Vers une détection à la source des activités malveillantes dans les clouds publics : application aux attaques de déni de service

Badis Hammi

Thèse
de doctorat
de l'UTT

Badis HAMMI

Vers une détection à la source des activités malveillantes dans les *clouds* publics : application aux attaques de déni de service

Spécialité :
Réseaux, Connaissances et Organisations

2015TROY0023

Année 2015

utt
université de technologie
Troyes

THESE

pour l'obtention du grade de

DOCTEUR de l'UNIVERSITE DE TECHNOLOGIE DE TROYES

Spécialité : RESEAUX, CONNAISSANCES, ORGANISATIONS

présentée et soutenue par

Badis HAMMI

le 29 septembre 2015

Vers une détection à la source des activités malveillantes dans les clouds publics : application aux attaques de déni de service

JURY

M. B. COUSIN	PROFESSEUR DES UNIVERSITES	Président
Mme I. CHRISMENT	PROFESSEUR DES UNIVERSITES	Rapporteur
M. H. DEBAR	PROFESSEUR TELECOM SUDPARIS	Rapporteur
M. G. DOYEN	MAITRE DE CONFERENCES	Directeur de thèse
M. R. KHATOUN	MAITRE DE CONFERENCES	Directeur de thèse
M. W. MALLOULI	INGENIEUR DE RECHERCHE, DOCTEUR	Examinateur

Mis en page avec la classe thesul.

Remerciements

C'est mon quatrième printemps à l'université technologique de Troyes. Lieu où l'angoisse des examens, les efforts fournis, le rire, la déception, la joie, la colère et enfin la réussite se côtoient au service du savoir. Lequel m'a été acheminé avec conscience, responsabilité, compétence et amour durant ce parcours passionnant, par d'éminents professeurs qui croient au progrès scientifique et humain.

Merci pour leur confiance, merci pour leur persévérance, merci pour leur disponibilité. Merci de m'avoir supporté et orienté dans le bon sens, avec beaucoup de professionnalisme et patience mes recherches. Et tout cela dans un seul souci, qu'est l'éclairage de ma lanterne.

En somme « rien de grand ne se fait sans de grands hommes. »

Mes premières pensées vont d'abord vers mes deux directeurs de thèse : messieurs Guillaume Doyen et Rida Khatoun pour la confiance qu'ils m'ont accordée ainsi que pour les qualités scientifiques et pédagogiques de leur encadrement. Je les remercie pour leur disponibilité, pour leur patience, pour leur pertinence et pour leurs précieux conseils, Ils ont fait montre d'un grand professionnalisme et d'une impressionnante compétence.

Je remercie Monsieur Bernard Cousin, Professeur des universités à l'école supérieure d'ingénieurs de Rennes, qui m'a fait l'immense honneur de présider mon jury de thèse.

Je remercie Madame Isabelle Chrisment, Professeur des universités à Télécom Nancy et Monsieur Hervé Debar, professeur des universités à Télécom SudParis, qui m'ont honoré d'accepter de rapporter cette thèse. Mes remerciements vont également à Monsieur Wissam Mallouli, Ingénieur de recherche à Montimage, qui m'a fait l'honneur à son tour, de bien vouloir juger ma thèse.

Avec toute mon affection et ma gratitude, je remercie profondément ma chère future femme Lilia, mes chers parents Ali et Beya, mon frère Mohamed Tahar et ma sœur Feyrouz qui ont fait preuve de bravoure et d'abnégation à mon égard.

Je ne remercierai jamais assez mon mentor, mon grand frère et cher ami Jean Philippe Vilotte pour son soutien indéfectible et qui n'a jamais épargné aucun effort pour que je puisse me frayer un chemin, non sans embûches, vers la réussite.

Je suis profondément reconnaissant envers mon cher frère et ami Mohamed Cherif Rahal pour tous ses conseils et son soutien qui m'ont été plus que bénéfiques et salutaires.

Je tiens également à remercier les membres de l'équipe ERA et plus spécifiquement mes collègues de bureau pour leur accueil, leur sympathie, surtout pour les précieux moments de partage.

Finalement, il m'est particulièrement agréable d'exprimer ma reconnaissance envers tous ceux qui ont contribué, de près ou de loin à la réalisation de mon modeste travail.

Sommaire

Annexes

Annexe A
Résultats complets de la campagne de caractérisation du comportement système d'un *botcloud*

Table des figures

Liste des tableaux

Introduction générale

Contexte

Le *cloud computing* est aujourd'hui une solution largement adoptée pour la production de services IT, et il est devenu désormais un acteur indispensable au fonctionnement des différentes infrastructures. Pour l'année 2014, Eurostat[1] [1], a produit les résultats d'une étude sur l'usage de la technologie du *cloud* dans les entreprises européennes. La figure 1.a représente les pourcentages d'utilisation des services *cloud* par pays et la figure 1.b répartit ces pourcentages par domaine d'application. On y voit qu'en moyenne 19% des entreprises de l'Europe des 28 utilisent des services *cloud* avec un taux d'usage supérieur à 40% pour les trois pays qui y ont le plus recours et la présence de 12 pays qui ont un usage supérieur à cette moyenne. On voit aussi que les domaines industriels des entreprises qui ont recours à cette solution sont très variés, et pas seulement réduis aux entreprises dont l'activité se concentre sur les technologies de l'information et la communication. L'usage du *cloud* est donc maintenant acquis et avéré et on s'attend à une augmentation de ce taux d'adoption dans les années à venir.

Le *cloud* représente un modèle informatique, où des capacités de la technologie d'information, massivement évolutives, sont livrées *en tant que service* et *à la demande* pour une clientèle externe, en utilisant les technologies d'Internet. Ainsi, il offre de nombreux avantages, tel qu'un déploiement rapide des services, la réduction des coûts pour son utilisateur, une facturation à la demande et l'évolutivité des infrastructures support.

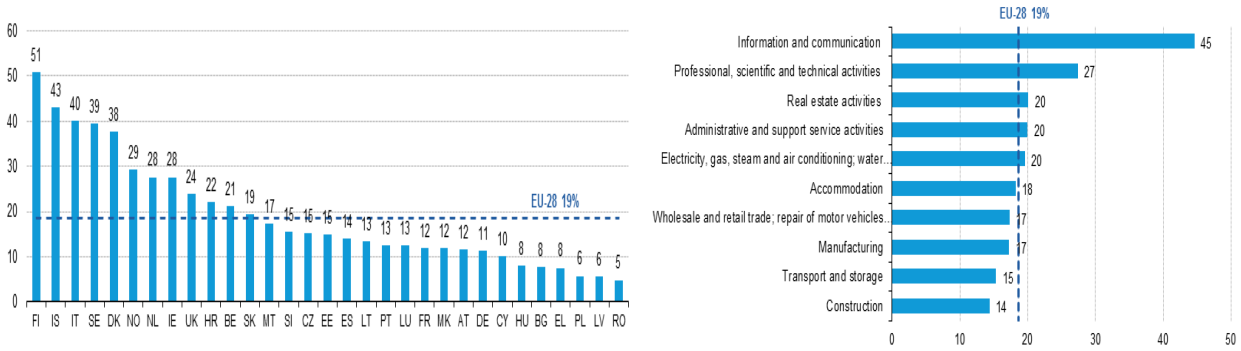
Afin d'illustrer l'efficacité et les bénéfices de l'utilisation des services d'un fournisseur *cloud*, nous citons l'exemple du quotidien New York Times. En 2008, dans le but de créer la *New York Times – TimesMachine*, les dirigeants du journal ont utilisé les services *cloud* d'Amazon pour exporter en PDF, leurs anciens numéros numérisés remontant à 150 ans. D'après leurs propres estimations, il aurait fallu 14 ans pour mener à bien ce projet avec la mise en service de leurs propres serveurs. Utilisant les services d'Amazon, ils ont accompli cette tâche en 36 heures pour la somme de 240 dollars[2]. Cet exemple montre ainsi la puissance offerte par le *cloud* en terme d'infrastructure mise à disposition, mais aussi sa facilité d'utilisation pour des usages très variés.

Problématique

En dépit des avantages offerts par le *cloud* pour les utilisateurs légitimes, des utilisateurs malveillants tirent parti de ses caractéristiques et avantages pour bénéficier d'une

[1]Eurostat est l'office statistique de l'union européenne. Il est chargé de fournir à cette dernière des statistiques au niveau européen permettant des comparaisons entre les pays et les régions.

[2]open.blogs.nytimes.com/2008/05/21/the-new-york-times-archives-amazon-web-services-



(a) (b)

FIGURE 1 – Statistiques sur l'utilisation du *cloud* en Europe [1] ; (a) pourcentage d'utilisation par pays ; (b) pourcentage d'utilisation par domaine

plate-forme d'attaque prête à l'emploi et dotée d'une puissance colossale, afin de déployer facilement leurs attaques envers des tiers connectés à l'Internet. Parmi les plus grands bénéficiaires de cette conversion en vecteur d'attaque, les *botnets* appelés dans ce contexte *botclouds*, sont utilisés pour perpétrer des attaques de déni de service distribuées (DDoS). Au delà des modèles de services légitimes du *cloud*, on parle alors de *DDoS as a Service*.

Une étude [8] ayant pour objectif d'étudier les potentiels avantages d'une infrastructure de services *cloud* pour un tiers malveillant a été menée. L'expérimentation a consisté en la souscription d'une offre IaaS (*Infrastructure as a Service*) auprès de cinq fournisseurs de services *cloud* majoritaires du marché actuel afin d'exploiter leurs services et produire différents types d'attaques contre une tierce partie et notamment des attaques de déni de service. Les expérimentations ont duré 21 jours. Afin de déterminer si la durée de l'attaque a un impact sur sa détection, les attaques DDoS ont duré 48 heures sans arrêt. Cependant, aucune réaction n'a été enregistrée de la part d'aucun des fournisseurs de services. Dans la même idée, une étude [9] a montré comment le *cloud* pourrait être exploité pour créer un large *botcloud*, en quelques minutes en usant d'un minimum d'effort et en passant par l'exploitation des services d'Amazon EC2 afin de réaliser des attaques DDoS et de fraude aux cliques. Le succès de ces expérimentations démontre que la simplicité, la robustesse, la flexibilité et le faible coût des services *cloud*, représente une solution adéquate à qui souhaite héberger des services malveillants.

Si les attaques DDoS, perpétrées par des *botnets* ont été largement étudiées par le passé, leur mode opératoire et leur contexte de mise en œuvre est ici différent et nécessite de nouvelles solutions. En effet, du côté du fournisseur de services *cloud*, les attaques hébergées peuvent être d'un débit conséquent et porter atteinte à son infrastructure, bien qu'il n'en soit pas la cible, le rendant victime de dommages collatéraux. Ensuite, et à l'inverse de la situation précédente, un attaquant peut réduire au minimum son débit d'attaque par machine infectée pour dissimuler son attaque dans une charge légitime à très grande échelle qui forme la majeure partie de l'activité d'un *cloud*. Enfin, pour un opérateur de *cloud*, être identifié comme source de ce type d'attaque peut aussi lui

timesmachine/

être préjudiciable économiquement. Détecter et bloquer ces attaques quelle que soit leur intensité et au plus proche de leur source est donc un enjeu important. Toutefois, dans ce contexte opérationnel, mettre en oeuvre une telle solution de détection présente plusieurs défis qui sont ceux auxquels notre travail de recherche fait face :

1. Généricité de l'approche de détection : sachant que dans le contexte d'un fournisseur de services de *cloud* public, des *botclouds* ou toute autre forme de logiciel malveillant peuvent héberger tout un ensemble d'activités malveillantes qui va au delà de la seule attaque par déni de service, une approche de détection à la source doit être suffisamment générique et extensible pour permettre la détection de toutes ces formes d'activités.
2. Volumétrie et vélocité des données : en raison de la volumétrie et de la vélocité des données générées par les sondes de surveillance, dont la grandeur est liée au nombre de machines virtuelles hébergées dans une infrastructure de *cloud computing*, il est impératif de considérer une approche efficace mais peu couteuse en terme de ressources et de temps de calcul qui puisse supporter le facteur d'échelle.
3. Confidentialité et vie privée : Effectuer une surveillance à l'aide de sondes implantées au niveau des machines virtuelles engendre des problèmes d'ordre juridique [10] relatif à la garantie de la vie privée des utilisateurs des infrastructures *cloud*. Par conséquent, la solution proposée doit prendre en compte cet aspect en reposant sur une approche non-intrusive qui ne sollicite que les données mises à la disposition d'un fournisseur de *cloud* telles que celles qui sont rendues disponibles au niveau de la couche de virtualisation.

La conception d'une approche de détection pour ce type d'attaque représente un véritable défi car elle induit la détection d'attaques, souvent à faible empreinte, dans une immense charge. Afin d'illustrer ces défis, nous présenterons à titre d'exemple sur la figure 2 les traces d'activité système sur trente minutes de deux machines virtuelles qui réalisent deux activités différentes. L'une est légitime et l'autre est malicieuse. Sur cette base, la question de recherche que nous posons est :

Sur la base des informations fournies par une couche de virtualisation, est-il possible de concevoir, mettre en oeuvre et valider une solution de détection capable de discriminer un ensemble d'activités malveillantes dans un contexte d'activités légitimes à grande échelle ?

Contributions

Nous proposons dans ce travail de thèse de répondre à cette question en élaborant une approche collaborative pour la détection d'attaques DDoS perpétrées par des machines virtuelles dans le contexte d'un opérateur public de *cloud*. Notre solution satisfait aux contraintes de respect de la vie privée et supprime tout dommage collatéral en se plaçant au niveau de la couche de virtualisation de l'infrastructure *cloud* et en ne reposant que sur les seules données systèmes connues à ce niveau. Les machines virtuelles sont ainsi perçues comme des boîtes noires. On note aussi que notre approche se détache ainsi de l'état de l'art qui repose essentiellement sur des approches fondées sur des métriques réseau pour la détection de ce type d'attaques. Par ailleurs, pour faire face à la volumétrie et la vélocité des données générées par un nombre de machines virtuelles qui peut atteindre le million

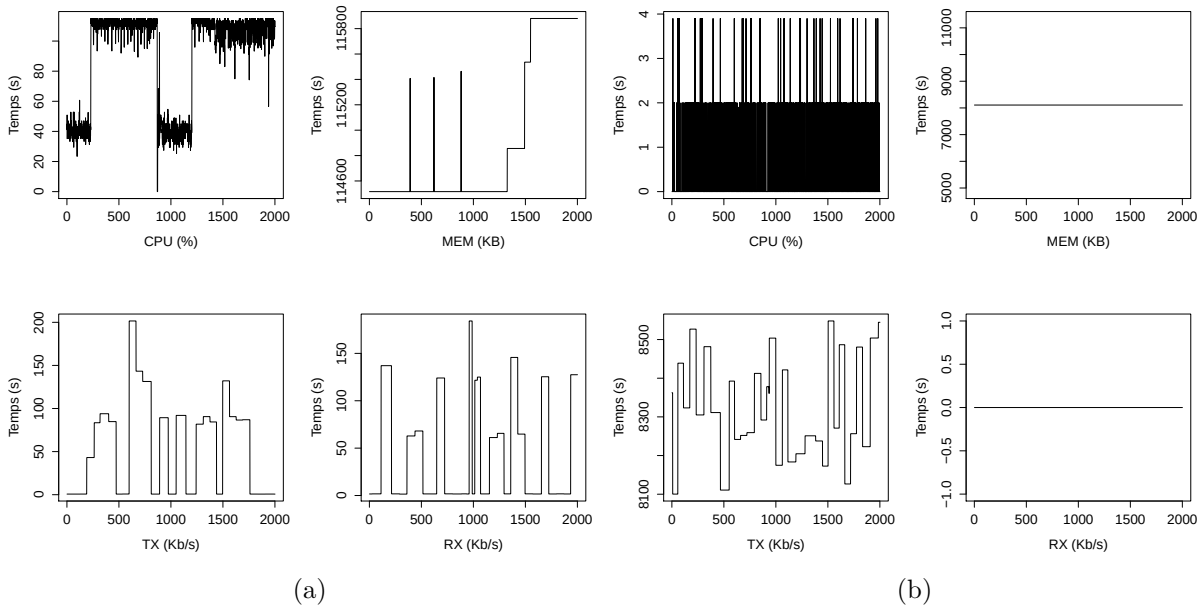


FIGURE 2 – Traces de l'activité système de deux machines virtuelles pendant 30 minutes ; (a) activité légitime ; (b) activité malicieuse

d'instances, nous proposons de limiter les données surveillées à nombre restreint et de distribuer notre solution.

Dans le but de pouvoir concevoir un tel système de détection, la démarche méthodologique que nous adoptons est la suivante. La première étape de notre travail consiste en la caractérisation comportementale de l'activité système d'un *botcloud*. Pour ce faire, nous avons conduit une large campagne expérimentale qui a consisté à mettre en œuvre *in situ* deux *botnets* différents, à savoir Hybrid_V1.0 et Kaiten, perpetrant des attaques DDoS de types UDP Flood et TCP SYN Flood, dans un contexte qui reproduise au plus proche celles d'un environnement réel de *cloud* public hébergeant une activité légitime hétérogène à grande échelle. La collecte de 31GB de fichiers journaux issus de cette campagne expérimentale ainsi que leur analyse permet de déduire des invariants comportementaux qui forment le socle d'un système de détection à base de signature. Nous utilisons ainsi ces informations pour construire un système de détection d'intrusion (IDS), fondé sur une méthode d'identification de composantes principales des *botclouds* déployés. Nous validons la performance empirique de ce détecteur en l'appliquant au jeu de données collecté et en utilisant les indicateurs standard de performance de classification. Pour satisfaire au support du facteur d'échelle, nous proposons enfin une solution de distribution de notre détecteur sur la base d'un réseau de recouvrement pair à pair structuré qui forme une architecture hiérarchique d'agrégation décentralisée. Nous montrons comment des détecteurs locaux peuvent opérer sur des sous-ensembles de machines virtuelles et comment les résultats obtenus peuvent être combinés, pour détecter des attaques DDoS.

Organisation du manuscrit

Le manuscrit est organisé en deux parties. La première présente un état de l'art sur le *cloud computing*, la problèmatique des *botnets*, des attaques DDoS et les solutions proposées pour chacun. La seconde partie décrit nos différentes contributions.

Partie I : Etat de l'art

Chapitre 1 : Concepts, modèles et technologies du *cloud* et de sa sécurité

Ce premier chapitre d'état de l'art présente le paradigme du *cloud computing* ; les types de service, les types de déploiement ainsi que la description de la technologie *open-source* de *cloud* la plus utilisée, à savoir OpenStack. La deuxième partie de ce chapitre décrit les problématiques de sécurité dans le *cloud*. Nous décrivons les vulnérabilités majeures de ce type d'infrastructure en fonction du modèle de déploiement et d'approvisionnement. Finalement nous abordons la problématique qui est au coeur de ce travail de recherche et qui porte sur l'utilisation malicieuse du *cloud*.

Chapitre 2 : *Botnets*, attaques DDoS et approches de détection

La première partie de ce chapitre présente un état de l'art sur les *botnets* et les attaques DDoS. Nous abordons en détails leurs architectures, leurs modes opératoires ainsi que leurs impacts. La deuxième partie est dédiée aux solutions de défense contre les attaques DDoS à savoir les IDS que nous présentons et classifions. Ensuite, nous nous concentrons sur les IDS collaboratifs et leurs différentes architectures. Pour chacun de ces points, nous détaillons quelques approches de détection caractéristiques.

Partie II : Contributions

Chapitre 3 : Caractérisation système des *botclouds* : Application aux DDoS

Dans ce premier chapitre de contribution, nous décrivons en détail notre contexte de travail. Par la suite, nous introduisons la campagne expérimentale que nous avons conduite, l'infrastructure utilisée ainsi que les scénarios reproduits. Ensuite, nous détaillons l'Analyse en Composantes Principales (ACP), la méthode statistique utilisée dans l'étude menée. Finalement, à l'aide de cette méthode, nous produisons une caractérisation d'un *botcloud* réalisant des attaques DDoS de types UDP Flood et TCP SYN Flood.

Chapitre 4 : Vers une approche de détection à la source des *botclouds*

Dans la première partie de ce chapitre, nous nous concentrons sur la production d'une signature pour les attaques générées par un *botcloud*. Ensuite, nous montrons comment intégrer cette signature dans le cadre de l'algorithme de détection que nous proposons. Nous produisons ensuite une évaluation de la performance empirique de cette approche de détection en l'appliquant au jeu de données collecté et en utilisant les indicateurs standard de performance de classification.

Chapitre 5 : Une solution de distribution pour l'approche proposée

Dans ce dernier chapitre de contribution, nous présentons une solution de distribution de l'approche centralisée, décrite dans le chapitre précédent. Nous proposons une approche fondée sur un réseau de recouvrement pair à pair structuré qui forme une architecture hiérarchique d'agrégation décentralisée. Nous présentons en détails cette approche ainsi qu'une évaluation de ses performances en l'appliquant au jeu de données collecté et en utilisant les mêmes indicateurs de performance que ceux utilisés dans le chapitre précédent.

