

[96] Chi-Chun Lo, Chun-Chieh Huang et Joy Ku : A Cooperative Intrusion Detection System Framework for Cloud Computing Networks. *In 39th International Conference on Parallel Processing Workshops (ICPPW)*, pages 280–284. IEEE, 2010.

[97] Mallik Mahalingam, D. Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T. Sridhar, Mike Bursell et Chris Wright : Virtual eXtensible Local Area Network (VXLAN) : A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. *Internet Requests for Comments*, 2014.

[98] Robert Marmorstein et Phil Kearns : A Tool for Automated iptables Firewall Analysis. *In* USENIX Association, éditeur : *Proceedings of the annual conference on USENIX Annual Technical Conference. ATEC '05*, pages 44–44, 2005.

[99] Frederic Massicotte, Francois Gagnon, Yvan Labiche, Lionel Briand et Mathieu Couture : Automatic Evaluation of Intrusion Detection Systems. *In 22nd Annual of Computer Security Applications Conference. ACSAC'06*, pages 361–370. IEEE, 2006.

[100] Frédéric Massicotte et Yvan Labiche : Specification-Based Testing of Intrusion Detection Engines using Logical Expression Testing Criteria. *In 10th International Conference on Quality Software (QSIC)*, pages 102–111. IEEE, 2010.

[101] Alain Mayer, Avishai Wool et Elisha Ziskind : Fang : a Firewall Analysis Engine. *In* IEEE Computer Society, éditeur : *Proceedings of the 2000 Symposium on Security and Privacy. SP '00*, page 177, 2000.

[102] Claudio Mazzariello, Roberto Bifulco et Roberto Canonico : Integrating a Network IDS into an Open Source Cloud Computing Environment. *In 6th International Conference on Information Assurance and Security (IAS)*, pages 265–270. IEEE, 2010.

[103] James P McDermott : Attack Net Penetration Testing. *In Proceedings of the 2000 workshop on New security paradigms*, pages 15–21. ACM, 2001.

[104] John McHugh : Testing Intrusion Detection Systems : a Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM transactions on Information and system Security*, 3(4):262–294, 2000.

[105] Ludovic Mé : Sécurité des systèmes d'information. *Hermès*, 2006.

[106] Peter Mell et Timothy Grance : The NIST definition of cloud computing. 2011.

[107] Darren Mutz, Giovanni Vigna et Richard Kemmerer : An Experience Developing an IDS Stimulator for the Black-Box Testing of Network Intrusion Detection Systems. *In Proceedings of the 19th Annual Computer Security Applications Conference*, pages 374–383. IEEE, 2003.

[108] G.V. Nadiammai et M. Hemalatha : Handling Intrusion Detection System using Snort Based Statistical Algorithm and Semi-supervised Approach. *Research Journal of Applied Sciences, Engineering and Technology*, 6(16):2914–2922, 2013.

[109] Timothy Nelson, Christopher Barratt, Daniel J Dougherty, Kathi Fisler et Shriram Krishnamurthi : The Margrave Tool for Firewall Analysis. *In USENIX Large Installation System Administration Conference*, 2010.

[110] James Newsome, David Brumley, Jason Franklin et Dawn Song : Replayer : Automatic Protocol Replay by Binary Analysis. *In Proceedings of the 13th ACM conference on Computer and communications security*, pages 311–321. ACM, 2006.

[111] Vincent Nicomette, Mohamed Kaâniche, Eric Alata et Matthieu Herrb : Set-Up and Deployment of a High-Interaction Honeypot : Experiment and Lessons Learned. *Journal in Computer Virology*, 7(2):143–157, 2011.

[112] NIST : Cloud Computing Taxonomy, Preliminary Draft, 2011.

[113] S.R. Nithin Chandra et T.M. Madhuri : Cloud Security using Honeypot Systems. *International Journal of Scientific & Engineering Research*, 3:1–6, 2012.

[114] Steven Noel, Matthew Elder, Sushil Jajodia, Pramod Kalapa, Scott O'Hare et Kenneth Prole : Advances in Topological Vulnerability Analysis. *In Conference For Homeland Security. CATCH'09. Cybersecurity Applications & Technology*, pages 124–129. IEEE, 2009.