Première partie

Etat de l'art

1

Concepts, modèles et technologies du *cloud* et de sa sécurité

Sommaire

1.1 Introduction

Dans ce premier chapitre d'état de l'art, nous présenterons les concepts, modèles et technologies qui forment le paradigme du *cloud computing* et de sa sécurité. Pour ce faire, dans une première partie, nous dressons un panorama complet de cette technologie en étudiant les différents services offerts par le *cloud* ainsi que ses différents modèles de déploiement. Nous étudierons ensuite les technologies clés qui forment le coeur de ces infrastructures et plus spécifiquement la virtualisation. Nous détaillons, enfin les principales plates-formes de gestion d'infrastructure *cloud* fondées sur un principe de distribution libre, en mettant l'accent sur OpenStack, la solution actuelle reconnue comme la plus complète et la plus adoptée dans ce domaine. La deuxième partie de ce chapitre est consacrée aux problématiques de sécurité dans le *cloud* que nous abordons sous deux angles. Le

premier angle est celui des attaques et intrusions qui ciblent le *cloud*, présentant ainsi le *cloud* comme une victime que les solutions de sécurité visent à protéger. Le second angle considère les usages détournés qui tirent profit des caractéristiques du *cloud*, transformant ces infrastructures en un support et vecteur pour la mise en oeuvre d'attaques.

1.2 Définitions, principes et modèles

1.2.1 Définition du *cloud computing*

Comme tout paradigme émergeant, le *cloud computing* s'est vu proposé des définitions qui sont le fruit de travaux académiques [11][12] ou normatifs [13][14]. Nous rappelons ici celles qui sont les plus reconnues et sur cette base, nous en proposons une qui en est la synthèse. Le National Institute of Standards and Technology (NIST) [13] définit le *cloud computing* comme suit :

Definition 1 *"Cloud computing is a model for enabling convenient, on demand network access to a shared pool of configurable and reliable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal consumer management effort or service provider interaction."*

Dans leur travail de caractérisation du *cloud*, Armbrust *et al.* [11] le définissent ainsi :

Definition 2 *"Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS), so we use that term. The datacenter hardware and software is what we will call a cloud."*

Enfin, dans le contexte de la sécurité, Subashini *et al.* [12], définissent le *cloud* comme :

Definition 3 *"A style of computing where massively scalable IT-enabled capabilities are delivered 'as a service' to external customers using Internet technologies."*

Sur la base de ces propositions, nous utilisons la définition suivante pour désigner par la suite le terme *cloud computing*[3] :

Definition 4 *"Le cloud n'est pas une simple technologie, mais un ensemble de technologies combinées et utilisées dans le but d'approvisionner des clients distants avec des services de différents types et échelles. En d'autres termes, grâce au cloud la puissance de calcul et de stockage d'informations est proposée à la consommation sous forme de services et facturée d'après l'utilisation réelle avec une qualité de service liée à l'élasticité, l'évolutivité et la fiabilité de la plateforme."*

A la vue des différentes définitions enoncées précedemment, il apparait que l'idée fondamentale portée par le *cloud computing* est celle de l'informatique comme utilitaire, rendue possible par le biais de la mutualisation. Cette idée forte, actuellement portée par

[3]Dans la suite du manuscrit, nous utilisons indifféremment les termes *cloud* et *cloud computing* pour

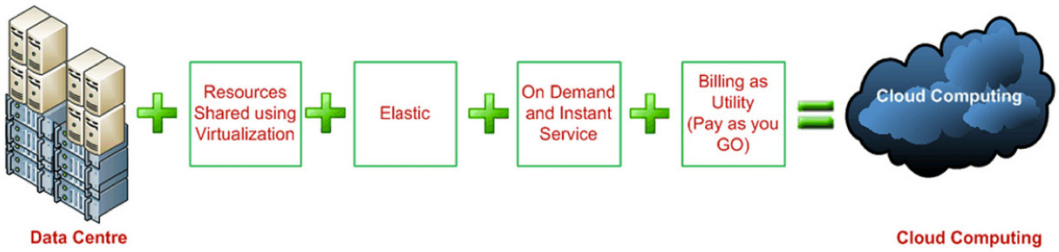


FIGURE 1.1 – Définition du *cloud computing* [2]

l'informatique reste pourtant similaire à d'autres formes de mutualisation apparues dans d'autres domaines technologiques, tels la mutualisation de la production et de l'acheminement de l'énergie électrique qui révolutionna, environ un siècle auparavant, l'ensemble des infrastructures et usines de production. On comprend ainsi que par ce biais, c'est aussi la réduction des coûts de production de services qui s'opère.

La figure 1.1 illustre graphiquement les points caractéristiques du *cloud computing*. Avec la même idée, le NIST [13], identifie cinq caractéristiques essentielles pour une infrastructure de *cloud* :

***Self-service* à la demande :** un client peut rapidement, généralement sans intervention nécessaire du coté fournisseur[4], avoir à disposition les ressources informatiques dont il a besoin.

Large accès réseau : les services offerts sont disponibles via le réseau et accessibles grâce aux mécanismes et protocoles standards qui supportent l'utilisation de plates-formes clientes hétérogènes telles que les téléphones mobiles, tablettes, ordinateurs portables et postes de travail.

Mise en commun des ressources : les ressources du fournisseur sont partagées afin de servir plusieurs clients en utilisant un modèle *multi-tenant*, supporté par des ressources physiques et virtuelles différentes (stockage, puissance de calcul, mémoire, bande passante réseau et machines virtuelles), dynamiquement assignées et réassignées selon les besoins du client. En général, le client n'a aucun contrôle ou connaissance de l'emplacement exact des ressources fournies, mais peut être capable de spécifier l'emplacement à un niveau supérieur d'abstraction (pays, état ou centre de données).

Elasticité rapide : représente la capacité d'étendre ou de réduire de façon rapide et efficace l'approvisionnement en ressources pour répondre aux exigences des demandes.

Paiement à l'usage : le concept *pay-per-use* est adopté pour gérer la facturation des services utilisés par les clients. Dans ce cadre, l'utilisateur paie uniquement pour les ressources qu'il a utilisées réellement.

se réferer à cette définition

[4]Dans la suite du manuscrit, nous utilisons indifféremment les termes fournisseur et fournisseur de service pour désigner le *Cloud Service Provider*

1.2.2 Modèles de service du *cloud*

Le concept de service à la demande fourni par une infrastructure *cloud* couvre l'ensemble des services offerts par une infrastructure opérée. Toutefois, il est reconnu que les niveaux de services offerts par un opérateur de *cloud* peuvent être scindés en différentes catégories. Par exemple Zhou *et al.* [15] propose une décomposition en six catégories : Infrastructure as a Service (IaaS), Network as a Service (NaaS), Platform as a Service (PaaS), Identity and Policy Management as a Service (IPMaaS), Data as a Service (DaaS), Software as a Service (SaaS). Alors que Rimal *et al.* [16] les décompose en dix catégories : SaaS (Software as a Service), PaaS (Platform as a Service), HaaS (Hardware as a Service), DaaS ([Development, Database, Desktop] as a Service), IaaS (Infrastructure as a Service), BaaS (Business as a Service), FaaS (Framework as a Service), OaaS (Organization as a Service).

Dans la suite du manuscrit, nous considérons la classification des modèles de services proposée par le NIST [13][17] qui fait référence dans la litterature et classifie ces derniers en trois modèles :

Software as a Service (SaaS) : représente la fourniture d'applications par un opérateur de *cloud* à ses usagers. A ce niveau de service, l'opérateur gère l'infrastructure matérielle, le système d'exploitation et les applications utilisées par le client qui, quant à lui, n'a pas plus de contrôle sur l'ensemble de ces composants.

Platform as a Service (PaaS) : représente la fourniture de toutes les ressources nécessaires pour développer et déployer des services et des applications par un opérateur de *cloud* à ses usagers.

Infrastructure as a Service (IaaS) : également appelé *Hardware as a Service* : représente la fourniture de ressources matérielles telles que l'espace de stockage, la puissance de calcul ou des capacités de communication réseau en tant que service à la demande par un opérateur de *cloud* à ses usagers.

Pour chacun de ces niveaux, nous donnons dans la suite une description détaillée.

1.2.2.1 Le modèle Software as a Service (SaaS)

Le modèle SaaS représente la capacité donnée aux consommateurs d'utiliser des applications logicielles d'un fournisseur par le biais d'une infrastructure de *cloud*. Les applications sont accessibles à partir de divers périphériques qui peuvent principalement être un client léger comme un navigateur Web, ou une API. Le consommateur ne gère ni contrôle l'infrastructure *cloud* sous-jacente, comprenant réseau, serveurs, systèmes d'exploitation, capacité de stockage, librairies et composants logiciels à l'exception éventuelle des paramètres de configuration de l'application. Parmi les applications SaaS connues et largement utilisées qui forment les exemples les plus pertinents de ce modèle de services,

on cite Google Docs[5], Google Gmail[6], Microsoft 365[7], Twitter[8], Facebook[9] et Flickr[10].

Le principal élément d'attractivité du modèle SaaS réside dans le coût d'utilisation réduit pour le client final. En effet, pour ce dernier, s'approvisionner en SaaS coûte, dans la plupart des cas, moins cher que le paiement des droits d'utilisation d'un logiciel dont il acquiert une licence à installer sur une inftrastrucuture opérée par ses soins. En outre, les avantages de ce modèle pour une entreprise qui souhaiterait avoir recours à ce type de service sont :

Réduction des coûts de main d'oeuvre : les infrastructures informatiques nécessitent des employés en charge de leur gestion qui présentent un coût (relatif par exemple à la masse salariale, l'espace physique qui leur est alloué, ...). L'externalisation des ressources réduit d'autant la nécessité de personnel.

Réduction des coûts d'infrastructure : Comme pour le personnel, un système informatique classique nécessite du matériel (serveurs, équipements d'interconnexion, ...). L'utilisation de services du *cloud* limite les coûts d'infrastructure, de gestion et de sécurité des ressources car c'est la responsabilité du fournisseur.

Marketing des applications : lorsqu'un prestataire développe une application pour un marché très étroit, il pourrait rencontrer des problèmes de marketing pour cette dernière. Cependant, grâce au SaaS, le marché est ouvert aux fournisseurs. Ainsi, le marketing et la publicité des nouvelles applications sont plus faciles et mieux ciblés.

Contrôle sur le logiciel : l'utilisation du SaaS permet au fournisseur d'accroître son contrôle sur l'utilisation des logiciels qu'il fournit en limitant la distribution des copies sans licence et en permettant une mise à niveau plus aisée à mettre en oeuvre et régulière pour ses clients. Le SaaS permet également aux fournisseurs de créer et contrôler des flux de revenus multiples avec le modèle *one-to-many*, réduisant ainsi la duplication des logiciels et des frais généraux [17].

Familiarité avec le World Wide Web : en général, les usagers savent manipuler les applications sur le *web*. Par conséquent, la courbe d'apprentissage pour l'utilisation de ce type d'applications se trouve réduite.

Au delà des avantages, le modèle SaaS souffre de certains inconvénients qui peuvent limiter son adoption. Nous les citons ci-après :

Une offre de logiciels limitée : Une organisation avec des besoins en applications informatiques très spécifiques, peut ne pas être en mesure de trouver de fournisseur *cloud* capable de les lui fournir via un modèle SaaS nécessitant pour elle une migration partielle et le maintien du coût lié à une part de ses infrastructures.

L'incompatibilité entre fournisseurs : Un client peut s'approvisionner chez plusieurs fournisseurs de solution SaaS. Dans ce contexte, la question de l'interopérabilité entre applications se pose. En effet, ce dernier pourrait faire face à des

[5] docs.google.com
[6] mail.google.com
[7] microsoftonline.com
[8] twitter.com
[9] facebook.com
[10] flickr.com

problèmes d'incompatibilité liés par exemple aux plateformes et environnements de développement utilisés par les fournisseurs.

La concurrence du modèle *open-source* : les logiciels diffusés selon le modèle *open-source* peuvent présenter un vrai défi aux fournisseurs de services *cloud*, car ils sont affranchis de coût d'utilisation et, même si opérés par un client au sein de sa propre infrastructure, ils peuvent être globalement moins coûteux qu'une solution SaaS.

1.2.2.2 Le modèle Platform as a Service (PaaS)

De façon analogue au modèle SaaS, le modèle PaaS est un autre modèle de fourniture de services *cloud*. Toutefois, il représente le cas où le service fourni est un environnement complet de développement et d'exécution d'applications, qui n'est pas réduit, pour le client, à la seule utilisation d'une application, comme forme de service offert. Le consommateur ne gère ni contrôle l'infrastructure *cloud* sous-jacente (réseau, serveurs, systèmes d'exploitation et stockage), mais il a le contrôle sur les applications déployées et éventuellement les paramètres de configuration de l'environnement d'hébergement. Les solutions PaaS diffèrent des solutions SaaS dans le fait qu'elles fournissent des plateformes de développement virtuelles, hébergées dans le *cloud* et accessible via un navigateur Web. En d'autre termes, le PaaS fournit toutes les ressources nécessaires pour construire et développer des applications et des services dans leur totalité, à partir d'Internet et sans avoir à mettre en oeuvre l'ensemble des outils liés au cycle de développement logiciel. Les services PaaS incluent ainsi les outils de conception, développement, test, déploiement et hébergement d'applications. Typiquement, les services PaaS incluent l'intégration des Web services, de bases de données, de composants de sécurité, d'évolutivité et de stockage. Les exemples de services PaaS qui forment le coeur des offres actuelles sont Forces.com[11], Google App Engine[12] et Red Hat's OpenShift[13].

Les avantages du modèle PaaS, en comparaison de solutions standard pour le développement d'applications, résident principalement dans leur accès par un client léger qui permet à des équipes géographiquement éloignées de travailler ensemble et de fusionner des services Web à partir de sources multiples. Comme pour le modèle SaaS, le modèle PaaS est aussi pour le client une source de réduction de coûts grâce à l'utilisation de services de sécurité et d'évolutivité intégrés à l'infrastructure, plutôt que d'avoir à les obtenir et les tester séparément.

Toutefois, ce modèle présente aussi l'inconvénient important de pouvoir, pour un client, ne bénéficier que de l'offre de solutions techniques proposée par le fournisseur choisi. Il peut se trouver ainsi réduit à des choix de plates-formes ou composants logiciels qui ne sont pas en accord total avec ses besoins propres. Pour dépasser cette contrainte, il est possible d'adopter une approche multi-fournisseurs qui toutefois complexifie l'ensemble du système de développement et augmente aussi les coûts pour le client.

[11] salesforce.com
[12] cloud.google.com/appengine/
[13] openshift.com

1.2.2.3 Infrastructure as a Service (IaaS)

Le modèle IaaS représente le niveau de production de services le plus réduit dans le cadre des modèles de services *cloud*. Il consiste en la fourniture au client de puissance de calcul, stockage, capacités de communication en réseaux et autres ressources informatiques fondamentales sur lesquelles le client peut déployer et exécuter les logiciels de son choix, qui peuvent inclure des systèmes d'exploitation et des applications. Le client ni ne gère ni ne contrôle l'infrastructure *cloud* sous-jacente mais a le contrôle sur les systèmes d'exploitation, le stockage et les applications déployées. Il peut aussi avoir un contrôle limité sur certains composants réseaux comme par exemple le pare-feu qui lui est associé. Le modèle IaaS représente le modèle qui reflète le plus la différence entre les systèmes IT traditionnels et les infrastructure basées sur le *cloud* : il incarne la livraison d'une infrastructure informatique en tant que service. En d'autres termes, là où les modèles SaaS et PaaS fournissent une partie logicielle aux clients, le modèle IaaS ne le fait pas. En effet, il offre un seul niveau matériel de sorte que le client, puisse y déployer toute solution logicielle dont il conserve la responsabilité. Les exemple de services IaaS qui forment le coeur de cette offre sont : Amazon Web Service (AWS)[14], Microsoft Azure[15] et Google Compute Engine (GCE)[16].

1.2.3 Modèles de déploiement du *cloud*

Selon l'emplacement physique des ressources mises en oeuvre pour la fourniture de service ainsi que la politique de distribution des services, différents modèles de déploiement de *cloud* sont adoptés. En effet, indépendamment du type de service fourni, un *cloud* peut être classifié comme :

Cloud public : les services du *cloud* sont ouverts au grand public (individus et sociétés). Son administration est détenue par une organisation tierce.

Cloud privé : le *cloud* est exploité par une seule société ou organisation. Son fonctionnement est assuré par cette même organisation ou bien par une tierce partie. L'infrastructure est géographiquement installée dans les locaux de la société ou ailleurs.

Cloud de communauté : Il a pour objectif de fournir les mêmes avantages et garanties que celles offertes par un *cloud* privé. Le *cloud* est partagé par plusieurs organisations qui ont des préoccupations communes, comme par exemple les exigences et politiques de sécurité.

Cloud hybride : Il s'agit d'une composition de deux ou plusieurs *clouds* (privé, communautaire ou public) qui demeurent des entités uniques, mais liés par une technologie normalisée qui permet le partage des données et des applications.

Dans la suite, nous détaillons chacun de ces modèles de déploiement.

[14] aws.amazon.com
[15] azure.microsoft.com
[16] cloud.google.com/compute/

1.2.3.1 Cloud public

Selon Baun *et al.* [18], le *cloud* public comprend toutes les offres *cloud* où les fournisseurs et les clients potentiels n'appartiennent pas à la même unité organisationnelle. Ces fournisseurs offrent habituellement un portail Web en self-service, où les utilisateurs peuvent spécifier leurs besoins. Armbrust *et al.* [11] le définit simplement comme un *cloud* où ses offres de services sont disponible et facturées à la manière *Pay-per-use*. L'infrastructure d'un *cloud* public est détenue et opérée par une société fournisseur de services, comme par exemple Amazon, Google, Salesforce.com ou Microsoft.

Le *cloud* public offre pour le client plusieurs avantages et caractéristiques attractives comme le déploiement de ses services sur des plateformes gérées, maintenues et mises à jours par le fournisseur, associée à la facturation selon l'utilisation réelle qui, associée au partage de ressources physiques, réduit les coûts. Economiquement, c'est donc la réduction immédiate et drastique des coûts d'infrastructure et d'exploitation qui motive l'utilisation d'un *cloud* public. En dépit de ces avantages, le *cloud* public présente aussi des barrières fortes quant à son adoption par une entreprise cliente. La principale réside dans le niveau de confiance qui doit être acquis entre un fournisseur de service *cloud* et le client. En effet, la délégation forte de moyens d'infrastructures, de plate-formes de services mais aussi de données à caractère privé et parfois très sensibles, font que le client peut préférer conserver la maîtrise de ces éléments. La sécurisation d'une infrastructure de *cloud* public est donc un enjeu essentiel et nous détaillerons cet aspect dans la section 1.4.

1.2.3.2 Cloud privé

Selon le NIST [13], un *cloud* privé représente une infrastructure de *cloud computing* dédiée à l'usage exclusif d'une seule organisation, comportant plusieurs consommateurs. Il peut être détenu, géré et exploité par l'organisation, une tierce partie, ou une combinaison des deux. Il peut être hébergé dans les locaux de l'organisation ou ailleurs. Baun *et al.* [18] le définit comme étant un *cloud* où les fournisseurs et utilisateurs appartiennent à la même unité organisationnelle.

En d'autres termes, les *clouds* privés offrent certains avantages du *cloud* public, tout en permettant à l'organisation de conserver un meilleur contrôle sur ses données et ses plate-formes de services. En général, ce sont les grandes organisations, ou les agences gouvernementales qui, le plus souvent, optent pour le déploiement d'une telle infrastructure et ce, par leurs propres moyens. La raison liée à la taille des organisations qui adoptent cette solution réside dans le coût que peut engendrer le déploiement d'un *cloud* privé. En effet, la mise en œuvre, d'une telle infrastructure demande plus de dépenses qu'une architecture IT traditionnelle. Ainsi, une organisation qui opte pour ce type de *cloud* doit être suffisamment conséquente, solide et experte, pour se permettre d'engager une migration de son infrastructure vers ce type de solution. A contrario, on note que les petites organisations peuvent également bénéficier des avantages du *cloud* privé, en renonçant à un déploiement de ce type d'infrastructure par leurs propres moyens mais en faisant appel à un fournisseur de *cloud* privé.

1.2.3.3 Cloud communautaire

Selon le NIST [13], le *cloud* communautaire représente une infrastructure *cloud* dédiée à un usage exclusif d'une communauté de consommateurs qui représente des organisations ayant des préoccupations partagées. Les ressources partagées d'une telle infrastructure peuvent être utilisées par des groupes qui ont des considérations communes, telles que les exigences de conformité, des objectifs d'affaires non concurrentielles ou un besoin de mutualiser des moyens de sécurité de haut niveau. Le *cloud* communautaire peut être détenu, géré et exploité par une ou plusieurs organisations appartenant à la communauté, une tierce partie ou une combinaison des deux. Il peut être hébergé au sein des locaux de la communauté ou délocalisé.

En d'autre termes, le *cloud* communautaire représente un type de déploiement intermédiaire entre le *cloud* public et le *cloud* privé. L'infrastructure de *cloud* est ici mutualisée entre plusieurs parties prenantes, mais restreintes ou pas à ces seules parties selon des conditions définies par la communauté. En ce sens, Buyya *et al.* [19] classifie d'ailleurs les *clouds* communautaires en deux types : privés et publics. Par exemple, un ensemble d'organisations qui mettent en commun leurs ressources pour seulement servir leurs propres fins, construisent ainsi un *cloud* communautaire privé. Contrairement à cela, un revendeur tel que ZIMORY[17] peut rassembler des ressources de différents fournisseurs et les revendre après. D'ailleurs, selon Briscoe *et al.* [20], les utilisateurs de *clouds* communautaires peuvent être des fournisseurs ou des utilisateurs finaux, ce qui permet de créer un marché virtuel composé de ressources et services informatiques entre les différentes organisations. On note toutefois que la gestion d'un *cloud* communautaire peut s'avérer complexe principalement du fait de la fluctuation des entités responsables de cette infrastructure. Cet état de fait peut rendre difficile, d'un point de vue technique et décisionnel, l'ensemble des activités de gestion concernant les ressources, la confidentialité, la performance et les exigences de sécurité.

1.2.4 Cloud hybride

Pour terminer la présentation des modèles de déploiement, nous nous intéressons au *cloud* hybride qui représente la combinaison de deux ou plus des types de *cloud* mentionnés auparavant. Le NIST [13] le définit comme étant une infrastructure composée de deux ou plusieurs infrastructures de *cloud* distinctes (privé, communautaire ou public) qui demeurent des entités uniques, mais qui sont liées entre elles par des technologies normalisées ou propriétaires, qui permettent la portabilité des données et des applications. Selon Buyya *et al.* [19], Un *cloud* hybride prend forme quand un *cloud* privé est complété par une capacité de calcul à partir de *clouds* publics.

Une caractéristique essentielle qui différencie le *cloud* hybride des autres types de déploiement est l'utilisation du *Cloud Bursting* ou *Cloudburst*. Un *Cloudburst* désigne généralement le déploiement d'une application dynamique qui, tout en s'exécutant principalement sur l'infrastructure interne d'une organisation, peut également être déployée sur un *cloud* public pendant les pics de demande [17].

[17] zimory.com

1.3 Les technologies du cloud

Le *cloud computing* n'est pas une technologie mais consiste plutôt en un large ensemble de technologies qui, mises ensemble, offrent les fonctionnalités que le *cloud* supporte [17]. Parmi celles-ci, on compte au niveau matériel la virtualisation et les processeurs multi-coeurs ; au niveau calcul distribué, l'informatique utilitaire et les grilles de calcul ; au niveau des technologies de l'Internet, les architectures orientées services, le web 2.0 et les web services ; enfin, au niveau de la gestion, l'automatisation au niveau data-center et l'informatique autonome [17]. Dans la section qui suit, nous concentrons notre présentation sur deux briques majeures qui forment le coeur des infrastructures *cloud* et qui s'intègrent dans le cadre de notre étude : la virtualisation et les plate-formes de gestion de ces infrastructures.

1.3.1 La virtualisation

Dans cette section, nous présentons la technologie de la virtualisation, technologie logicielle qui a pour objectif d'émuler le matériel qui héberge un ensemble de systèmes virtualisés.

1.3.1.1 Définitions

De nombreuses propositions existent pour définir le concept de virtualisation. Nous choisissons ici d'en citer deux sont issus d'états de l'art académiques du domaine. Selon Nanda *et al.* [21] la virtualisation est définit comme suit :

Definition 5 *"Virtualization is a technology that combines or divides computing resources to present one or many operating environments using methodologies like hardware and software partitioning or aggregation, partial or complete machine simulation, emulation, time-sharing, and others."*

White *et al.* [22] la définit comme :

Definition 6 *"Virtualization is a mechanism permitting a single physical computer to run sets of code independently and in isolation from other sets."*

En d'autres termes, la virtualisation représente une technologie informatique, consistant à faire fonctionner et cohabiter plusieurs environnements logiques indépendants, séparément sur une même machine physique. La virtualisation représente ainsi une extension de l'émulation qui consiste à substituer un ou plusieurs éléments physiques par une application.

1.3.1.2 Techniques de virtualisation

L'implémentation du principe de virtualisation, tel qu'énoncé précédemment, peut prendre plusieurs formes que l'on exprime en termes de techniques et niveaux. La figure 1.2 décrit les architectures de ces différentes techniques.

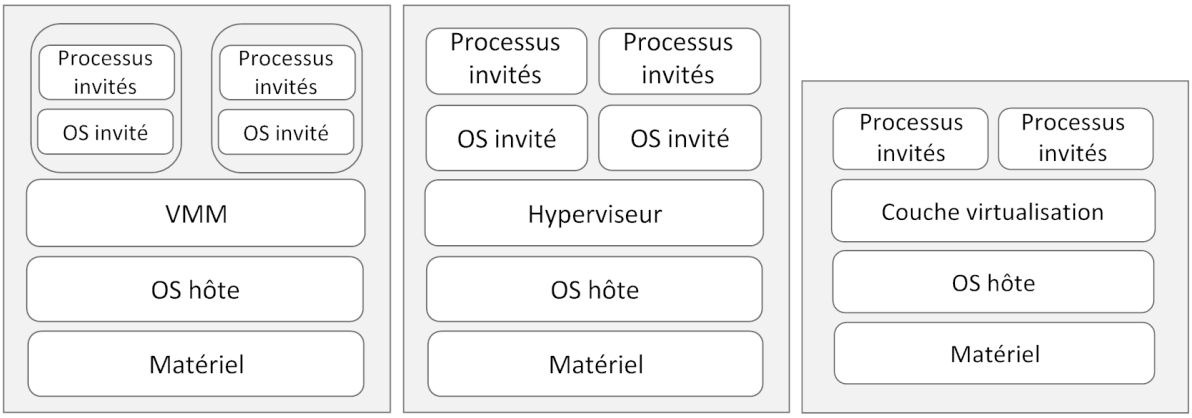


FIGURE 1.2 – Architectures des techniques de virtualisation [3] (a) virtualisation totale ; (b) paravirtualisation ; (c) virtualisation par conteneurs

La virtualisation totale : La Figure 1.2.a décrit les différentes couches que doit comprendre un système supportant la virtualisation totale. Ce type de virtualisation consiste à émuler l'intégralité d'une machine physique pour le système invité afin que ce dernier croit s'exécuter sur une véritable machine physique. La machine physique hôte doit posséder un OS ainsi qu'une sur-couche logicielle appelée *Virtual Machine Monitor* (VMM). Un avantage de cette technique de virtualisation réside dans sa capacité à émuler n'importe quelle architecture matérielle. Ainsi, l'utilisateur peut faire fonctionner les OS qu'il désire indépendamment de l'architecture du système hôte. Dans la virtualisation totale, la machine hôte doit implémenter une gestion complète de la couche matérielle des VM. Les performances de ces dernières sont donc limitées par la performance de la couche applicative implémentée par le système hôte ainsi que par la qualité de l'émulation du matériel. Les exemples les plus significatifs de technologies supportant cette technologie sont Parallels Desktop for Mac, Parallels Workstation, VMware Workstation, VMware Server, VirtualBox, Win4BSD, et Win4Lin Pro.

La virtualisation assistée par le matériel : Ce type de virtualisation, aussi appelé *accelerated virtualization, native virtualization* ou *Hardware Virtual Machine* par Xen a pour but de permettre une virtualisation totale, telle que définie précédemment, efficace en se reposant sur des capacités matérielles. Dans ce cas, le matériel (processeur) est modifié et se voit ajouter des extensions de virtualisation. L'avantage de cette technique par rapport à la seule visualisation totale réside dans la capacité de la machine virtuelle à tirer pleinement profit de la puissance et des performances de la machine hôte. Cette technologie a par exemple été intégrée aux processeurs à base d'architecture x86 (Intel VT-x et AMD-V).

La paravirtualisation : La paravirtualisation représente une technique qui permet à plusieurs systèmes d'exploitation de fonctionner simultanément sur le même périphérique matériel, tout comme la virtualisation totale. Cependant, à la différence de cette dernière, les OS invités sont modifiés et ont conscience d'être virtualisés.

La couche logicielle située entre le matériel et les systèmes invités est appelée hyperviseur. Cet hyperviseur est responsable de l'application des politiques d'accès aux ressources matérielles pour les systèmes invités. La figure 1.2.b décrit l'architecture de ce type de virtualisation. Etant donné les modifications apportées, les VM et ainsi leur OS ont été adaptés au niveau de leur noyau afin de pouvoir communiquer directement avec l'hyperviseur. Ainsi, du fait de l'élimination de l'émulation complète du matériel, la paravirtualisation offre des gains de performance par rapport à la virtualisation totale [17]. Cependant sa flexibilité est limitée du fait que certaines distributions d'OS ne créent pas de versions adaptées à ce type de virtualisation. L'exemple le plus représentatif de cette technologie est Xen.

La virtualisation par conteneurs : Aussi appelée virtualisation au niveau du système d'exploitation *OS Level Virtualization (OSLV)*, elle représente une technologie qui consiste à séparer l'OS d'une machine en différents environnements utilisateurs distincts. Ainsi les utilisateurs de la machine ne se voient pas entre eux et les données sont séparées. Les environnements utilisateurs sont complètement cloisonnés. En effet, OLSV n'est pas une technologie de virtualisation "lourde" comme celles déjà décrites, mais une alternative beaucoup plus légère étant donné que dans cette technologie, l'OS est le même pour tous les utilisateurs [3]. En outre, on ne parle pas de systèmes invités mais d'environnements utilisateur cloisonnés. Il existe d'autre appellations à cette forme de virtualisation comme *containers, Virtualization Engines (VE), Virtual Private Servers (VPS)* ou *Jails*. La figure 1.2.c décrit l'architecture de cette technique de virtualisation.

Ce type de virtualisation est essentiellement utilisé sur les systèmes Linux, Unix et BSD. L'allocation des ressources est effectuée en donnant des quotas de disque à ces conteneurs ainsi qu'une limitation de la puissance CPU. La mémoire vive et la bande passante pour les communications en réseau peuvent également être limitées. La première implémentation connue de cette technologie est *chroot*. Depuis, de nombreuses implémentations avancées ont été développées comme par exemple : *Linux-VServer, OpenVZ, Solaris Containers, FreeBSD Jail, Docker* et *Linux Containers (LXC)*.

A ses débuts, ce type de virtualisation était utilisé chez les hébergeurs web qui proposent des environnements privés à moindre coût afin que les utilisateurs puissent gérer leurs sites web comme s'ils possédaient chacun un serveur personnel [23]. Actuellement, elle représente une technologie émergente dans les environnement de *cloud computing* du fait des avantages qu'elle offre en comparaison aux solutions de virtualisation lourde. Ceux-ci sont :

— La légèreté et rapidité d'exécution en raison de l'absence de la couche d'abstraction comme celle des technologies fondées sur un hyperviseur.
— La possibilité de créer et détruire un conteneur plus rapidement qu'une VM.
— A machine physique identique, la possibilité de déployer beaucoup plus de conteneurs que de VMs.

1.3.1.3 Linux Containers (LXC)

LXC représente un environnement de virtualisation de type conteneur qui permet l'exécution de plusieurs systèmes Linux isolés sur une seule machine. Nous le décrivons

ici car il est l'environnement sur lequel nous avons effectué les campagnes expérimentales présentées dans ce manuscrit. LXC fournit un ensemble de nouvelles commandes qui, à partir d'une simple racine, identique à celle utilisée pour un chroot, permet de créer, démarrer, arrêter ou encore surveiller un conteneur. Plus spécifiquement, il consiste en une extension comprenant un ensemble de fonctions, ajoutées au noyau Linux qui reposent sur les *Control Groups (cgroups)* pour l'isolation des ressources comme le processeur, la mémoire, les entrées/sorties et le réseau. un *cgroup* produit également de nombreux isolateurs permettant de contrôler un jeu de processus et leurs fils :

— *UTS namespace :* permet d'avoir pour chaque conteneur son propre *hostname*, distinct de celui de l'hôte.
— *IPC Namespace :* permet d'avoir pour chaque conteneur son jeu privé de fonctions.
— *User Namespace :* permet d'avoir par conteneur, des utilisateurs distincts de ceux de l'hôte, y compris root.
— *PID Namespace :* permet au conteneur de disposer de sa propre liste de processus. Les processus hôte ne sont donc plus visible du conteneur.
— *Network Namespace :* permet au conteneur de disposer de ses propres interfaces réseaux, y compris la boucle locale *(localhost)*.

Le projet LXC est disponible sous licence GNU LGPLv2.1+ et est composé de : (1) la librairie liblxc, (2) de multiple couplages à des langages cibles tels que : python, lua, Go, ruby et Haskell et (3) des patrons de gestion de conteneurs.

1.3.2 Les plateformes de gestion de cloud

Les plateformes de gestion de *cloud computing* servent à construire et gérer des architectures distribuées basées sur la virtualisation, quelque soit le type de cette dernière. Grâce à ce type de plateforme, les sociétés, entreprises ou administrations, peuvent mettre en place leurs propres architectures de *cloud* privé, hybride ou communautaire, en exploitant leurs infrastructures existantes. Le développement de ces plateformes a connu une forte croissance ces dernières années. Outre les solutions propriétaires, principalement mises en œuvre par les fournisseurs publics, une multitude de solutions *open source* ont été développées. Dans cette section nous présentons les principaux composants qui forment une architecture de *cloud* et nous décrivons quelques plateformes de gestion d'infrastructure *cloud* diffusées librement qui forment le coeur de l'offre libre actuelle, à savoir Eucalyptus, OpenNebula et OpenStack.

1.3.2.1 Eucalyptus

Eucalyptus [24] représente une plateforme ayant une structure hiérarchique. Elle a été conçue comme alternative *open source* à Amazon EC2. D'ailleurs, son interface utilisateur appelée *Euca2ools* est très proche de celle de ce dernier.

La Figure 1.3 représente l'architecture globale d'Eucalyptus où quatre composants sont représentés. Elle montre également la hiérarchie entre ces différents composants. Afin de mieux comprendre la fonction de chacun d'entre eux ainsi que l'encapsulation de leurs tâches, nous en proposons une description qui commence par le niveau le plus bas.

— *Node Controller (NC) :* est responsable du contrôle de l'exécution, l'inspection et

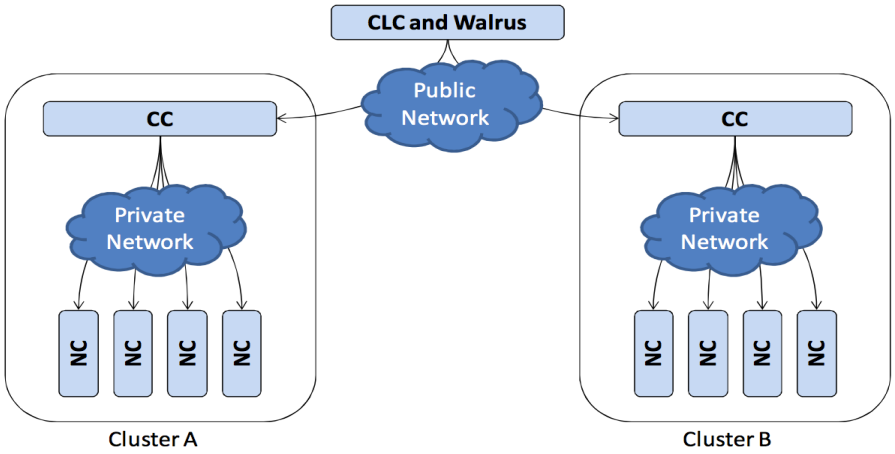


FIGURE 1.3 – Architecture d'Eucalyptus

la terminaison des VM de l'hôte sur lequelle il est implémenté.

— *Cluster Controller (CC) :* assure la collecte de données concernant les VM hébergées dans les *NC* qu'il supervise. Il assure également la gestion du réseau virtuel au niveau d'une grappe de serveurs.

— *Storage Controller (Walrus) :* représente un service de stockage qui implémente l'interface d'Amazon S3, fournissant un mécanisme pour le stockage et l'accès aux images de VM et les données des utilisateurs.

— *Cloud Controler (CLC) :* représente le noeud ayant le plus haut niveau. Il représente le point d'entrée du *cloud*, pour les utilisateurs et les administrateurs. Il assure la supervision et la gestion des noeud en assurant une planification de haut niveau après avoir collecté les informations de disponibilité sur les ressources fournies par les *CC*.

1.3.2.2 OpenNebula

OpenNebula est une plateforme de gestion conçue pour tout type de *cloud* : privé, public ou hybride [25]. Cependant, contrairement à Eucalyptus où la fonction de gestion est distribuée, elle est, dans le cas d'OpenNebula, très centralisée. Cela permet une administration facile et une grande facilité pour la personnalisation. En revanche, cette architecture représente un véritable goulot d'étranglement dans le cas d'une grande infrastructure. Par conséquent, il est établi que OpenNebula est plutôt efficace dans la gestion des *cloud* privés ayant une taille moyenne.

Comme décrit dans la Figure 1.4 [26], l'architecture d'OpenNebula comprend trois couche : *Drivers, Core* et *Tools* [27].

— *Drivers :* représente la couche responsable des interactions directes entre le matériel et la pile logicielle. Elle gère les communications avec les hyperviseurs et également les mécanismes de transfert de données et fichiers.

— *Core :* consiste en un ensemble de composants servant au contrôle et à la surveillance des VM, des réseaux virtuels, du stockage et des machines hôtes.

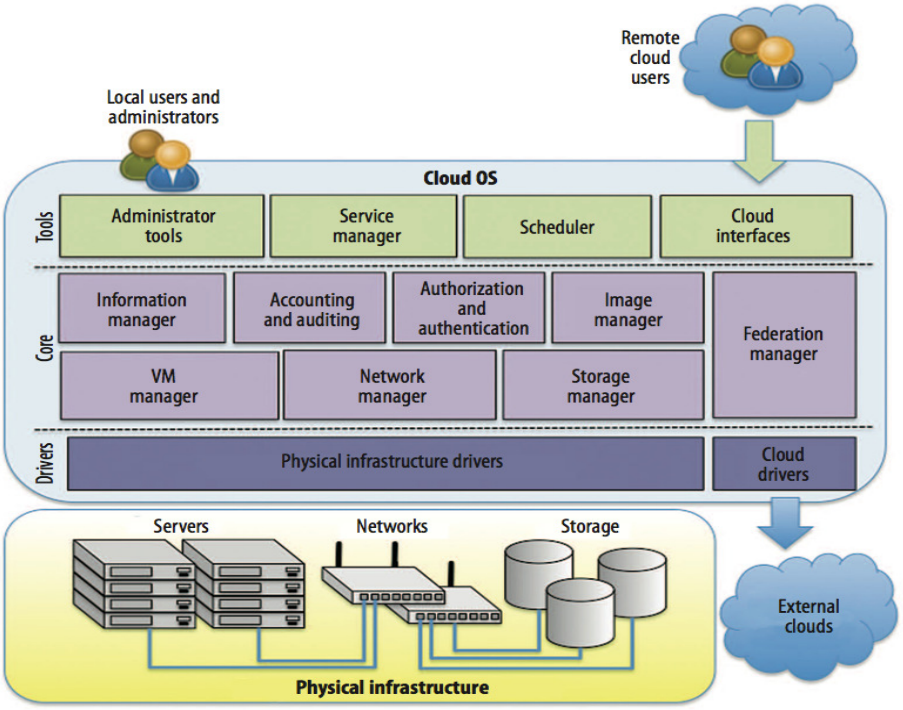


FIGURE 1.4 – Architecture d'OpenNebula

— *Tools :* comprend un ensemble d'outils assurant des fonctions d'administration et de gestion, tel qu'une interface en lignes de commandes pour les utilisateurs et les administrateurs, un ordonnanceur, une implémentation de *libvirt* et des interfaces de programmation RESTful.