[115] Steven Noel, Sushil Jajodia, Lingyu Wang et Anoop Singhal : Measuring Security Risk of Networks using Attack Graphs. *International Journal of Next-Generation Computing*, 1(1):135–147, 2010.

[116] Rodolphe Ortalo : *Evaluation quantitative de la sécurité des systèmes d'information*. Thèse de doctorat, Institut National Polytechnique de Toulouse, 1998.

[117] Rodolphe Ortalo, Yves Deswarte et Mohamed Kaâniche : Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security. *IEEE Transactions on Software Engineering*, 25(5):633–650, 1999.

[118] Cynthia Phillips et Laura P. Swiler : A Graph-Based System for Network-Vulnerability Analysis. *In Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM, 1998.

[119] Gerald J. Popek et Robert P. Goldberg : Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[120] David Powell et Robert Stroud : Conceptual Model and Architecture of MAFTIA. *Technical Report Series-University of Newcastle Upon Tyne Computing Science*, 2003.

[121] Thibaut Probst, Eric Alata, Mohamed Kaâniche et Vincent Nicomette : An Approach for the Automated Analysis of Network Access Controls in Cloud Computing Infrastructures. *In 8th International Conference on Network and System Security. NSS*, pages 1–14. Springer, 2014.

[122] Thibaut Probst, Eric Alata, Mohamed Kaâniche et Vincent Nicomette : Automated Evaluation of Network Intrusion Detection in IaaS Clouds. *In 11th European Dependable Computing Conference. EDCC*. 2015.

[123] Nicholas J. Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee et Ronald A. Olsson : A Methodology for Testing Intrusion Detection Systems. *IEEE Transactions on Software Engineering*, 22(10):719–729, 1996.

[124] Sanjay Ram : Effective Analysis of Cloud Based Intrusion Detection System. *International Journal of Computer Applications & Information Technology. IJCAIT*, 1(2):16–22, 2012.

[125] Thomas Ristenpart, Eran Tromer, Hovav Shacham et Stefan Savage : Hey, you, get off of my cloud : exploring information leakage in third-party compute clouds. *In Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[126] Sebastian Roschke, Feng Cheng et Christoph Meinel : Intrusion Detection in the Cloud. *In 8th International Conference on Dependable, Autonomic and Secure Computing. DASC'09.*, pages 729–734. IEEE, 2009.

[127] Shai Rubin, Somesh Jha et Barton P. Miller : Automatic Generation and Analysis of NIDS Attacks. *In 20th Annual Computer Security Applications Conference*, pages 28–38. IEEE, 2004.

[128] Jyotiprakash Sahoo, Subasish Mohapatra et Radha Lath : Virtualization : A Survey on Concepts, Taxonomy and Associated Security Issues. *In Proceedings of the 2010 Second International Conference on Computer and Network Technology*, ICCNT '10, pages 222–226, Washington, DC, USA, 2010. IEEE Computer Society.

[129] Karen Scarfone, Murugiah Souppaya, Amanda Cody et Angela Orebaugh : Technical Guide to Information Security Testing and Assessment. *NIST Special Publication*, 800:115, 2008.

[130] Bruce Schneier : Attack Trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[131] Peter Schoo, Volker Fusenig, Victor Souza, Márcio Melo, Paul Murray, Hervé Debar, Houssem Medhioub et Djamal Zeghlache : Challenges for Cloud Networking Security. *In Mobile Networks and Management*, pages 298–313. Springer, 2011.

[132] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann et Jeannette M. Wing : Automated Generation and Analysis of Attack Graphs. *In Proceedings of IEEE Symposium on Security and Privacy*, pages 273–284. IEEE, 2002.

[133] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann et Jeannette M Wing : Practical Attack Graph Generation for Network Aefense. *In 22nd Annual Computer Security Applications Conference. ACSAC'06*, pages 121–130. IEEE, 2006.

[134] Oleg Sheyner et Jeannette Wing : Tools for Generating and Analyzing Attack Graphs. *In Formal methods for components and objects*, pages 344–371. Springer, 2004.

[135] Richard Sizer : Information Technology Security Evaluation Criteria. *Computer Bulletin (London, 1986)*, 5, 1993.

[136] Joel Sommers, Vinod Yegneswaran et Paul Barford : A Framework for Malicious Workload Generation. *In Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 82–87. ACM, 2004.