1.3.2.3 OpenStack

Openstack est une plate-forme logicielle qui permet la construction de *clouds* privés, publics ou hybrides. Nous détaillons ici à cette plateforme car son architecture modulaire représente un très bon reflet de tout ce qu'un *cloud* doit déployer. OpenStack est un logiciel libre distribué selon les termes de la licence Apache. Son objectif principal est de fournir une plate-forme, simple et adaptable capable de répondre à différentes tailles d'infrastructures. Initialement, la NASA à créé un *cloud* privé sur la plateforme Eucalyptus. Toutefois, Eucalyptus ne s'est pas montré aussi flexible et ouvert que la NASA l'espérait. Il a alors été décidé de construire un nouveau contrôleur de *cloud* appelé "Nova" qui a ensuite été publié sous une licence *open-source*. Au même moment, Rackspace[18] se préparait à faire la même chose pour son moteur *Compute* et son contrôleur de stockage. Il a donc été décidé de fusionner ces initiatives pour créer un logiciel libre pour les infrastructures *cloud*. En juillet 2010, le lancement d'OpenStack a été annoncé et en octobre de la même année la première version baptisée "Austin" a été rendue publique. A ce jour, on compte onze versions d'Openstack qui sont mise en oeuvre par une très grande communauté,

[18]Rackspace est un fournisseur de service d'hébergement et de *cloud computing*

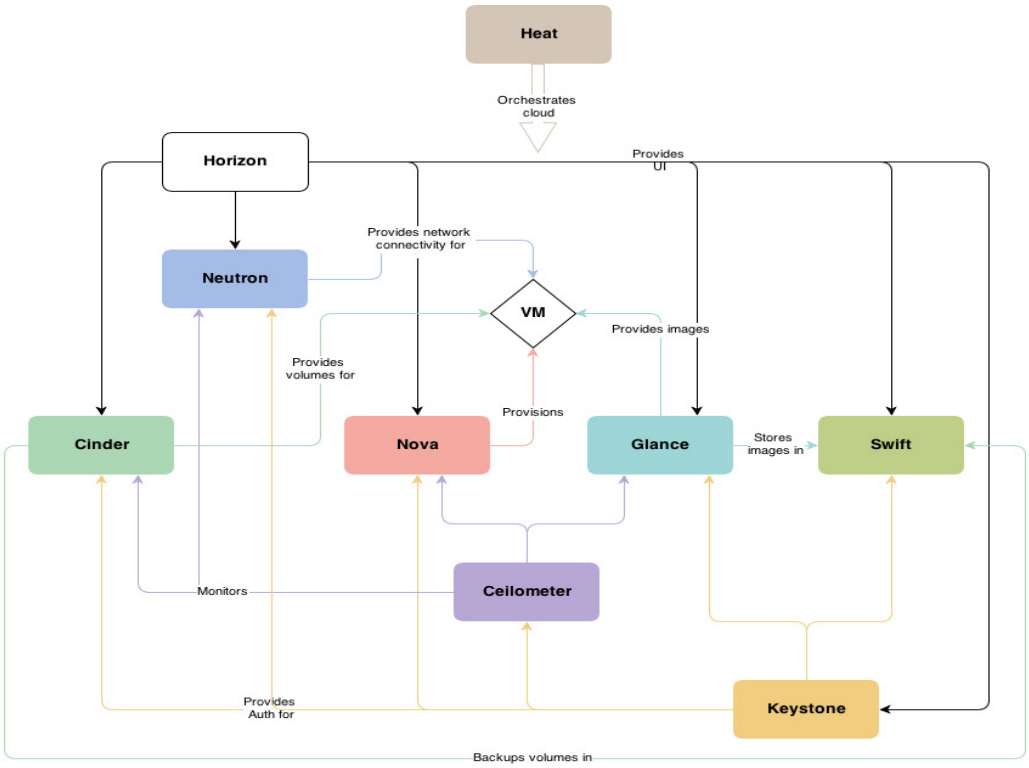


FIGURE 1.5 – Architecture conceptuelle des relations inter-services d'OpenStack [4]

comprenant de grandes entreprises qui participent à l'écosystème du *cloud* telles que : AMD, Citrix, Dell, Intel, Microsoft et Cisco.

OpenStack possède une architecture modulaire qui comprend de nombreux composants. Par ce biais, Openstack permet à son opérateur de construire son infrastructure en assemblant tout ou partie de ces composants qui présentent une indépendance forte dans l'architecture. La figure 1.5 [4] représente une vue fonctionnelle de ces composants. Elle décrit comment ces différents composants sont interconnectés ainsi que la nature de leurs relations. Nous décrivons chacun d'entre eux ci-après.

Service d'application (Nova) : Openstack Compute (Nova) est le contrôleur de l'infrastructure de *cloud computing*. Il est écrit en Python et utilise plusieurs bibliothèques externes comme Eventlet (pour la programmation concurrente), Kombu (pour la communication AMQP), et SQLAlchemy (pour l'accès aux bases de données). Il est conçu pour gérer et automatiser plusieurs ressources informatiques et peut travailler avec la plus part des technologies de virtualisation existante (KVM, XenServer, etc.).

Service réseau (Neutron) : Ce service gère des réseaux à la demande pour les différents *tenants*. C'est un service distribué qui est mis en oeuvre sur plusieurs noeuds de l'infrastructure *cloud*. Le gestionnaire de réseau dans OpenStack est un service fortement indépendant. Cette indépendance s'illustre d'ailleurs par le remplacement du composant initial nova-network par Neutron dans les distributions

actuelles.

Service de stockage en mode objet (Swift) : Ce service gère le stockage de données dans le *cloud*. Les données sont stockées de façon redondante pour assurer la tolérance aux pannes (les objets et les fichiers sont sauvegardés sur plusieurs disques-durs répartis sur plusieurs serveurs).

Service de stockage en mode bloc (Cinder) : Ce service gère les disques pour les machines virtuelles. Il fournit des possibilités de stockage de blocs en vue de leur utilisation par les instances d'OpenStack. Le système de stockage de bloc gère la création, l'attribution et le retrait des blocs des serveurs. Il est approprié pour les gestions sensibles à la performance telles que le stockage de base de données ou les systèmes de fichiers extensibles.

Service de tableau de bord (Horizon) : Le tableau de bord OpenStack offre aux administrateurs et aux utilisateurs une interface graphique pour gérer automatiquement les ressources mises en oeuvre dans un *cloud* OpenStack. Sa conception extensible facilite son interaction avec les ressources OpenStack.

Service d'image (Glance) : Ce service gère le stockage des images des VM. Il offre la possibilité de copier ou de faire des instantanés d'images en cours d'utilisation dans OpenStack. Les images stockées peuvent être utilisées comme des modèles pour générer d'autres instances.

Service d'identité (Keystone) : Ce service a deux fonctions principales : (1) gérer les utilisateurs en les définissant et en paramétrant leur contrôle d'accès aux ressources de l'infrastructure ; et (2) cataloguer les services et les rendre accessibles aux autres services ainsi que leurs points d'accès respectifs.

Service de collecte de données (Cielometer) : Ce service permet de fournir une collecte de données précise et détaillée de l'usage des ressources de l'infrastructure. Il permet ainsi, entre autres fonctions de surveillance, la mise en oeuvre des services de facturation fondés sur l'usage réel des ressources.

Service d'orchestration (Heat) : Heat représente le composant responsable de l'orchestration des actions à mettre en oeuvre entre les différents composants pour réaliser une opération de gestion dans une infrastructure *cloud*. En d'autres termes, il permet aux développeurs de stocker les exigences d'une application de *cloud* dans un fichier de script qui définit les ressources nécessaires au déploiement de cette application.

Ces composants sont liés les uns aux autres à différents degrés. Au coeur des interactions, on trouve Keystone qui interagit avec quasiment tous les composants OpenStack car il contient la fonction d'authentification et de catalogue de tous les services offerts par les composants ainsi que leur point d'accès. Cielometer, en tant que service surveillance de chaque composant d'une infrastructure *cloud* et Horizon, en tant qu'interface utilisateur pour les administrateurs et usagers, sont également en lien avec la majeure partie des composants. Glance, en tant que gestionnaire d'images, est associé à Swift qui lui permet de les stocker et les recouvrer. Enfin Nova, comme service de coeur de la plate-forme, interagit avec Neutron pour la mise en oeuvre des accès au réseau, Swift pour l'allocation

de l'espace de stockage en mode bloc aux machines virtuelles et enfin Glance, comme service de gestion de ces machines virtuelles. On constate ainsi que la richesse d'Openstack provient de sa forte modularité qui induit toutefois d'autres problématiques comme la difficulté à maintenir une infrastructure cohérente au fil des évolutions de ses composants qui peuvent évoluer indépendamment et engendrer des problématiques d'interopérabilité.

1.4 Le cloud et la sécurité

Le *cloud computing* représente un modèle de production de services IT qui attire fortement l'attention de tous les secteurs industriels et académiques. La raison de cet intérêt réside dans les avantages et bénéfices tels que présentés dans les sections précédentes. Toutefois, en dépit de ces avantages, la sécurité reste l'une des préoccupations majeures du *cloud computing*. Selon un sondage effectué en 2009 [12], 74% des Directeurs des Systèmes d'Information[19] ont cité la sécurité comme étant l'obstacle majeur pour leur adoption du *cloud*.

En effet, une propriété majeure du *cloud computing* est l'externalisation des ressources, donc le déplacement des applications et des bases de données sur des centres de données distants, où la gestion des données et des services n'est pas toujours fiable et transparente. Cet état de faits pose de nombreux problèmes et défis en matière de sécurité [28] qui sont le reflet de la complexité de l'ensemble de l'infrastructure et des services qui la composent. Pour en illustrer l'étendue, nous en citons les principaux, sans toutefois en faire un inventaire exhaustif. Ceux-ci incluent ainsi les vulnérabilités d'accessibilité, de virtualisation, des applications Web telles que l'injection SQL (Structured Query Language) et le *cross-site scripting*, les problèmes d'accès physique, les problèmes liés à l'intégrité, la confidentialité et le contrôle de données, la gestion des identités et l'authentification. Au delà de ces domaines techniques, s'ajoutent aussi des problématiques d'ordre juridique qui sont liées aux différents pays qui définissent le territoire et les droits relatifs aux données qu'ils hébergent [12] [29].

1.4.1 Vulnérabilités majeures du cloud computing

Il existe de nombreuses façons d'identifier et périmétrer les vulnérabilités de sécurité dans le *cloud*. Leur étendue et différence est le reflet de la difficulté actuelle à cerner et maitriser cette problématique. Selon le NIST [30], la sécurité, l'interopérabilité et la portabilité représentent les principaux obstacles d'une large adoption du *cloud*. Armbrust *et al.* [11], a identifié dix obstacles de l'adoption du *cloud computing*[20]. Ness *et al.* [31] identifie trois principaux obstacles quant à l'adoption du *cloud computing* qui sont le besoin de nouvelles approches pour la sécurité dans le *cloud*, pour la virtualisation des réseaux et l'automatisation des fonctions fonction de gestion qui passe par des approches autonomes. De son coté, Leavitt *et al.* [32] décrit six défis pour l'avenir du *cloud computing* : contrôle,

[19]DSI : responsable des technologies de l'information et de la communication dans une société

[20]à savoir : disponibilité des services, verrouillage des données, confidentialité et audit des données, surcharge du réseau due au transfert de données, imprévisibilité des performances, stockage évolutif, bugs dans les grands systèmes distribués, évolutivité rapide, *fate sharing*et l'octroi de licences de logiciels

performance, latence et fiabilité, sécurité et confidentialité, coût liés à la bande passante, restrictions dues aux technologies propriétaire[21] et normalisation et finalement transparence. Enfin, la *cloud Security Alliance (CSA)* [33][34] résume les principales menaces de la sécurité dans le *cloud* en : l'usage d'interfaces et API non sécurisées, les malveillances internes, les problèmes induits par le partage de technologies et de ressources, la perte et fuite de données, le piratage de compte, de service et de trafic, les profils de risques inconnus [2] et dernièrement, les abus et utilisations malveillantes du *cloud computing* qui forment le cadre d'étude du travail de recherche développé dans ce manuscrit. Pour conclure, on constate que, quelle que soit la démarche d'identification adoptée, les vulnérabilités du *cloud computing* sont liées à la complexité de son écosystème technologique nécessitant l'implication d'acteurs qui, outre les fournisseurs et les clients, sont impliqués dans la fourniture du service par l'intégration de leurs composants. La sécurisation intrinsèque de ces composants mais aussi de leur association pour créer une infrastructure complexe reste donc un important verrou de cette technologie.

1.4.2 Identification par modèle de services

Comme nous l'avons décrit dans la section 1.2.2, le *cloud* adopte majoritairement trois modèles de prestation par lesquelles différents types de services sont livrés à l'utilisateur final, à savoir SaaS, PaaS et IaaS. Chacun de ces modèles de services nécessite un niveau et un type d'exigences de sécurité différent [12] [29]. Dans cette section nous les passons en revue.

1.4.2.1 Problématiques de sécurité du SaaS

Comme nous l'avons précédemment décrit, le modèle SaaS représente un modèle de déploiement de logiciels où ces derniers sont hébergés chez un fournisseur de service distant. Grâce à sa prise en charge quasi-complête de l'infrastructure matérielle et logicielle, il est souvent considéré comme le modèle de déploiement le plus abouti pour répondre aux besoins des services IT des entreprises [12].

Cependant, une grande partie des entreprises reste réticente quant à la migration de ses logiciels dans une offre SaaS. Les problèmes de sécurité représentent leur principale cause du désintéressement pour le SaaS[22]. Plus précisément, c'est principalement le manque de transparence et de visibilité sur la façon dont les données sont stockées et sécurisées qui sont un frein à son adoption. En effet, dans le SaaS, un client dépend totalement de son fournisseur de services pour appliquer les mesures de sécurité appropriées aux données et applications mais aussi pour assurer une isolation complète des données des différents utilisateurs. Cette exigence forte nécessite une confiance mutuelle importante entre ces deux parties dans le cadre de la mise en oeuvre effective de ces politiques de sécurité. Toutefois, il s'avère actuellement difficile, voire impossible pour le client de veiller à ce que ces mesures de sécurité appropriées soient effectivement appliquées [35]. Par exemple, le fournisseur peut répliquer les données sur plusieurs serveurs dans différents pays pour assurer leurs forte disponibilité. En outre, il peut aussi avoir recours aux services d'autres

[21] www.linfo.org/vendor_lockin.html
[22] social.technet.microsoft.com

prestataires *cloud* afin de déléguer l'infrastructure support à leurs services. Dans ce cas, le contrôle et la sécurité des données de l'utilisateur ne dépendent pas seulement de son fournisseur de services mais d'une tierce partie, dont le client n'a pas connaissance de l'existence dans la plupart des cas.

1.4.2.2 Problématiques de sécurité du PaaS

Dans le modèle PaaS, le fournisseur de services offre aux utilisateurs un certain contrôle de l'environnement de travail afin de développer leurs applications. Dans ce contexte, toute la sécurité sous-jacente au niveau application tel que la détection et la prévention contre les attaques et intrusions réseaux et systèmes, reste dans le périmètre de responsabilité du fournisseur de services.

Contrairement aux environnements classiques de développement de logiciel, tels que les IDE (Eclipse, Microsoft Visual Studio), le modèle PaaS offre un environnement de développement partagé. Par conséquent, l'authentification, le contrôle d'accès et les mécanismes d'autorisation doivent être combinés pour assurer l'identification des clients ainsi que leur isolation. Ainsi, une infrastructure d'authentification et d'autorisation forte est essentielle pour garantir que seuls les utilisateurs individuels autorisés puissent accéder aux services offerts. Les attaques ciblants les systèmes d'authentification PaaS sont similaires à celles rencontrées dans des environnements standard et comprennent : l'emprunt d'identité, les attaques de *phishing* et *social engineering*, les attaques par force brute et l'attaque par réinitialisation de mot de passe. En revanche, au détriment de la complexité des mécanismes de sécurité cités ci-dessus, la plupart des fournisseurs PaaS utilisent encore des systèmes d'authentification faibles (basés sur un nom d'utilisateur et un mots de passe) pour l'authentification et la mise en place d'un mécanisme de contrôle d'accès aux données et aux applications [12].

Les services du modèle PaaS sont souvent sollicités afin de déployer des bus intergiciels[23] [36]. Leur premier but est d'assurer la connectivité entre les services et les applications qui n'ont pas été conçues pour fonctionner ensemble. Les applications suffisamment complexes pour construire un ESB, doivent garantir sa sécurité en s'appuyant sur un protocole tel que Web Service Security (WS-Security, WSS) [37] [12] [38].

1.4.2.3 Problématiques de sécurité du IaaS

Dans le modèle IaaS les problématiques de sécurité sont différentes de celles des modèles PaaS et SaaS. En effet, de par la nature même du service offert et selon le modèle de déploiement (public ou privé), les problèmes de sécurité induits par le modèle IaaS diffèrent légèrement. Avec un *cloud* privé, l'utilisateur a le contrôle total de l'infrastructure depuis le matériel déployé jusqu'aux couches applicatives. En revanche, avec un *cloud* public, l'utilisateur a le contrôle des machines virtuelles et des services en cours d'exécution sur ces dernières, mais ne contrôle pas les cycles de processeur, le réseau et le stockage

[23] Enterprise Service Bus (ESB)

de l'infrastructure sous-jacente. Selon Shinder[24] [25], de nombreux problèmes de sécurité, dont souffrent les réseaux classiques ainsi que les problèmes liés aux technologies de virtualisation sont présents dans le contexte du *cloud computing*tels que ceux liés à : (1) la protection contre les fuites de données et surveillance de l'utilisation des ressources ; (2) l'authentification et l'autorisation ; (3) la réaction aux incidents et capacités de *forensics* ; (5) le chiffrement de bout en bout ; (6) la gestion de la *Multitenancy* ; (7) la sécurité des hyperviseurs et des VMs ; et (8) la sécurité des images déployées.

Outre ces problèmes connus, certains liés spécifiquement à la technologie du *cloud* sont apparus, le plus connu étant le *VM leakeage* qui a pour objectif d'accéder à une VM d'un client et de récupérer ses données. En effet, la plupart des vulnérabilités liées à la virtualisation reposent sur un accès administrateur ou sur des erreurs de paramétrage. Cependant, Ristenpart *et al.* [39] ont révélés des vulnérabilités liées aux technologies de virtualisation déployées par les fournisseurs de services *cloud*, notamment Amazon EC2, et qui ne nécessitent ni un accès administrateur ni une erreur de paramétrage. Ces vulnérabilités élargissent ainsi la surface d'attaque des clients et offrent aux attaquants des possibilités inédites, grâce à l'exploitation de canaux latéraux liés à la co-résidence sur la même machine physique de la VM cible. Il ont également démontré que les technologies utilisées permettent la réalisation d'une cartographie de l'architecture de l'infrastructure sous-jacente. Ainsi, un attaquant est capable de déterminer la machine physique sur laquelle la VM cible est allouée. Il peut ensuite, en utilisant judicieusement certains paramètres d'initialisation, d'instancier autant de nouvelles VMs que nécessaire, de façon à ce que l'une d'elles ait un minimum de chances d'être co-résidente avec la cible et partage ainsi ses même ressources, ce qui lui ouvre une voie d'accès.

1.4.3 Utilisation malicieuse du *cloud computing*

Comme les sections précédentes l'ont mis en évidence, les solutions techniques relatives à la sécurité dans le *cloud* se concentrent principalement sur la protection des utilisateurs légitimes des services du *cloud* contre les attaques et intrusion externes ainsi que la sécurité des données en termes de confidentialité, intégrité et disponibilité [40] [41] [42] [43] [44] [45]. En revanche, très peu d'attention est accordée à la possibilité d'utiliser le *cloud* comme vecteur d'attaques. En effet, grâce à sa facilité d'utilisation, sa fiabilité et son évolutivité, le *cloud computing* offre une plateforme d'attaque prête à l'usage, pour les utilisateurs malveillants, et les plus grands bénéficiaires de cette reconversion sont les *botnets*.

Pour illustrer cette utilisation détournée de ces ressources, nous prenons le cas de l'étude menée par la société Stratsec [8] qui a eu pour objectif d'étudier les potentiels avantages d'une infrastructure de services *cloud* pour un tiers malveillant. Pour ce faire, une campagne expérimentale faite auprès de cinq fournisseur de services *cloud* les plus connus, mais dont le nom n'a pas été dévoilé, a été menée. En leur sein, le déploiement d'un *botcloud* (*botnet* dans le *cloud*) a eu lieu. Du côté de la victime, simulée par la

[24] blogs.technet.com/b/privatecloud/archive/2011/10/12/security-considerations-for-infrastructure-as-a-service-iaas-private-cloud.aspx

[25] windowsecurity.com/articles-tutorials/Cloud_computing/Security-Considerations-Infrastructure-Service-Cloud-Computing-Model.html

société qui a mené cette étude, des machines virtuelles ont été déployées dans un environnement contrôlé, qui contiennent des installations de services Web, FTP et SMTP. Chaque *botcloud* a été utilisé pour lancer les attaques de Déni de Service Distribué (DDoS), installation de Malwares, *Cross-site-scripting*, injection SQL, lancement de Shellcode, trafic non conforme aux RFC et attaques par force brute. Les expérimentations ont duré 21 jours et, pour déterminer si les durées de l'attaques impactent leur détection, les attaques DDoS ont duré 48 heure sans arrêt. Le résultat obtenu à l'issue de cette campagne de tests longue et explicite est le suivant. Aucune réaction n'a été rencontrée et ce pour chacun des différents cas de tests : les connections entrantes ou sortantes n'ont pas été interrompues ou réinitialisées ; aucun trafic n'a été bridé ou limité de quelque sorte ; aucun e-mail ou appel téléphonique n'a été reçu de la part des fournisseur de services et finalement aucune suspension temporaire ou définitive de leur souscription n'a été réalisée.

Dans la même idée, une autre étude [9] a démontré comment le *cloud* pourrait être exploité pour créer un large réseau de bots, en quelques minutes et avec un effort minimum. Cette étude fut fondée sur l'exploitation des services d'Amazon EC2 et a démontré cette faisabilité par la mise en oeuvre d'attaque DDoS et fraudes aux clics. Par ailleurs, des initiatives individuelles telles que celle de Thomas Roth [46] ont démontré comment casser des mots de passe WPA de réseaux Wifi en un temps record, c'est à dire moins de six minutes, et pour moins de 6$ à l'aide d'une attaque par force brute en utilisant ici aussi les services d'Amazon EC2. Le succès avéré de ces expérimentations atteste que la simplicité, la robustesse, la flexibilité et le faible coût des services *cloud*, représente une solution prête à l'emploi à quiconque souhaiterait mettre en oeuvre des services malveillants.

En outre, dans le cas des *botcloud*, il apparait que ceux-ci présentent plus d'avantages en comparaison avec les *botnets* classiques. En effet, un *botnet* traditionnel nécessite un temps de construction important, que l'on dimensionne en nombre de mois, pour prendre le contrôle de machines à l'insu de leurs usagers. A l'inverse, un *botcloud* peut être fonctionnel en quelques minutes. Ensuite, si un *botnet* n'est pas toujours fiable en raison du comportement du détenteur de la machine qui l'héberge (comme par exemple en raison de son extinction), un *botcloud* est toujours en ligne et prêt à l'usage. Enfin, si un *botnet* ne peut pas utiliser pleinement les ressources du processeur ou de la bande passante de sa machine support pour des raisons de furtivité, un *botcloud* peut être pleinement utilisé sans crainte d'interruption.

Au delà de ces considérations de mise en oeuvre, il existe en outre un marché pour l'utilisation de ce type de services frauduleux. Une étude de Verisign [9] a estimé le prix de location d'un *botnet* entre 9$ de l'heure et 67$ le jour. En revanche, bénéficier de ce genre de service est extrêmement difficile, car il nécessite un accès à des ressources cachées et à caractère illégal. Cependant, malgré le coût lié à la création d'un *botcloud*, cette opération se fait de façon tout à fait légale et accessible à tout utilisateur. De surcroit, le vol d'identité et de cartes de crédit permet aux personnes malicieuses de résoudre le problème du coût associé, tout en masquant leur identité.

1.5 Conclusion

Dans ce premier chapitre, nous avons introduit le paradigme du *cloud computing* et détaillé ses différentes caractéristiques et modèles. Nous avons ensuite présenté la technologie de la virtualisation, qui, parmi l'ensemble des technologies constitutives des infrastruc-

tures *cloud*, est celle qui apparait comme la plus importante pour l'adoption du *cloud*. Afin d'illustrer, les points précédents, nous avons décrit et étudié quelques plateformes de gestion qui permettent la mise en oeuvre de telles infrastructures.

Dans la seconde partie de ce chapitre nous avons abordé les problématiques de sécurité qui menacent le *cloud*, en étudiant en premier lieu les problèmes, attaques et intrusions qui le ciblent. En second lieu, nous avons abordé le cas particulier, qui constitue le cadre du travail présenté dans ce manuscrit, et qui concerne les attaques et intrusions qui tirent profit des avantages et de la puissance du *cloud* à des fins malveillantes, le transformant ainsi en un vecteur pour toute attaque ciblant un tiers connecté à l'Internet.

Afin de comprendre si les solutions actuelles pour la sécurité des infrastructures réseaux et du *cloud computing* permettent de répondre à cet usage détourné, nous présentons dans le chapitre suivant l'état de l'art de ces techniques.

2

Botnets, attaques DDoS et approches de détection

Sommaire

2.1 Introduction

Un *botnet* est un réseau de machines contrôlées et utilisées par un utilisateur malveillant, à l'insu de leurs utilisateurs légitimes, pour mener diverses activités illégales. Les *botnets* sont principalement utilisés pour perpétrer des attaques de déni de service qui peuvent engendrer des dommages aux conséquences désastreuses. Les outils traditionnels tels que les pare-feu ne sont plus en mesure de garantir un bon niveau de sécurité. Pour résoudre ce problème, les systèmes de détection d'intrusion constituent la meilleure ligne de défense. Dans ce cadre, nous présenterons dans ce chapitre un état de l'art sur : (1) les *botnets*, leur architectures et leur méthodes de détection ; (2) les attaques DDoS, leur

architectures et différents types ; et (3) les IDS, en mettant l'accent sur les IDS collaboratifs.

2.2 Les botnets

Les *botnets* sont considérés comme étant la menace la plus importante pour la sécurité et la stabilité d'Internet [47] [48] [49]. Ces réseaux sont créés afin de pouvoir mener des activités illégales à très grande échelle, telles que les attaques DDoS, fraudes au clic, envoi massif de courriers indésirables et vol d'identité.

2.2.1 Définitions et dommages potentiels

Durant la dernière décennie, les *botnets* ont été largement étudiés. Nous rappelons ici les définitions qui sont les plus reconnues. Dans leur état de l'art, Silva *et al.* [47] définissent les *botnets* comme suit :

Definition 7 *"Botnets are networks formed by "enslaving" host computers, called bots (derived from the word robot), that are controlled by one or more attackers, called botmasters, with the intention of performing malicious activities".*

Dans leur taxonomie des *botnets*, Khattak *et al.* [48] les définissent comme :

Definition 8 *"A botnet is a collection of compromised machines (bots) receiving and responding to commands from a server (the C&C server) that serves as a rendezvous mechanism for commands from a human controller (the botmaster)".*

Enfin, dans leurs travaux de caractérisation des *botnets*, Eslahi *et al.* [50] le définissent ainsi :

Definition 9 *"A bot, originating from the term 'robot', is an application that can perform and repeat a particular task faster than a human. When a large number of bots spread to several computers and connect to each other through the Internet, they form a group called a botnet, which is a network of bots."*

Les *botnets* peuvent infliger de graves dommages et être ainsi nuisibles à la sécurité globale de l'Internet. Selon [51] et [47] environ 80% du trafic de messagerie électronique est du courrier indésirable (*spam*) et la majeure partie de ces messages est générée par des *botnets*. Par ailleurs, en 2007, l'International Telecommunication Union [52] a estimé que les pertes causées par ce type de courriers reviennent à 100 milliards US$ dont 35 milliards US$ uniquement aux Etats Unis.

Au delà de cet usage voué à la génération de courriers indésirables, les *botnets* sont également la première source des attaques de déni de service distribuées. Ces dernières sont les responsables des plus importants dommages du secteur de l'IT et les pertes associées se comptent en millions de dollars par jour. Ces attaques sont présentées en détails dans la section 2.3. Avant cela, afin de montrer la dangerosité et les dommages qu'une attaque DDoS générée par des *botnets* peut causer, nous citons l'exemple de la cyber-attaque

contre l'Estonie [53] [54]. En 2007, ce pays a subit une attaque DDoS massive générée par plusieurs *botnets*. Cette dernière a visé les banques, les sites gouvernementaux et les chaines d'informations pendant trois semaines. Par conséquent, toute l'infrastructure réseau du pays s'est trouvée hors d'usage et les conséquences collatérales ont largement dépassé le seule cadre de l'Internet. L'accusé principal de cette attaque fût la Russie, du fait d'un désaccord politique avec l'Estonie[26] et suite à cela, les membres de l'Organisation du traité de l'Atlantique Nord (NATO) ont qualifié pour la première fois cet évènement de cyber-guerre[27] [54].

2.2.2 Architectures

Le mécanisme de commande et de contrôle représente le mécanisme et l'interface utilisés afin d'assurer le fonctionnement et les communications entre le *botmaster*, le serveur C&C et les *bots*. Selon l'architecture de ce mécanisme ainsi que les protocoles de communication utilisés, un *botnet* peut être classifié selon son degré de décentralisation. On trouve ainsi des architectures centralisées [49], décentralisés [55] et hybrides [56] [57] [58]. Dans cette section nous présentons chacune d'entre elles.

2.2.2.1 Architectures centralisées

L'approche centralisée est similaire au modèle client-serveur. Dans cette architecture, tous les *bots* établissent leur canal de communication avec un ou quelques points de connexion. Ces points représentent les serveurs C&C qui sont les responsables de la fourniture des commandes et des mises à jour des *bots*.

Internet Relay Chat (IRC) [59] et *HyperText Transfer Protocol* (HTTP) [60] représentent les principaux protocoles utilisés par les architectures centralisées. Dans le cas des *botnets* reposant sur le protocole IRC, le *botmaster* crée des canaux IRC sur les serveurs C&C sur lesquels les *bots* vont se connecter et attendre les commandes. IRC permet la communication à travers des groupes de multicast appelée "canaux de communication" ou à travers des communications unicast privées entre deux membres. Cette caractéristique permet au *botmaster* d'avoir un contrôle flexible sur son *botnet* [47].

Bien que le protocole IRC soit très flexible et bien adapté aux besoins du C&C, il souffre d'une facilité de détection qui peut causer l'interruption du fonctionnement du *botnet*. En effet, la détection du trafic IRC est aisée car son trafic est peu commun et rarement utilisé dans les réseaux d'entreprise, et il est d'ailleurs généralement bloqué. Ainsi, un administrateur peut empêcher l'activité d'un *botnet* IRC en bloquant le trafic IRC au niveau d'un pare-feu [58]. Face aux inconvénients du protocole IRC, le protocole HTTP a été proposé pour assurer les communications du C&C. Comparativement, son principal avantage réside dans la permissivité du trafic HTTP qui est autorisé dans la plupart des réseaux et qui permet de dissimuler les communications au sein du *botnet*.

L'avantage d'une approche centralisée réside dans l'absence d'élément intermédiaire entre le C&C et les *bots* qui apporte (1) une bonne coordination ; (2) un temps de réaction court ; et (3) un meilleur contrôle sur les *bots* et l'état général du *botnet* par le *botmaster*.

[26] http ://www.theguardian.com/world/2007/may/17/topstories3.russia
[27] http ://news.bbc.co.uk/2/hi/europe/6665145.stm

L'inconvénient majeur de ce type d'approche réside dans le serveur C&C lui même qui représente un point unique de défaillance et qui facilite la mise hors service de tout le *botnet* dès sa détection. A titre d'exemples de *botnets* centralisés, nous citons : Hybrid_V1.0 (HTTP), Zeus (HTTP), Festi (HTTP), Bobax (HTTP), Asprox (HTTP), Srizbi (HTTP), Kaiten (IRC), Agobot (IRC), Rxbot (IRC), SDbot (IRC) et EggDrop (IRC).

2.2.2.2 Architectures décentralisées

Les différentes faiblesses de l'architecture centralisée ont motivé la proposition de solutions décentralisées. Les *botnets* possédant une telle architecture sont plus difficiles à mettre hors d'état de nuire car même la découverte de nombreux *bots* ne signifie pas nécessairement la perte de l'ensemble du *botnet*, du fait de l'absence d'un serveur C&C central.

Les *botnets* décentralisés sont généralement basés sur une variété de protocoles pair à pair (P2P) et fonctionnent tel un réseau *overlay* qui selon [61] peuvent être classés en :

Overlay pair-à-pair structuré : Ce type de réseaux utilise souvent des Tables de Hachage Distribuées (DHT), telle que celles basées sur les protocoles CAN [62], Chord [63], Pastry [64] et Tapestry [65]. Ce type d'architecture permet une communication ciblée et optimale entre les différents noeuds du réseau.

Overlay pair-à-pair non-structuré : Ce type de réseaux fait référence aux topologies aléatoires avec différents degrés de distribution. Ainsi, ils n'offrent aucune possibilité de routage.

Overlay de super-pairs : Dans ce type de réseau, tous les pairs ne sont pas égaux, et un sous-ensemble de pairs est automatiquement sélectionné en tant que serveurs temporaires pour soutenir les fonctions réseau, telles que la recherche et le contrôle. Les réseaux de super-pairs sont plus visibles et plus vulnérables aux attaques ciblées. Ainsi, il est préférable que les *botnets* n'adoptent pas cette conception [47].

L'absence de point central de contrôle représente le principal avantage de ce type d'architecture. Cependant, il demeure vulnérable aux infiltrations. En effet, si un noeud (*bot*/serveur) est détecté, une grande partie des noeuds auxquels il est connecté peut être découverte. Comme exemple de *botnets* décentralisés, nous citons : Conficker, TDL-4, Waledac, Phatbot, SpamThru, Nugache et Peacomm.

2.2.2.3 Architectures hybrides

Les architectures hybrides combinent les caractéristiques des *botnets* centralisés et décentralisés. Selon [56], les *bots* d'un *botnet* hybride peuvent être classifiés en deux groupes : les *bots servants* et les *bots* clients. Les *bots servant* se comportent comme des serveurs et comme des clients. Ils sont configurés avec des adresses IP statiques et routables. Les *bots* clients sont configurés avec des adresses IP dynamiquement désignées ou non routables et n'acceptent aucune connexion entrante. En effet, dans cette architecture, les *bots servants* sont les seuls candidats qui peuvent avoir leurs adresses IP sur les listes de pairs. Afin de gérer les connexions entrantes, ils écoutent sur un port déterminé puis utilisent une clé de chiffrement symétrique auto-générée pour la communication, ce qui rend difficile la

détection du *botnet*. Tous les *bots* doivent périodiquement se connecter aux *servants* de leur liste de pairs afin de récupérer les commandes émises par le *botmaster*.

2.2.3 Méthodes de détection des Botnets

Compte tenu de la dangerosité et de la puissance des *botnets* pour mener différentes activités malveillantes, leur détection représente un enjeu essentiel. Dans ce but, différentes approches de détection ont été proposées. Ces dernières sont classées en deux familles principales : celles fondées sur la mise en place de réseaux de pots de miel *(honeynets* et *honeypots)* [66] et celles basées sur des systèmes de détection d'intrusions. Selon Khattak *et al.* [48], ces méthodes de détection se composent de trois catégories : (1) détection de bots ; (2) détection de canaux C&C ; (3) détection du *botmaster*.

Un pot de miel représente une entité logicielle ou matérielle imitant le fonctionnement d'un programme informatique ou d'une machine physique. Les réseaux de pots de miel représentent des éléments indispensables pour la détection et l'analyse des nouvelles menaces de sécurité des réseaux et ils sont les mieux adaptés pour assurer la collecte des informations sur les *bots*. Ces informations sont ensuite utilisées dans la compréhension et l'analyse des caractéristiques principales des *botnets*. Toutefois, les réseaux de pots de miel souffrent aussi d'importantes limites, comme la détection au seul périmètre des machines sur lesquelles ils sont implémentés.

Concernant les IDS, à petite échelle, ces solutions se basent généralement sur les signatures du trafic généré par les *botnets* pour identifier les communications malveillantes. Par ailleurs, au niveau d'un opérateur réseau, le système de détection a la possibilité d'analyser et de corréler le trafic de tous les utilisateurs afin d'identifier les *bots* participant aux mêmes activités malveillantes. Le problème des IDS déployés au niveau d'un opérateur et à grande échelle nécessite la prise en considération de différentes contraintes qui concernent les aspects techniques et législatifs. Nous citons à titre d'exemple : l'anonymisation du trafic, l'analyse en temps réel d'une quantité extrêmement grande de données et le changement continu de l'architecture réseau des *botnets*.

Plusieurs méthodes de détection de *botnets* ont été proposées ces dernières années. Une méthode générique de détection de *botnets* est basée sur l'identification du canal de communication C&C des machines infectées avec leur serveur de contrôle. Pour s'y connecter, les *bots* ont besoin de l'adresse IP du serveur. Afin de récupérer cette adresse, les machines infectées utilisent le protocole DNS. Selon le protocole DNS, cette association entre l'adresse IP et le nom du serveur reste actif tant que le TTL [28] n'est pas nul. La méthode générique de la détection de *bots* consiste à bloquer les noms de domaines malveillants. Le filtrage de ces noms de domaines devient toutefois de plus en plus compliquée avec l'utilisation de la technique appelée *domain-flux* [67]. A titre d'exemple, pour bloquer le *botnet* Conficker, il aurait fallu bloquer des dizaines de milliers de noms de domaine par jour, ce qui s'avère impossible en pratique.

Les *botmasters* utilisent souvent des algorithmes de génération de domaines (DGA : Domain Generation Algorithms) pour produire dynamiquement un grand nombre de noms de domaines au hasard et sélectionner ensuite un sous-ensemble de ces domaines. Anto-

[28] *Time To Live* – Un champ dans la réponse du serveur DNS

nakakis *et al.* [68] ont utilisé les propriétés statistiques de ces algorithmes pour grouper les noms de domaine d'un même *botnet*. Ils ont aussi utilisé les chaines de Markov cachées pour identifier les noms de domaine du serveur C&C.

Wurzinger *et al.* [69] proposent un outil de génération de signatures de trafic malveillant qui pourrait être utilisé ultérieurement par un IDS pour détecter les communications entre les machines infectées et le serveur de contrôle. L'idée consiste à mettre en ligne un réseau de pots de miel afin d'attirer l'opérateur du *bot*. Ces signatures sont ensuite générées en observant la communication des machines infectées du réseau de pot de miel avec leur serveur de contrôle après avoir reçu les commandes du serveur. Malheureusement, cette technique ne peut pas générer une signature si la communication avec le serveur de contrôle est chiffrée.

Guofei *et al.* [70] analysent le trafic généré par un réseau local et l'intègre dans un système de détection d'intrusion. Pour évaluer le comportement d'une machine, ce système appelé SCADE (*Statistical Scan Anomaly Detection Systems*) se base sur une série d'évènements comme le scan de ports de l'extérieur vers l'intérieur et inversement, le téléchargement d'un fichier binaire infecté et la communication avec le canal de contrôle. Chaque évènement est pondéré, et ensuite, en combinant tous ces évènements, SCADE génère une alerte d'infection. SCADE est efficace pour la détection de certains *botnets* connus comme Agobot, Rxbot et SxBot. Malheureusement il n'est pas efficace dans le cas d'utilisation de certaines méthodes de scan et de chiffrement de la communication entre la machine infectée et le serveur de contrôle.

Ces systèmes et méthodes de détection se basent sur les signatures pour détecter les *botnets*. Par conséquent, ils identifient uniquement les *botnets* qui possèdent une signature connue dans leur base mais sont inéfficaces face aux nouveaux *botnets*. Un autre inconvénient de ces systèmes est qu'une légère modification dans les communications entre les machines infectées et leur serveur empêchera ces systèmes de détecter les *bots*. En outre, le passage à l'échelle d'un réseau d'opérateur de ces approches n'est pas toujours possible. Pour détecter les *botnets* à large échelle, Karasidis *et al.* [71] proposent une méthode qui isole les machines qui génèrent du trafic malicieux (spam, DDoS, etc.). Ensuite, ils corrèlent les connexions de ces machines afin de détecter les membres du même *botnet* ainsi que leur serveur C&C.

Dans ce contexte, la détection des *botnets* qui reposent sur les réseaux P2P est confrontée à plusieurs défis. En effet, le partage de fichiers P2P par certains logiciels comme BitTorrent ou eMule permet aux opérateurs de *bots* de fondre leur trafic malicieux dans de trafic P2P. Quelques approches capables de détecter les *botnets* P2P ont été proposé dans [72] [68] et [48]. Récemment, Zhang *et al.* [73] ont proposé une nouvelle approche basée sur les méthodes de classification pour identifier les machines qui communiquent en P2P avec leur serveur de contrôle. L'approche consiste à récupérer les empreintes digitales statistiques des communications P2P et ensuite distinguer les communications P2P légitimes de celle qui sont illégitimes. Ce système a été évalué sur la base du trafic réseau et a montré une précision élevée avec 0,2 % de taux de faux positif et une grande évolutivité (traitement de 80 millions de flux en 48 minutes).

2.3 Les attaques de déni de service distribuées

Parmi les différentes formes d'activités malveillantes mises en oeuvre par les *botnets*, les attaques de déni de service sont parmi celles qui sont les plus perpétrées. Dans cette section nous présenterons en détail ces formes d'attaques d'un point de vue fonctionnel et architectural.

On rappelle avant tout que le déni de service (DoS) représente une attaque où une victime reçoit un flux de paquets malveillants qui épuise une ressource clé. Cet épuisement se traduit par le déni de ce service (la ressource) aux clients légitimes de la victime. Un attaquant peut réaliser une attaque DoS et épuiser une ressource clé de deux manières : (1) en exploitant certaines vulnérabilités dans les protocoles ou logiciels utilisés par la victime (attaques de vulnérabilité), comme par exemple dans le cas de l'attaque historique *Ping of Death* ; ou (2) par le simple envoi d'un volume de trafic plus élevé que celui que les ressources de la victime peuvent gérer ; on parle alors d'attaque par inondation (*flooding attacks*) dont un exemple est *UDP Flood*. Afin de réaliser une attaque de vulnérabilité, l'attaquant envoie généralement des paquets d'un type ou d'un contenu spécial. Comme les vulnérabilités peuvent souvent être exploitées par quelques paquets, les attaques de vulnérabilité ont un faible volume de trafic. Ces deux caractéristiques (paquets de type spécial et faible volume) permettent de simplifier le traitement de ce type d'attaque, par l'utilisation de correctifs logiciels ou par la simple détection de ces paquets spéciaux.