[137] Jan Steffan et Markus Schumacher : Collaborative Attack Modeling. *In Proceedings of the 2002 ACM symposium on Applied computing*, pages 253–259. ACM, 2002.

[138] Ivan Studnia, Eric Alata, Yves Deswarte, Mohamed Kaâniche, Vincent Nicomette *et al.* : Survey of Security Problems in Cloud Computing Virtual Machines. *Proceedings of Computer and Electronics Security Applications Rendez-vous (C&ESAR 2012)*, pages 61–74, 2012.

[139] Laura P. Swiler, Cynthia Phillips, David Ellis et Stefan Chakerian : Computer-Attack Graph Generation Tool. *In Proceedings of DARPA Information Survivability Conference & Exposition II. DISCEX'01*, volume 2, pages 307–321. IEEE, 2001.

[140] Jacob W Ulvila et John E Gaffney : Evaluation of Intrusion Detection Systems. *Journal of Research - National Institute of Standards and Technology*, 108(6):453–474, 2003.

[141] Geraldine Vache-Marconato : *Evaluation quantitative de la sécurité informatique : approche par les vulnérabilités*. Thèse de doctorat, INSA de Toulouse, 2009.

[142] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres et Maik Lindner : A Break in the Clouds : Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[143] Pavan Verma et Atul Prakash : FACE : A Firewall Analysis and Configuration Engine. *In Proceedings of the 2005 Symposium on Applications and the Internet*, pages 74–81. IEEE, 2005.

[144] Kleber Vieira, Alexandre Schulter, Carlos Becker Westphall et Carla Merkle Westphall : Intrusion Detection for Grid and Cloud Computing. *IT Professional*, 12(4):38–43, 2010.

[145] Giovanni Vigna, William Robertson et Davide Balzarotti : Testing Network-Based Intrusion Detection Signatures using Mutant Exploits. *In Proceedings of the 11th ACM conference on Computer and communications security*, pages 21–30. ACM, 2004.

[146] Ping Wang, Wen-Hui Lin, Pu-Tsun Kuo, Hui-Tang Lin et Tzu Chia Wang : Threat Risk Analysis for Cloud Security Based on Attack-Defense Trees. *In 8th International Conference on Computing Technology and Information Management (ICCM)*, volume 1, pages 106–111. IEEE, 2012.

[147] Ruoyu Wu, Weiguo Li et He Huang : An Attack Modeling Based on Hierarchical Colored Petri Nets. *In International Conference on Computer and Electrical Engineering. ICCEE 2008*, pages 918–921. IEEE, 2008.

[148] Geoffrey G Xie, Jibin Zhan, David A Maltz, Hui Zhang, Albert Greenberg, Gisli Hjalmtysson et Jennifer Rexford : On Static Reachability Analysis of IP Networks. *In Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM 2005*, volume 3, pages 2170–2183. IEEE, 2005.

[149] Lihua Yuan, Hao Chen, Jianning Mai, Chen-Nee Chuah, Zhendong Su et Prasant Mohapatra : Fireman : A Toolkit for Firewall Modeling and Analysis. *In 2006 Symposium onSecurity and Privacy*, pages 15–pp. IEEE, 2006.

[150] Yinqian Zhang, Ari Juels, Michael K. Reiter et Thomas Ristenpart : Cross-VM Side Channels and Their Use to Extract Private Keys. *In Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316. ACM, 2012.

Résumé

Ces dernières années, le développement d'Internet a contribué à l'essor du modèle cloud computing, dans lesquels des fournisseurs mettent à disposition des clients des ressources informatiques en tant que services. Ces ressources, généralement hébergées chez le fournisseur, peuvent être des infrastructures informatiques, des plateformes de développement et d'exécution ou des applications. L'objectif est de favoriser la réduction des coûts de déploiement et d'opération de ressources traditionnellement hébergées dans les locaux des clients. Dans le modèle de service *Infrastructure as a Service*, les clients peuvent créer et administrer des infrastructures virtuelles entières hébergeant tout ou une partie de leur système d'information. Aux bénéfices du modèle cloud sont associés des problématiques de sécurité, comme dans tout système informatique réparti. La diversité des acteurs mêlée à la variété des technologies dans le cloud implique un grand nombre de menaces et rend la sécurisation des données complexe. Pour prévenir et détecter les attaques, des mécanismes de sécurité réseau sont déployés dans le cloud. Nous nous intéressons au contrôle d'accès réseau et à la détection d'intrusion réseau, respectivement assurés par les pare-feu et les systèmes de détection d'intrusion. Or, il n'est pas aisé pour les administrateurs de déployer correctement ces outils de sécurité sans perturber le fonctionnement du cloud. Il est donc essentiel de rechercher régulièrement les faiblesses, déviances ou incohérences dans le déploiement de ces outils.