2.3.1 Définitions et caractéristiques

D'après Mirkovic *et al.* [74], le déni de service distribué (DDoS) est défini par :

Definition 10 *"An overwhelming quantity of packets being sent from multiple attack sites to a victim site. These packets arrive in such a high quantity that some key resource at the victim (bandwidth, buffers, CPU time to compute responses) is quickly exhausted."*

et selon Prolexic [75] :

Definition 11 *"DDoS attacks are attempts to make a computer resource (i.e. website, e-mail, VoIP, or a whole network) unavailable to its intended users. Overwhelmed with massive amounts of unsolicited data and/or requests, the target system either responds so slowly as to be unusable or crashes completely. The data volumes required to do this are typically achieved by a network of remotely controlled zombie or botnet (robot network) computers."*

Les attaques DDoS sont des attaques DoS effectuées à partir de plusieurs machines détournées *(bots)*, contrôlées par un attaquant. Par conséquent, la machine cible est inondée par du trafic réseau. En général, dans le scénario le plus fréquemment utilisé, toutes les machines sont engagées simultanément et commencent à générer autant de paquets que possible envers la victime. L'attaquant peut également contrôler le volume de trafic généré par chaque machine et c'est la participation d'un grand nombre de machines (*bots*), même si ces dernières ne disposent que de capacités modestes, qui permet la surcharge rapide des victimes.

Au delà de ces définitions fonctionnelles, les attaques DDoS présentent les caractéristiques suivantes :

Usurpation d'adresse IP : Généralement, les attaquants utilisent une technique d'usurpation des adresses IP source *(IP spoofing)* augmentant ainsi la difficulté à tracer les *bots* et rendant possible leur ré-utilisation pour d'autres attaques. En outre, bien que le traçage des *bots* puisse avoir lieu, ce dernier ne permet pas de tracer l'attaquant. Enfin, la grande variété des adresses des paquets envoyés complique la tâche des systèmes de défense. Outre cette caractéristique de dissimulation de l'attaquant et de ses *bots*, l'usurpation d'adresse permet aussi de mettre en oeuvre des attaques réflectives *(reflector attack)* [76], simples ou distribuées. Le principe de ce type d'attaque consiste pour un attaquant à envoyer des requêtes à un ou plusieurs serveurs sur Internet en remplaçant son adresse source par celle de la victime. Par conséquent, c'est cette dernière qui recevra les réponses des serveurs interrogés. Dans ce cas, l'attaquant ne dispose pas uniquement d'un large *botnet* de machines détournées, mais il maintient également une liste de réflecteurs. Un réflecteur représente n'importe quelle adresse de serveur en mesure de retourner une réponse à une requête comme par exemple un serveur web, un serveur DNS ou un serveur NTP.

Taille du *botnet* : Il existe de nombreuses techniques performantes pour tracer des machines attaquantes. Cependant, bien que les *bots* puissent être tracés, la question des actions qui pourraient être prises contre des milliers voir des centaines de milliers de machines reste ouverte. En outre, dans le cas d'une attaque *flooding* générée par un grand nombre de *bots*, bien que les systèmes de défense puissent bloquer ou dévier le trafic de l'attaque, c'est le chemin vers la victime et son point d'entrée réseau qui seront complètement submergés. Afin d'éviter de telles situations, les mécanismes de défense doivent détecter et bloquer ce trafic malveillant au plus proche de sa source [74] [77].

Similitude entre le trafic d'attaque et le trafic légitime : Tout type de trafic peut être utilisé afin d'effectuer un déni de service mais certains types de trafic nécessitent plus de volume que d'autres pour atteindre cet objectif. Ainsi, une attaque de déni de service peut être dissimulée dans une quantité conséquente de trafic qui contient des paquets légitimes rendant difficile la différentiation entre une attaque et un phénomène légitime de pic de charge (*flash crowd*). Ceci explique la difficulté à réaliser un système de défense contre les DDoS qui repose sur le contrôle des paquets.

2.3.2 Typologie des attaques DDoS

Il existe de multiples classifications pour les attaques DDoS [5], [78], [79], [80]. Parmi celles-ci, [5] propose une version consensuelle, en accord avec les autres classifications existantes, dont une version détaillée par sous-classe est donnée dans la figure 2.1, comme fondement de sa taxonomie. Si cette classification permet de catégoriser de façon fine et exhaustive l'ensemble des attaques existantes, dans le cadre de notre étude et dans le seul but de recencer les différentes formes d'attaques existantes, nous considérons une

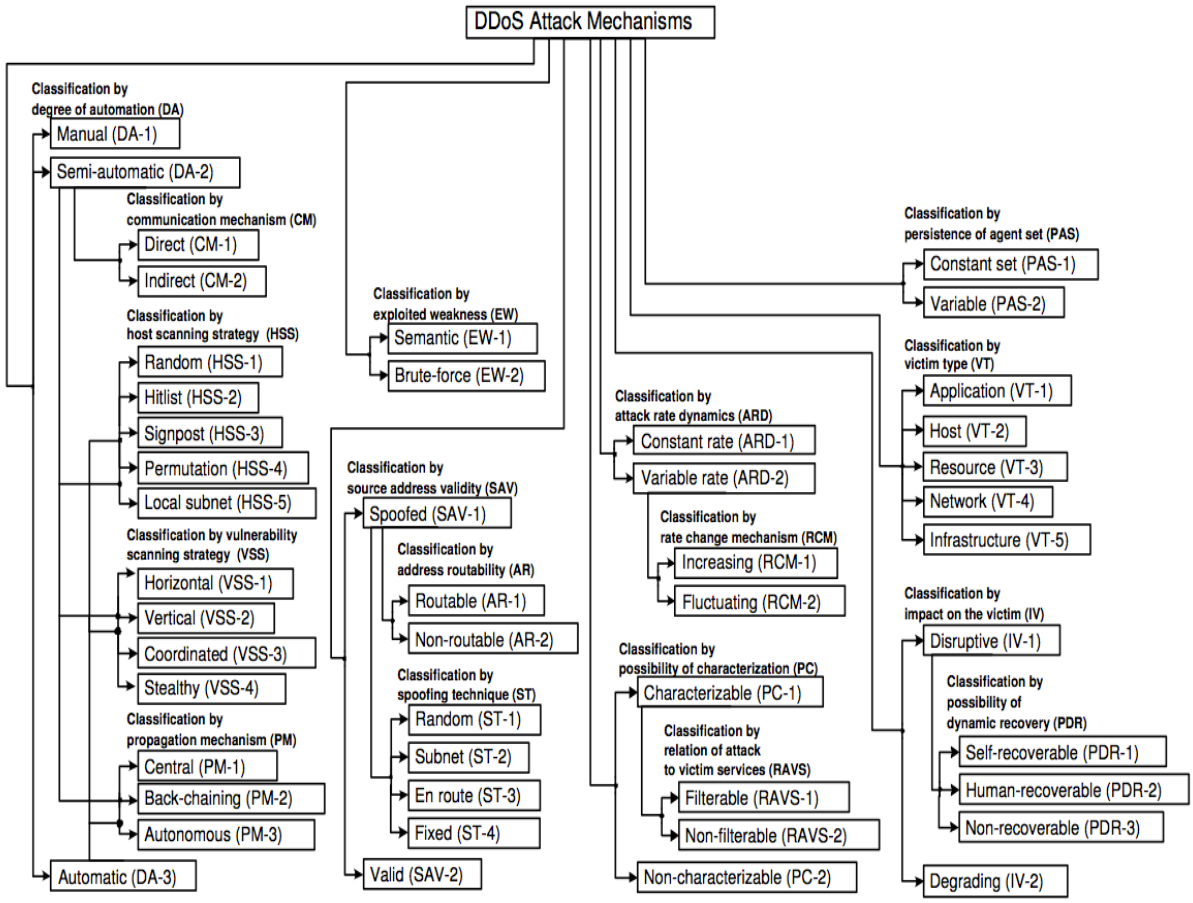


FIGURE 2.1 – Taxonomie des attaque DDoS selon [5]

classification proposée par [81] et [6] qui classifient les attaques selon la couche protocolaire qu'elles ciblent, à savoir au niveau de l'infrastructure réseau ou de la couche application.

Pour illustrer son usage, nous indiquons les résultats d'une enquête menée par *Arbor networks* [81] qui a consisté à classifier les attaques subies par chacun des participants. Ces résultats sont basés sur des données collectées via ATLAS *(Active Threat Level Analysis System)*[29]. Les résultats produits montrent l'usage majeur d'attaques au niveau infrastructure (dont 61% d'attaques volumétriques et 20% d'attaques par épuisement d'état) a contrario des attaques sur la couche application qui ne comptent que pour 24%. Ces résultats sont par ailleurs confirmées par l'étude menée par *Akamai Technologies* [6] et dont un extrait est représenté sur la figure 2.2.b décrivant l'ensemble des attaques détectées tout au long du troisième trimestre de l'année 2014. On y constate que les attaques sur la couche infrastructure totalisent plus de 89%.

2.3.2.1 Attaques sur la couche infrastructure

Cette classe comprend l'ensemble des attaques qui ciblent les services des couches de niveau 3 et 4 du modèle OSI. Elle se divise en deux sous classes :

[29] ATLAS est le premier et le plus grand réseau d'analyse de menaces à l'échelle mondiale. Lancé en

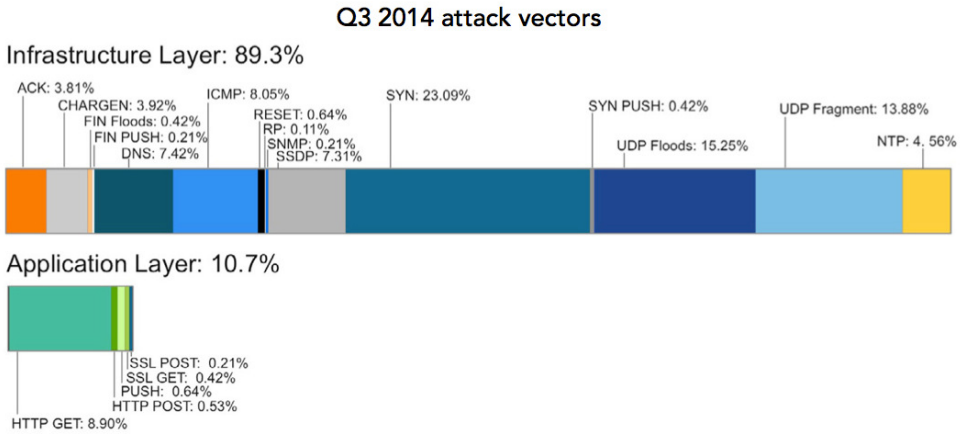


FIGURE 2.2 – Répartition des attaques perpétrées en 2014 selon leur niveau de couche protocolaire par Akamai Technologies [6]

Attaques volumétriques : Ces attaques ont pour but la consommation de la bande passante provoquant la congestion de la cible. Cette dernière peut être un réseau, un simple service ou le lien entre un réseau ou un service avec le reste d'Internet [81].

Attaque par épuisement d'état : Ces attaques tentent de consommer les tables d'état de connexions présentes dans de nombreux composants d'une infrastructure, comme les équilibreurs de charge, les pare-feu, les serveurs d'applications et même les IPS (*Intrusion Protection System*). Ils peuvent assaillir des appareils de grande capacité, capable de maintenir l'état de millions de connexions. Ces attaques sont souvent combinées avec des attaques de type *flooding* afin d'amplifier leur impact [82].

Les attaques sur la couche infrastructure représentent de loin celles qui sont les plus perpétrées et les exemples de protocoles ciblés par ce type d'attaques sont les suivants :

UDP Flood : Durant cette attaque, la victime est submergée par du trafic UDP qui sature sa bande passante. Afin d'exploiter pleinement cette ressource, les paquets ont généralement une grande taille. La puissance de cette attaque réside dans sa simplicité. En effet, l'attaquant n'a pas besoin de découvrir ou d'exploiter une vulnérabilité pour la réaliser. Il lui suffit d'envoyer un grand nombre de paquets. Afin d'accroitre les effets de cette attaque, les attaquants utilisent en général un grand nombre de *bots* pour générer les paquets. L'attaque UDP Flood est l'une des attaques les plus utilisées. Durant ces trois dernières années ce type d'attaque a représenté environ 15% de l'ensemble des attaques répertoriées [81] [6] [75] et la plus importante attaque UDP Flood recensée en 2014 a atteint 321 Gb/s [6].

TCP SYN Flood : Cette attaque repose sur l'exploitation par l'attaquant d'une

2007, il bénéficie actuellement de la coopération de plus de 300 opérateurs de l'Internet. Ces derniers partagent leur données de trafic de façon anonyme. Ainsi, les sondes ATLAS traitent environ 90 Tb/s de trafic en continu ce qui représente un tiers du trafic Internet.

faille dans le protocole TCP. Cette dernière réside dans l'étape d'allocation des ressources par le serveur pour le client. En effet, après la seconde étape du *three-way handshake*, la connexion est dite semi-ouverte et consomme un certain nombre de ressources du côté du serveur comme de la mémoire et des cycles processeur. Ainsi, un attaquant peut supprimer la dernière étape et ne pas répondre avec le message ACK attendu. Le serveur attend alors un certain temps avant de libérer les ressources. En générant suffisamment de connexions semi-ouvertes, il est possible de surcharger les ressources du serveur, l'empêchant ainsi d'accepter de nouvelles requêtes, avec pour résultat un déni de service. Dans de cas rares, le serveur peut même nécessiter un redémarrage par manque de ressources. La plus grande attaque TCP SYN Flood recensée en 2014 a atteint 169 million de paquets par seconde [6].

ICMP Flood : Durant cette attaque, l'attaquant génère d'énormes flux de paquets *ICMP ECHO* visant la victime sans attendre de réponse. La victime essaie de répondre à chaque requête ICMP, consommant ses ressources CPU pour la génération des réponses et les ressources réseau. Cependant, si le flux de paquets *ICMP ECHO* est trop important, notamment quand il est généré par un grand nombre de *bots*, la victime n'est pas en mesure de générer autant de réponses que nécessaire, ce qui cause un déni de service. Cette attaque est aussi simple a réaliser que celle de UDP Flood et selon [75] [6], elle est aussi déployée que cette dernière.

2.3.2.2 Attaques sur la couche application

Cette classe comprend l'ensemble des attaques qui ciblent les services ou applications de la couche 7 du modèle OSI. Elles sont plus complexes que les attaques sur la couche infrastructure, furtives et plus sophistiquées, car elles peuvent être très efficaces avec très peu de machines attaquantes générant un faible trafic. En effet, ces attaques peuvent être structurées de manière à surcharger certains éléments spécifiques sur un serveur applicatif. Par exemple, des attaques ciblant les pages d'authentification avec des identifiants et mots de passes aléatoires, ou les fouilles aléatoires répétitives sur des sites web dynamiques, peuvent surcharger les bases de données et l'activité CPU des serveurs qui les hébergent.

Les exemples phares de protocoles de la couche application qui sont ciblés par ce type d'attaques sont les suivants :

HTTP Flood : Quand un client HTTP communique avec un serveur HTTP, il envoie des requêtes qui peuvent être de plusieurs types, les deux principaux étant GET et POST. Une requête GET est paramétrée par une URL[30] et sert généralement à récupérer un ou plusieurs objets HTML. Quant aux requêtes POST, elles sont utilisées pour soumettre des formulaires. Durant l'attaque, l'attaquant submerge le serveur cible avec de nombreuses demandes, de manière à saturer ses ressources. L'attaque fonctionne mieux lorsque le serveur alloue beaucoup de ressources en réponse à une demande unique. Etant donné que les requêtes POST incluent des paramètres, elles déclenchent habituellement un traitement relativement complexe sur le serveur, comme des accès à des bases de données, ce qui est beaucoup plus coûteux pour le serveur que la réponse à une requête GET. Ainsi, les

[30] Unified Resource Locator

attaques HTTP POST Flood ont tendance à être plus efficaces que les les HTTP GET Flood. Toutefois, étant donné que les requêtes GET sont beaucoup plus communes, il est souvent plus facile pour l'attaquant de les utiliser. Selon [75] et [6], durant ces trois dernières années, seulement environ 3% des attaques surveillées était des POST Flood contre environ 20 % de GET Flood.

DNS Reflection : Le DNS représente un service support essentiel au fonctionnement d'Internet. Il est principalement utilisé pour traduire les noms de domaine en adresses IP. Durant une attaque DNS reflection/amplification, l'attaquant envoie un flux de requêtes DNS à un ensemble de serveurs DNS ouverts (*open DNS resolvers*), tout en remplaçant l'adresse IP source des requêtes par celle de la victime. Une requête DNS demande généralement un grand jeu d'enregistrements, ce qui engendre une amplification du trafic de l'attaque. En utilisant plusieurs *bots* pour envoyer des requêtes à plusieurs serveurs DNS, un attaquant peut provoquer de très gros volumes de trafic d'attaque provenant de sources largement distribuées. L'autre facteur de puissance de ces attaques réside dans le grand nombre de serveurs DNS. En effet, actuellement, on estime qu'il existe plus de 32 millions serveurs DNS sur Internet, dont 28 millions qui représentent une menace potentielle[31].

Si les attaques de type DNS Reflection ont toujours été mises en oeuvre avec une puissance similaire aux autres attaques, en mars 2013, une attaque contre *Spamhaus*[32] a battu tout les records, devenant à cette date, la plus grande attaque DDoS de l'histoire avec un débit d'attaque agrégé de plus de 300 Gb/s. La puissance de cette attaque a perturbé l'ensemble du réseau Internet[33] et pour ce faire, elle a nécessité plus de 30 000 serveurs DNS [6].

NTP Reflection : NTP[34] est le protocole utilisé par les machines connectées à Internet pour régler avec précision leur horloge. Comme le DNS, NTP représente un outil idéal pour générer des attaques DDoS. Le protocole NTP utilise une commande appelée *monlist* qui peut être envoyée à un serveur NTP à des fins de surveillance. Ce dernier renvoie les adresses des 600 dernières machines avec lesquelles le serveur NTP a interagi, générant ainsi une réponse volumineuse. C'est cette dernière qui est utilisée pour amplifier les attaques. Ainsi, un attaquant, armé d'une liste de serveurs NTP sur Internet, peut facilement réaliser une attaque DDoS puissante, d'autant plus qu'il utilise un grand nombre de *bots*. En outre, les serveurs NTP ne sont pas difficiles à solliciter. En effet, les outils communs comme Metasploit et NMAP ont des modules capables d'identifier facilement les serveurs NTP qui supportent la commande *monlist*. Les attaques NTP Reflection tout comme les DNS Reflection ont toujours été connues mais leur succès avéré date de février

[31] openresolverproject.org

[32] *The Spamhaus Project* est une organisation internationale non gouvernementale. Son objectif est de fournir une protection en temps réel contre le Spams aux réseaux opérant sur Internet et de travailler en coopération avec les autorités en charge de la répression contre le crime organisé pour identifier et engager des poursuites envers les bandes de sapeurs et de logiciels malveillants dans le monde entier.

[33] https ://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet/

[34] Network Time Protocol

2014, date où une attaque contre *CloudFlare*[35] a atteint plus de 400 Gb/s dépassant de plus de 100 Gb/s celle contre *Spamhaus*. Depuis, ce type d'attaque connait une large adoption.

2.4 Les systèmes de détection d'intrusions

Les Systèmes de Détection d'Intrusion (IDS) représentent des mécanismes logiciels et/ou matériels conçus pour surveiller les activités d'un réseau ou d'un ordinateur et émettre des alertes aux administrateurs lorsque d'éventuelles intrusions ou activités suspectes sont détectées.

2.4.1 Typologie des IDS

Dans cette section nous décrivons les différents types d'IDS en fonction de trois critères de classification des approches. Tout d'abord, selon la nature des algorithmes de détection qu'il met en oeuvre, nous classifions les IDS selon une approche qui peut être par scénario et/ou comportementale. Ensuite, la source de données qu'il considère le font appartenir à la catégorie des NIDS *(Network based IDS)*, s'il considère des données issues du réseau, ou HIDS *(Host based IDS)*, s'il considère des données issues du système des machines surveillées. Enfin, son emplacement, à la source des attaques, à la cible ou sur tout élément appartenant au chemin entre ces deux entités, est le dernier critère que nous retenons pour classifier ces systèmes. Les IDS peuvent être utilisés pour détecter un large panel d'activités malveillantes et dans la suite nous considérons seulement les systèmes de défense contre les attaques DDoS.

2.4.1.1 Classification par approche

En général, la détection d'intrusion repose sur deux méthodes à savoir l'approche par scénario et l'approche comportementale.

Approche par scénario : Selon [83], l'approche par scénario (*misuse detection*) définit quelles sont les actions utilisateurs spécifiques qui constituent un abus et utilise des règles définies à priori pour encoder et détecter des formes d'intrusions connues. La technique de comparaison la plus répandue est celle des algorithmes de recherche de ces motifs *(pattern matching)* dans les paquets d'informations transitant sur le réseau ou présents dans les fichiers de journalisation. Cependant, d'autres approches existent, comme celle proposée par Mé [84] et qui est basée sur des algorithmes génétiques pour l'analyse des audits systèmes, ou comme celle de Lunt *et al.* [85] qui s'est intéressée aux systèmes experts.

Cette technique repose sur la définition préalable de motifs caractérisant des attaques connues et insérées dans une base de données. Aussi, dans ce contexte, tout ce qui n'est pas interdit est autorisé. Ainsi, cette approche détecte seulement les attaques et les intrusions connues. De ce fait, elle nécessite des mises à jour fréquentes et son efficacité dépend fortement de la précision de sa base de signatures.

[35] https ://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack/

Par conséquent, ce type de système est en mesure d'atteindre de hauts niveaux de précision avec un faible taux de faux positifs. Par contre, il est incapable de détecter des attaques non enregistrées dans la base de signatures et même les variations d'attaques connues.

Approche comportementale : L'analyse comportementale *(anomaly detection)* repose sur la définition du comportement normal de l'élément surveillé (système, application ou réseau) [86]. Elle nécessite donc une phase d'apprentissage qui va permettre de définir le modèle de comportement normal. Dans cette approche toute déviation significative du comportement normal définie par un profil établi génère une alerte car signifiant un comportement anormal. Le modèle du comportement normal est d'une référence collectée par observation ou spécifiée à priori. La difficulté de cette approche réside dans l'établissement d'un profil de normalité. En revanche, son principal avantage réside dans la détection des attaques sans connaissance préalable de ces dernières, ce qui lui permet de détecter des attaques inconnues.

2.4.1.2 Classification par source de données

En général, la détection d'intrusion repose sur deux types de données à savoir les données réseaux et les données récupérées au niveau des hôtes.

IDS réseau : Un IDS réseau (*Network Based IDS – NIDS*) surveille le trafic sortant et entrant sur le réseau protégé en examinant le trafic paquet par paquet en temps réel. Les NIDS sont les équipements les plus connus et les plus répandus. Leur objectif est de surveiller les flux d'informations transitant en un point du réseau pour détecter tout ce qui pourrait présenter un caractère non conforme. Si le principe de détection est simple, il pose cependant des problème lors de l'utilisation de flux chiffrés et lorsque les débits des réseaux augmentent. Les exemple de NIDS les plus usités actuellement sont Snort[36], Bro[37] et Suricata[38].

IDS hôte : Un IDS hôte (*Host based IDS – HIDS*) est implémenté au niveau d'un hôte ou d'un dispositif dans le réseau. Il surveille l'activité système tel que le taux d'utilisation du processeur et de la mémoire de l'hôte protégé, ainsi que son trafic entrant et sortant. Dans un environnement virtualisé, un HIDS peut être implémenté au niveau du système hôte de la machine physique, à l'intérieur des VM instantiées ou au niveau de la couche de virtualisation. Chaque niveau induit un accès aux données de surveillance qui sont étroitement liées à leur emplacement dans cette architecture et sert des objectifs de surveillance qui sont variés. Les exemples phares de HIDS sont OSSEC[39] et Tripwire[40].

[36] snort.org
[37] bro.org
[38] suricata-ids.org
[39] ossec.net
[40] tripwire.com

2.4.1.3 Classification par emplacement

Les flux d'une attaque DDoS proviennent de machines distribuées et situées à différents emplacements de l'Internet. Ces flux convergent vers leur cible par leur routage au travers de différents réseaux et systèmes autonomes. Cet acheminement implique ainsi de trois types de réseaux : (1) les réseaux sources qui hébergent les machines attaquantes ; (2) plusieurs réseaux intermédiaires qui acheminent le trafic d'attaque vers la victime ; et (3) le réseau victime qui héberge la cible [74]. Chacun des réseaux impliqués peut héberger des systèmes de défense et des sondes pour détecter et contrer les attaques, conduisant à trois types de détection différentes.

Au niveau de la victime : La majorité des IDS ont été conçu afin de protéger les systèmes des attaques dont ils sont la cible. Dans ce cas, la détection est plus simple que dans les autres cas car l'IDS peut observer de près la victime, modéliser son comportement et remarquer ses anomalies. Toutefois, cette approche souffre de quelques limites. Par exemple, lorsque le réseau est inondé par de grands volume de trafic, le point de défaillance est alors simplement déplacé de la victime ciblée pour devenir le système de défense lui même ou le point d'entrée du réseau.

Au niveau du réseau intermédiaire : Le problème des dangers d'une attaque DDoS sur les ressources du réseau dans le cas ou le système de défense est implémenté au niveau de la victime, a été abordé en déplaçant la défense plus en amont, dans le réseau intermédiaire.

Pour éviter les effets inévitables d'inidation au niveau des réseaux cibles d'attaques par déni de service, il est possible de placer les systèmes de détection en amont, dans les réseaux intermédiaires. Ce type d'IDS est généralement installé sur les routeurs de bordure des opérateurs de l'Internet. En effet, ce type d'équipement possède une bonne capacité de traitement des flux de trafic et est très efficace dans la détection des attaques à forte volumétrie sans toutefois résoudre complètement le problème des dommages collatéraux. De plus, ces IDS ne sont pas aussi efficace contre les attaques qui ne nécessitent pas de grands volumes de trafic mais qui infligent tout autant de dommages à leurs victimes.

Au niveau de la source : Cette solution a pour but de réaliser une détection la plus proche possible des réseaux sources de l'attaque. La proximité des IDS des sources augmente l'efficacité de la réaction et évite les dommage collatéraux sur les réseaux intermédiaires. Toutefois le caractère distribué de l'attaque fait qu'un IDS source ne peut détecter qu'une partie de l'attaque seulement et pour réaliser la détection complète d'une attaque, un grand nombre d'IDS de ce type doivent coopérer étroitement.

Remarque : La BCP[41] 38, identifiée par la RFC 2827 [87] publiée en mai 2000, s'inscrit dans cet effort de détection et arrêt des attaques par déni de service en recommandant aux opérateurs réseaux de l'Internet de filtrer toute forme de trafic dont l'adresse IP source a été usurpée. Toutefois, à ce jour, cette recommandation reste très peu déployée, favorisant la propagation des attaques par déni de service.

[41] Best Current Practice

Qu'ils soient situées du côté de la victime, de l'attaquant ou sur tout réseau intermédiaire, un IDS ne dispose que d'une vue locale pour sa prise de décision et sa mise en oeuvre de contre-mesures. Pour faire face à la nature distribuée et complexe des attaques actuelles, il est indispensable de construire une vue globale des informations collectées par les IDS et agir de façon coordonnée. Par conséquent, la collaboration entre IDS est nécessaire. Dans la section suivante, nous présenterons ce type d'IDS appelés IDS collaboratifs.

2.4.2 Les systèmes collaboratifs pour la détection d'intrusion

Un CIDS peut être vu comme un réseau de recouvrement qui permet à un IDS d'échanger avec d'autres IDS des informations d'intrusions afin d'améliorer la performance de détection globale du système. En d'autres termes, l'objectif d'un CIDS réside dans la production d'un aperçu de haut niveau sur l'état de sécurité de l'ensemble du réseau protégé ainsi que le maintien d'un bon équilibre entre les faux positifs et faux négatifs. Dans un CIDS, chaque noeud (IDS) peut ainsi bénéficier d'une vue plus globale sur les intrusions en recevant et mettant en corrélation des alertes d'autres IDS situés sur d'autres réseaux. Par exemple, il apparait qu'un IDS isolé s'appuie sur des mises à jour de règles et ne détecte pas les attaques dont les signatures ne sont pas enregistrées dans la base. La collaboration entre IDS permet à chaque entité d'utiliser les connaissances des autres afin d'atteindre une meilleure performance de détection d'intrusion, ce qui est particulièrement utile pour la prévention de nouvelles attaques. Les CIDS ont ainsi la capacité de détecter des attaques très distribuées à l'échelle de l'Internet, en en surveillant différents réseaux.

Les CIDS ont également le potentiel de réduire les coûts de calcul en exploitant les ressources des différents composants qui le constituent. En règle générale, un CIDS se compose de deux unités fonctionnelles principales :

1. **Unité de détection :** elle se compose de plusieurs capteurs ou IDS, où chacun surveille son propre sous-réseau ou hôte séparément, puis génère des données - de bas niveau - relatives à la détection, comme les états et les alertes d'intrusions.
2. **Unité de corrélation :** elle combine les différentes données - de bas niveau - reçues pour décider de l'état global de l'infrastructure surveillée.

Face à la complexité à l'hétérogénéïté des attaques, on note qu'un nouveau type de CIDS voit le jour : les réseaux de CIDS (*CIDS Networks*) [88]. Un réseau de CIDS représente un système où plusieurs IDS de différents types provenant de différents développeurs et traitant des données complètement différentes, collaborent afin d'augmenter la qualité de détection.

2.4.3 Architectures des IDS collaboratifs

De nombreux systèmes et schémas ont été proposés afin de permettre une agrégation et corrélation efficace des informations reçues de la part des différentes instance d'IDS. Ces schémas peuvent être classifiés en trois catégories, à savoir (1) approches centralisées, (2) approches hiérarchiques et (3) approches complètement distribuées.

2.4.3.1 Architecture centralisée

Dans ce type d'approche, toutes les informations collectées auprès de chaque IDS sont rapportées en un point central pour l'analyse. Chaque instance d'IDS joue le rôle d'une unité de détection où elle collecte les données et produit les alertes localement. Ensuite, ces dernières sont envoyées à un serveur central qui a le rôle d'une unité de corrélation pour l'analyse et la décision de l'état de sécurité global. Les approches centralisées sont généralement adaptées à des infrastructures à petite échelle comme celle d'une entreprise par exemple. Toutefois, elle ne sont pas adaptées à des réseaux à très grande échelle comme Internet. Cela est dû à deux principales faiblesses : (1) l'unité de corrélation (serveur central) devient un unique point de défaillance ou de vulnérabilité ; et (2) le passage à l'échelle.

Comme exemples illustratifs de cette architecture nous présentons plusieurs contributions phares de l'état de l'art :

Unsupervised Behavior Learning : UBL (*Unsupervised Behavior Learning*) [89] est un HIDS assurant une détection des anomalies de performance dans les systèmes virtualisés tel que le *cloud computing*. Il utilise un ensemble de modules d'apprentissage continu du comportement des VM afin de capturer leurs modèles de fonctionnement normal en s'appuyant sur des paramètres système (CPU, MEM, TX, RX). Afin d'atteindre cet objectif et prévoir les anomalies, UBL repose sur *Self Organizing Map* qui est une méthode d'apprentissage non supervisé au sein de laquelle on observe les déviations par rapport aux comportements normaux de la VM surveillée. UBL souffre toutefois de certaines limites ; la plus importante réside dans son implémentation sur un seul hôte. Ainsi, cette méthode ne peut pas construire de vision globale des intrusions et ne peut pas être efficace dans la détection d'attaques largement distribuées ayant une propagation rapide tels que les attaques de déni de service.

Dastjerdi *et al.* : [41] est une proposition de système de détection basée sur les agents mobiles qui peut être utilisée par les clients du *cloud*. Cette solution est en fait l'adaptation de deux autres approches aux spécificités de l'environnement *cloud*. Ces dernières sont DIDMA, un IDS distribué [90] et une solution P2P qui intègre les agents mobiles [91]. Les fondements de cette solution hybride se reposent sur quatre éléments principaux, à savoir : *IDS Control Center (IDSCC), IDS Agency, Application Specific Static Agent Detectors (SAD)* et *Specialized Investigative Mobile Agent (IMA)*. Les *SAD* sont distribués et déployés sur les différentes VMs surveillées. Le rôle des *SAD* réside dans la génération d'alerte pour toute activité suspecte détectée qui s'apparente à une possible attaque. Ces alertes sont structurées dans des messages où elles sont représentées par des identifiants qui correspondent aux types de menaces ou attaques. Ces messages sont envoyés au *IDSCC* qui envoie à son tour un *IMA* qui visitera chaque *Agency* qui a envoyé une alerte similaire. Le *IMA* va visiter et enquêter sur toutes les VMs, collecter des informations, les corréler et enfin envoyer le résultat au *IDSCC*. Par conséquent, l'*Alerting Console*, un composant du *IDSCC* analysera ces informations, en les comparant à des modèles d'intrusion enregistrés dans la base de données du IDSCC. Si une intrusion est détectée, une alarme est déclenchée. La VM compromise

sera ajoutée à une liste noire et une notification sera envoyée à toutes les VMs exceptées celles figurant sur cette même liste. Pour limiter la propagation de l'intrusion et garantir la sécurité globale de l'infrastructure, les VMs de la liste noire n'ont plus la possibilité de migrer. La plus-value apportée par cette solution est la proposition d'un système hautement évolutif, dynamique, qui s'exécute de manière asynchrone et autonome, opérant dans des environnements *cloud* hétérogènes et ayant un comportement robuste et tolérant aux pannes. Cependant, l'augmentation du nombre de VMs attachées au *IMAs* entraine l'augmentation substantielle de la charge réseau.

2.4.3.2 Architecture hiérarchique

Afin lever les limites de support du facteur d'échelle de l'approche centralisée, plusieurs conceptions hiérarchiques ont été introduites. Dans ce type d'approche, le CIDS est divisé en plusieurs petits groupes de communication basés sur un des critères suivants ; (1) emplacement géographique, (2) contrôle administratif, (3) plateformes logicielles similaires, et (4) types d'intrusions à détecter. Chaque groupe de communication représente un sous-ensemble de la hiérarchie et contient une unité de corrélation qui lui est propre. Cette unité de corrélation représente le noeud parent du groupe. Après traitement des données, ce dernier envoie les données corrélées a un noeud de niveau plus haut dans la hiérarchie pour une analyse ultérieure.

Les approches hiérarchiques supportent des échelles bien plus importantes que les approches centralisées. Cependant, les noeuds des niveaux plus élevés dans la hiérarchie limitent encore la fiabilité et le support du facteur d'échelle des CIDS, et leur défaillance peut arrêter le fonctionnement de l'ensemble de leur sous-arbre. En outre, les noeuds du niveau le plus élevé ont souvent une couverture de détection limitée en raison du niveau élevé d'abstraction des données d'entrée [76].

Il existe de nombreux travaux qui intègrent ce type d'architecture, nous citons :

EMERALD : Le projet EMERALD *(Event Monitoring Enabling response to Anomalous Live Disturbances)* [92] représente une proposition d'IDS hiérarchique qui vise la détection d'intrusions dans de gros réseaux d'entreprises. Dans cette approche, le réseau surveillé est divisé en trois couches abstraites : la couche service *(service layer)*, la couche domaine *(domain-wide layer)* et la couche entreprise *(enterprise-wide layer)*. Chaque couche comprend indépendamment des détecteurs qui combinent l'analyse des signatures et le profilage statistique.

Premièrement, la couche service représente la couche ayant le plus bas niveau dans la hiérarchie. Cette dernière est chargée de surveiller les composants individuels et les services réseaux au sein d'un seul domaine. Deuxièmement, les détecteurs de la couche du domaine surveillent et analysent l'utilisation malicieuse à travers de multiples services et composants. Troisièmement, la couche entreprise représente celle ayant le plus haut niveau. Elle est responsable de l'analyse de l'activité sur l'ensemble des domaines du système. Afin d'améliorer la capacité de détection, un mécanisme de communication par abonnement est utilisé afin de coordonner la diffusion de l'information. Par exemple, chaque détecteur est en mesure de souscrire

à l'information provenant des autres détecteurs au même niveau ou de niveau inférieur dans la hiérarchie. En effet, ce modèle de communication asynchrone permet à EMERALD de communiquer efficacement l'information là où elle est nécessaire. Cependant, en l'absence d'un réseau *overlay* structuré, assurer une communication fiable des message engendre de gros surcoût de communication et plus spécifiquement dans les réseaux a grande échelles.

Dependency-based Distributed Intrusion Detection : [93] représente une proposition de CIDS hiérarchique. Les machines participantes sont regroupées en régions coopératives. Le regroupement est basé sur la connaissance du réseau tel que la proximité entre les machines, leurs propriétés locales ou leurs contraintes et politiques de fonctionnement. Un Modèle de Markov Caché [94] est utilisé pour agréger les alertes collectées au sein de la même région. Ensuite, un test d'hypothèse séquentiel [95] est appliqué pour corréler globalement les résultats obtenus de l'ensemble des régions.

DSOC : Distributed Security Operation Center (DSOC) [96] est une architecture CIDS hiérarchique. Cette dernière comprend trois couches : (1) boite de surveillance ; (2) Analyseur Local ; et (3) Analyseur Global. Les informations suspectes sont collectées localement par les outils et dispositifs de la première couche, les boites de surveillance, telles que les pare-feu, les IDS ou les capteurs. Ensuite, les analyseurs locaux dispositifs de la deuxième couche analysent ces activités suspectes et génèrent des alertes en conséquence. Finalement, ces alertes sont envoyées à l'analyseur global. Ce dernier corrèle et analyse ces alertes et décide de l'état de sécurité global.

Multi-Agent Reinforcement Learning for Intrusion Detection : [97] représente une autre proposition de CIDS collaboratif hiérarchique qui repose sur un système multi-agents. Ce CIDS comprend trois niveaux. Le premier comprend des agents représentés par capteurs distribués sur différentes région du réseau surveillé. Leur rôle réside dans la collecte de données suspectes au niveau local. Les informations collectées sont envoyées au second niveau représenté par des agents ayant le rôle d'IDS régionaux. Un IDS régional applique une méthode apprentissage par renforcement [98] afin d'analyser les données recueillies et décider de celle à envoyer au troisième niveau. Ce dernier est représenté par un IDS qui applique également l'apprentissage par renforcement pour l'analyse de données et la décision de l'état de sécurité du système surveillé.

2.4.3.3 Architecture décentralisée

Afin de résoudre les limites de support du facteur d'échelle citées dans les deux dernières sections, de nombreux CIDS décentralisés ont été proposés. Dans ces approches entièrement distribuées, chaque IDS participant comprend deux unités fonctionnelles : une unité de détection responsable de la collecte de données au niveau local ; et une unité de corrélation qui représente une partie du schéma de corrélation distribué. Les IDS participants communiquent entre eux en utilisant des protocoles dédiés à la distribution de données, tels que le pair à pair (P2P), le multicast ou le protocole *publish/subscribe*.

De nombreux travaux se sont intéressé aux CIDS distribués. Parmi ces derniers nous citons :

Worminator : Worminator [99] est une proposition de CIDS complètement décentralisé qui repose sur une architecture pair-à-pair. Il a été proposé dans le but de permettre aux IDS de se partager des informations d'alerte afin de détecter la propagation d'anomalies et de vers. Chaque participant utilise un IDS pour surveiller son sous-réseau ou les hôtes dont il est responsable. Un outil appelé *Worminator* est exécuté sur les hôtes participants au sein desquels, à des intervalles réguliers, il convertit les alertes des IDS locaux sous forme de *watchlist* représentée par une liste d'adresses IP suspectes ainsi que les numéros de port associés, puis encode cette *watchlist* via les filtres de Bloom [100]. Ces *watchlist* encodées sont ensuite distribuées sur l'ensemble des participants via un réseau pair-à-pair. Enfin, *whirlpool* un outils de corrélation, est utilisé pour agrégé l'ensemble des alertes.

Denver *et al.* : [101] représente un CIDS complètement distribué qui utilise l'inférence probabiliste distribuée pour la détection des intrusions. Le système comprend trois composants : (1) les détecteurs locaux qui servent à la détection d'intrusion au niveau local par le déclenchement d'une alerte si le nombre de connexions nouvellement créées par unité de temps dépasse un certain seuil défini ; (2) Les détecteurs globaux responsables de la génération des vues globales d'attaques par l'analyse des informations sur l'état des différents détecteurs locaux en se basant sur un modèle probabiliste ; et (3) un système de partage d'informations, qui utilise un protocole épidémique *(gossip)* afin d'assurer la communication entre les détecteurs locaux et globaux. Le système proposé utilise uniquement le taux de nouvelles connexions comme indicateur d'intrusions, ce qui n'est pas efficace contre les intrusions et vers qui se propagent sans connexion, en reposant sur UDP.