Dans ce manuscrit, nous décrivons les travaux de thèse où nous avons proposé une approche pour l'évaluation et l'analyse automatisée des mécanismes de sécurité réseau dans les infrastructures virtuelles de cloud computing. Notre objectif est de permettre l'audit de manière expérimentale des contrôles d'accès réseau et des systèmes de détection d'intrusion réseau protégeant une infrastructure virtuelle donnée. Afin de solutionner les problèmes liés à la mise en œuvre d'une telle approche, nous l'avons décomposée en trois phases. La première phase consiste à créer une copie de l'infrastructure à analyser, de manière à ne pas perturber la production du client durant les opérations d'audit. La deuxième phase concerne l'analyse des contrôles d'accès, où le but est de déterminer les canaux de communications réseau entre machines virtuelles. Nous permettons de la réaliser statiquement, à partir des informations des configurations, et dynamiquement, en injectant du trafic réseau. L'intérêt de pouvoir mener deux analyses différentes est d'identifier d'éventuelles déviances dans les résultats obtenus. Dans la troisième phase, les canaux de communications trouvés sont utilisés pour exécuter des campagnes d'attaque réseau avec du trafic d'évaluation rejoué à partir de modèles que nous avons définis. La réaction des systèmes de détection d'intrusion est alors étudiée pour générer des métriques d'évaluation. L'approche développée a donné lieu à un prototype pour les solutions cloud VMware. Ce prototype, testé sur une plateforme de maquettage et d'expérimentation, a permis de valider les méthodes conçues dans le cadre de l'approche. Les résultats expérimentaux obtenus sont encourageants et donnent confiance dans l'élaboration de nouvelles extensions et perspectives de recherche.

Abstract

Over the last few years, the development of the Internet contributed to the rise of the cloud computing model, wherein providers offer computing resources as services to clients. These resources, generally hosted by the provider, can be infrastructures, development and execution platforms or applications. The goal is to boost the reduction of the deployment and operation costs of resources traditionally hosted on-premises. In the Infrastructure as a Service (IaaS), clients can create and administrate entire virtual infrastructures hosting their information system or a part of it. Beside the benefits of the cloud model, security concerns arise, as in any distributed computing system. Mixing the diversity of the actors with the variety of technologies in the cloud implies a great number of threats and makes the securing of data more complex. In order to prevent and detect attacks, network security mechanisms are deployed in the cloud. We are interested in network access control and network intrusion detection, respectively carried out by firewalls and intrusion detection systems. It is not yet easy for administrators to correctly deploy security tools while not disturbing the cloud. Therefore, it is essential to look for weaknesses, discrepancies or inconsistencies in their deployment on a regular basis.

In this manuscript, we describe the thesis in which we propose an approach for the automated evaluation and analysis of network security mechanisms in cloud computing virtual infrastructures. Our objective is to allow, in an experimental fashion, the audit of network access controls and network intrusion detection systems protecting virtual infrastructures. To work around the problems due to the implementation of such an approach, we divided it in three phases. The first phase consists in creating a copy of the infrastructure to analyze, to avoid disturbing the client's business during the audit operations. The second phase is about the analysis of access controls, where the goal is to determine network communication paths between the virtual machines. We allow a static analysis, conducted from configuration information, and a dynamic analysis, performed by injecting network traffic. The interest in achieving two different types of analysis is to identify potential discrepancies in the results. In the third phase, the discovered communication paths are utilized to execute network attack campaigns based on evaluation traffic we replay using models we defined. Then, the reaction of intrusion detection systems is studied to generate evaluation metrics. The developed approach resulted in a prototype for VMware cloud solutions. It has been experimented on a mock-up platform in order to validate the methods we designed as part of our approach. The experimental results we obtained are encouraging and build confidence in the elaboration of new extensions and research perspectives.