DOMINO : Le projet DOMINO [102] représente un CIDS distribué qui vise la surveillance des épidémies à l'échelle d'Internet. Le système se compose de trois types de noeuds : (1) les noeuds satellites sont organisés hiérarchiquement afin de composer des communautés de satellites et sont responsables de la collecte des données d'intrusion et de leur envoi au noeud parent dans la hiérarchie. Ces derniers agrègent les données reçues ainsi que celles collectées et ils les envoient à leur tour au parents jusqu'à atteindre les noeuds axe ; (2) les noeuds axe sont connectés par un overlay pair-à-pair qui représente le composant central de DOMINO ; enfin (3), les contributeurs terrestres représentent un moyen d'étendre la couverture en incluant d'autres ensembles externes de données d'intrusions. Les noeuds axes s'échangent périodiquement les informations d'intrusions. Chaque noeud axe maintient deux vues sur les activités d'intrusions : (1) une vue locale créée à partir des données et alertes collectées ainsi que celles récupérées par leurs noeuds satellite respectifs et (2) une vue globale créée à partir des alertes reçues par d'autres noeuds axe du système.

MADIDF : Mobile Agents based Distributed Intrusion Detection Framework (MADIDF) [103] représente un CIDS complètement distribué qui repose sur les agents mobiles. Dans ce système, chaque hôte participant se compose de quatre agents : (1) un agent de surveillance, (2) un agent d'analyse, (3) un agent exécutif et (4) un

agent gestionnaire. L'infrastructure dispose également de deux agents mobiles : un agent de récupération d'information et un agent de résultat qui peuvent se déplacer entre les hôtes. Lorsque le système est sujet à une attaque distribuée, l'agent d'analyse détecte l'activité suspecte en se basant sur les informations recueillies par l'agent de surveillance, et envoie une demande à l'agent gestionnaire pour un complément d'enquête. À la réception de cette demande, l'agent gestionnaire envoie un agent de récupération afin de recueillir des informations pertinentes à partir d'autres hôtes. Ces derniers - visités par l'agent de récupération - envoient un agent de résultat à l'agent gestionnaire initial. L'agent gestionnaire peut donc prendre une décision finale en se basant sur les informations reçues.

FireCol : FireCol [77] représente un CIDS distribué qui vise la détection des attaques DDoS de type *flooding* au plus proche possible de la source. FireCol repose sur une architecture distribuée composée de multiples IPSs qui forment des réseaux overlay sous forme d'annaux de protection autour des clients qui y souscrivent. Un anneau est composé d'un ensemble d'IPS qui sont à la même distance (nombre de sauts) du client. Chaque IPS appartenant à FireCol analyse le trafic agrégé et collabore via des communications verticales donc à travers différents niveaux d'anneaux, en calculant et échangeant des informations sur de potentielles attaques, tout au long du chemin jusqu'au client protégé. Si la probabilité d'une potentielle attaque est élevée, les IPSs utilisent alors des communications horizontales, donc au sein du même anneau virtuel. De cette manière, la menace est mesurée sur la base de la taille de bande passante globale du trafic dirigé vers le client par rapport à la bande passante maximale qu'il peut supporter.

D-WARD : DDoS Network Attack Recognition and Defense (D-WARD) [104] est un système de défense contre les attaques DDoS, dont l'objectif est la détection des attaques DDoS au niveau du réseau source leur arrêt en contrôlant le trafic sortant à destination de la cible. La détection est assurée par la surveillance et l'observation des paquets échangés entre le réseau source, qui est surveillé, et le monde extérieur. Cette détection est assurée par une approche comportementale. D-WARD comprend trois composants principaux : (1) un composant d'observation qui réalise des statistiques sur les flux et les connexions et en se basant sur ces statistiques, il classifie ces connexions ; (2) un composant de politique de trafic qui joue le rôle d'un observateur et qui détermine le taux de lissage de flux quand c'est nécessaire ; et (3) un composant de limitation de trafic. Ce dernier reçoit les résultats de classification du composant d'observation et de l'historique de comportement de flux du composant de surveillance et calcul ainsi une valeur de taux limite pour les flux. D-WARD offre de bonne performances de détection. Cependant, pour être efficace, de nombreux domaines doivent l'implémenter ce qui rend son utilisation pratique, limitée.

2.4.4 Synthèse des approches étudiées

Le tableau 2.1 présente une synthèse des travaux présentés précédemment, classifiés selon les critères d'architecture, type de données, emplacement et enfin, méthodes de

détection, tels que vus dans ce chapitre. On y constate que les architectures proposées sont variées, avec une tendance à la proposition de solutions distribuées pour les contributions les plus récentes. En outre, majoritairement, les données considérées sont issues des piles protocolaires réseau a contrario des données système. On note aussi que la grande majorité des approches est conçue pour être déployée au niveau de la cible de l'attaque, pour pouvoir la protéger au mieux. Enfin, on remarque que la majeure partie des approches de détection sont comportementales.

Le contexte de travail dans lequel nous inscrivons ce travail de thèse est celui d'un fournisseur de services *cloud* dont les ressources sont utilisées de façon mal-intentionnée pour mettre en oeuvre des attaques envers des tiers connectés à l'Internet. Il apparait que si les solutions actuelles présentent toutes des éléments de similitudes avec notre travail, elles ne répondent pas totalement aux besoins exprimés dans ce manuscrit. En effet, on comprend dans ce cadre qu'une solution de détection au plus proche de la source de l'attaque est nécessaire pour supprimer tout dommage collatéral, sur l'Internet mais aussi sur l'infrastructure opérée par le fournisseur de services. Le premier élément d'infrastructure auquel une VM est confronté est la couche de virtualisation et c'est pourquoi, nous proposons d'y placer notre solution de détection qui est dans ce cadre de type HIDS. Sachant que les VMs appartenant à l'attaquant peuvent être instantiées sur plusieurs hôtes physiques, il apparait aussi nécessaire de distribuer la solution de détection entre les hôtes. Enfin, de par la nature hétérogène de l'activité d'un *cloud*, il nous apparait plus abordable, comme première solution, de mettre en oeuvre une approche par signature.

2.4.5 Formalisation des informations échangées

Dans le contexte de la détection collaborative, la collecte des alertes est une étape essentielle qui s'avère compliquée du fait de l'hétérogénéité des sondes. De nombreuses recherches s'appuient sur les travaux de l'IDWG *(Intrusion Detection Work Group)* qui ont décrit un format d'échange de messages de détection d'intrusion IDMEF *(Intrusion Detection Message Exchange Format)*. Ces travaux ont été repris par Debar et Wespi [105] en 2001 pour structurer les messages au format XML. En 2007, le modèle fait l'objet de la RFC 4765 [106]. En 2007 Danyliw *et al.* [107] publiaient la RFC 5070 relative au modèle d'échange de données intitulé IODEF *(Incident Object Description Exchange Format)*. En novembre 2010 deux autres RFC ont été publiée la RFC 6045 [108] et la RFC 6046 [109] qui permettent de normaliser l'échange de données entre les équipements.

Le modèle de données IDMEF comprend les trois principales informations qui permettent de caractériser une attaque et qui sont indispensables aux différents procédés de corrélation : la source de l'attaque, la cible de l'attaque et l'horodatage de l'alerte. D'autres informations peuvent également y être adjointes pour mieux caractériser l'évènement à l'origine de l'alerte comme par exemple l'identifiant de l'attaque, le niveau de criticité et l'identifiant de la sonde.

Approaches	**Architecture**			**Type de données**		**Emplacement**			**Méthode de détection**	
	Centrale	Hiérar.	Distrib.	Réseau	Syst.	Cible	Interm.	Source	Comport.	Sign.
UBL [89]	x				x	x			x	
Dastjerdi *et al.* [41]	x					x				x
EMERALD [92]		x		x		x			x	x
Dependency-based IDS [93]		x		x		x			x	
DSOC [96]		x		x	x	x			x	
MARL [97]		x		x		x			x	
Worminator [99]			x	x		x				
Denver *et al.* [101]			x	x		x			x	
DOMINO [102]			x	x		x			x	
MADIDF [103]			x	x	x	x			x	x
FireCol [77]			x	x			x		x	
D-WARD [104]			x	x				x	x	

Table 2.1 – Comparaison des IDS collaboratifs.

2.5 Conclusion

Dans ce chapitre nous avons exposé en premier lieu le problème des *botnets*. Nous en avons détaillé les types, architectures et dommages potentiels. Les *botnets* sont utilisés principalement pour la génération d'attaques DDoS. Par conséquent, nous avons par la suite introduit ces types d'attaques, leur classification et techniques de mises en oeuvre. Les *botnets* et les attaques DDoS représentent la plus grande menace de la sécurité de l'Internet et de ses utilisateurs et les IDS sont considérés comme la meilleure solution pour pallier cette menace. La deuxième partie de ce chapitre a ainsi traité de ce type de solution de détection. La plupart des solutions d'IDS repose sur un déploiement sur un emplacement unique induisant de nombreux inconvénients tels qu'un point de défaillance unique et l'incapacité de construire une vue globale sur l'état du réseau surveillé et de sa sécurité. Pour résoudre le problème de l'isolation des IDS, les IDS collaboratifs ont été proposés et dans la dernière partie de ce chapitre nous avons présenté leur architectures, fonctionnements ainsi que des propositions caractéristiques de l'état de l'art. Dans le chapitre suivant, nous présentons la première étape de la conception d'une approche collaborative de détection à la source et qui consiste en la caractérisation des *botclouds*.

Deuxième partie

Contributions

3

Caractérisation système des *botclouds* : Application aux DDoS

Sommaire

3.1 Introduction

Dans le but de concevoir un système de détection à la source de *botclouds*, la première étape de notre travail réside dans la caractérisation de ces derniers. En effet, comprendre le comportement et les réactions d'un *botcloud* est une étape clé dans la conception d'un système de détection. Pour atteindre cet objectif, nous avons conduit une large campagne de mesures où nous avons réalisé un ensemble d'expérimentations visant la reproduction *in situ* d'attaques DDoS de types UDP Flood et TCP SYN Flood produites par deux implémentations logicielles différentes de *botnet*, à savoir Hybrid_V1.0 et Kaiten. Cette étude a conduit à la collecte de 31,15 GBytes de fichiers journaux. Afin d'étudier ces

données et caractériser le comportement des *botclouds*, une méthode d'analyse multidimensionnelles de données fut nécessaire et l'Analyse en Composantes Principales (ACP) fut celle que nous avons sélectionnée pour traiter ce type de cas.

Ce chapitre décrit ainsi une première caractérisation du comportement global d'un *botcloud*. Nous analysons, via l'ACP le comportement des paramètres système de l'ensemble des VM appartenant à un *botcloud*, et plus précisément le comportement intrinsèque de ces dernières et leurs corrélations. Ensuite, nous produisons une caractérisation plus approfondie du comportement d'un *botcloud* pendant la seule phase d'attaque et nous démontrons que ce comportement est très peu variable quelque soit le débit d'attaque ou l'implémentation logicielle mise en oeuvre.

L'organisation de ce chapitre est la suivante. Dans un premier temps, nous décrivons en détail notre contexte de recherche. Dans un second temps, nous présentons le contexte général de notre étude où nous décrivons le cadre expérimental que nous avons utilisé lors de nos expérimentations ainsi que les scénarios reproduits. Dans un troisième temps, nous présentons les principes et applications de l'Analyse en Composantes Principales, qui est la méthode statistique utilisée dans l'étude menée et poursuivie tout au long de ce manuscrit. Dans un quatrième temps, nous décrivons une première caractérisation macroscopique d'un *botcloud* réalisant des attaques DDoS de types UDP Flood et TCP SYN Flood. Enfin, nous affinons cette étude pour identifier les invariants comportementaux des *botclouds*.

3.2 Cadre de l'étude

Cette section traite du cadre de l'étude de caractérisation que nous avons menée en décrivant son contexte général, le cadre expérimental et les scénarios des expérimentations que nous avons mises en oeuvre, ainsi que la méthode statistique utilisée.

3.2.1 Contexte général

Le contexte général de notre étude représente le cas d'un opérateur de *cloud* public offrant des services de type IaaS, comme c'est le cas pour Amazon EC2. Le fournisseur de services est propriétaire d'une infrastructure constituée de différents centres de données (*Data Centers*). Chaque centre de données est composé d'un ensemble de serveurs physiques hébergeant un ensemble de VMs appartenant à un ensemble de *tenants*. Dans ce contexte, un *tenant* est défini comme une infrastructure logique, composée d'un ensemble de VMs et appartenant à une tierce entité. Parmi ces *tenants*, un ou plusieurs sont malicieux et hébergent un ou plusieurs *botclouds*. Nous considérons que les machines composant le *botcloud* sont dédiées uniquement à l'attaque, comme dans le cas de l'étude [8] détaillée dans la section 1.4.3 et qui vise l'étude des potentiels avantages d'une infrastructure de services *cloud* pour un tiers malveillant.

Comme premier cas d'étude, nous nous intéressons aux attaques DDoS et plus précisément aux attaques de type *flooding* telles que UDP Flood et TCP SYN Flood. En effet, la popularité de ces attaques est due à leur grande efficacité contre tout type de service, étant donné qu'elles ne nécessitent pas d'identifier ou d'exploiter une faille majeure dans le protocole ou le service cible [77], mais il suffit juste de l'inonder.

3.2.2 Paramètres considérés

Comme données de surveillance *(monitoring)* de l'ensemble des VMs, nous nous intéressons aux paramètres système disponibles au niveau de l'hyperviseur ou le système hôte dans le cas des technologies de virtualisation à base de conteneur, obtenues en reposant sur une approche non intrusive. Nous considérons une telle approche de surveillance et de tels paramètres pour les raisons suivantes :

1. Effectuer une surveillance à l'aide de sondes implantées au niveau des VM pose des problèmes d'ordre juridique [10]. Afin de garantir la vie privée des utilisateurs, nous considérons une approche *Black-Box*, c'est à dire qui n'expose aucune donnée sur l'activité interne de la VM mis à part celles rendues disponibles à un hyperviseur.
2. Au delà de notre cas d'usage qui porte sur les attaques DDoS, la considération de paramètres système permet d'élargir la spectre des attaques à détecter, incluant notamment celles qui ne se reposent pas sur une activité réseau, comme une attaque *Brute-Force* par exemple.
3. En raison de la volumétrie des données de surveillance liée au nombre de VM hébergées dans une infrastructure de *cloud computing*, il est impératif de considérer une approche efficace mais peu couteuse en terme de ressources et de temps de calcul. Par conséquent, nous choisissons dans notre approche de ne reposer que sur un faible nombre de métriques.

Par conséquent, les paramètres système collectés sont : (1) la consommation processeur, exprimé en pourcentage du taux d'utilisation, (2) la mémoire virtuelle, exprimée en KiloBytes, (3) la bande passante envoyée en Kilobit par seconde et (4) la bande passante reçue en Kilobit par seconde. Dans la suite de ce manuscrit, nous appelons ces métriques respectivement CPU, MEM, TX et RX.

3.2.3 Cadre expérimental et scénarios

Afin d'obtenir des résultats en accord avec la réalité d'un *botcloud* réalisant des attaques DDoS, il est impératif d'utiliser des données qui proviennent de mises en oeuvre réelles. Cependant, l'utilisation d'une véritable infrastructure de *cloud* public nécessite l'étroite coopération de son opérateur, ce qui s'avère une tâche difficile à réaliser.

Comme alternative, il est possible (1) d'utiliser un jeu de données existant ou (2) d'utiliser des modèles de charge issus de l'état de l'art. Concernant la première alternative, il existe de nombreuses bases de données et dépôts qui rassemblent de grandes quantités de jeux de données, dédiés à une utilisation libre par la communauté scientifique. Parmi ceux-ci, on recence GoogleClusterData[42], CGIAR[43], NASA NEX[44], Apache Software Foundation Public Mail Archives[45] et Freebase Quad Dump[46]. Toutefois, ces jeux de données ne sont pas exploitables dans le cadre de notre étude car ils ne fournissent

[42]http ://code.google.com/p/googleclusterdata/
[43]http ://ccafs.cgiar.org
[44]https ://aws.amazon.com/fr/nasa/nex/
[45]http ://mail-archives.apache.org/mod_mbox/
[46]https ://aws.amazon.com/datasets/2052645406658757

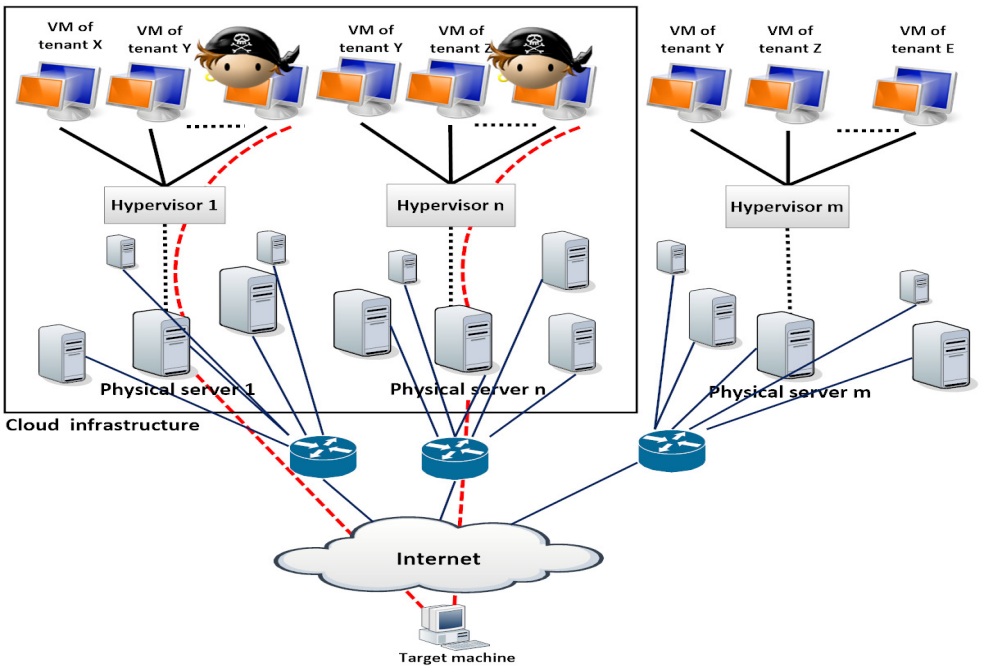


FIGURE 3.1 – Environnement d'expérimentation

aucune donnée relative aux paramètres que nous avons choisis. Concernant la seconde alternative, il existe également une multitude de modélisations de la charge supportée par une infrastructure de *cloud computing*. Cependant, le point de vue de modélisation de ces dernières est dédié à l'étude de la planification [110] [111] et la répartition de charge [112] [113] ainsi que l'économie d'énergie [114], et s'avère par conséquent pas adapté aux études sur la sécurité. Il s'avère donc qu'aucune de ces deux alternatives ne permet de nourrir notre étude et c'est pourquoi, nous avons reproduit des attaques DDoS *in situ*.

Concernant le choix de l'environnement d'exécution, nous avons besoin d'un environnement virtualisé dans lequel plusieurs *tenants* exécutent des services légitimes et où un *tenant* malicieux peut déployer un *botcloud* et produire des attaques. Comme environnement ouvert et réaliste pour nos expérimentations, nous avons choisi PlanetLab [115]. En effet, PlanetLab représente une plateforme virtualisée grâce à LXC et elle repose sur le principe de *multi-tenancy* qu'elle met en oeuvre sous la forme d'espaces cloisonnées de ressources appelés *slices*. On note toutefois que la seule caractéristique que PlanetLab n'est pas en mesure de reproduire est la migration de VMs entre hyperviseurs. Cependant, la répartition statique des VMs sur des serveurs physiques effectuée par PlanetLab n'affecte pas notre étude car les seules métriques que nous mesurons (MEM, TX, RX et CPU) sont indépendantes de toute localisation ou migration d'un hyperviseur à un autre.

La figure 3.1 décrit l'environnement d'expérimentation où un ensemble de serveurs physiques (plus de 40) de PlanetLab est choisi pour représenter l'infrastructure physique de l'opérateur de *cloud*. Chaque serveur physique héberge une VM infectée et contrôlée par un *Botmaster* distant qui leur donne l'ordre d'attaquer des machines connectées à Internet.

Débit d'attaque	#Serveurs physiques	#Tenants (attaquant incl.)	#VMs (attaquant incl.)	# VMs attaquantes	Débit d'attaque agrégé
Expérimentations UDP Flooding					
8 Mb/s	41	123	1288	41	328 Mb/s
16 Mb/s	41	118	1261	41	656 Mb/s
40 Mb/s	43	123	1310	43	1,68 Gb/s
56 Mb/s	41	114	1241	41	2,25 Gb/s
80 Mb/s	40	103	1198	40	3,12 Gb/s
Experimentations TCP SYN Flooding					
159 p/s	41	123	1288	41	6500 p/s
171 p/s	47	115	1376	47	8000 p/s
290 p/s	47	114	1310	43	12500 p/s
420 p/s	48	118	1509	48	20000 p/s

TABLE 3.1 – Paramètres numériques des expérimentations menées avec le *bot* Hybrid_V1.0

Concernant le choix du *botnet*, il existe une multitude d'implémentations utilisées par les pirates informatique et utilisateurs malveillants. Comme décrit dans la section 2.2, un *botnet* se caractérise notamment par son implémentation logicielle (langage de programmation, algorithme, etc.) et par la technologie de communication qu'il utilise pour relier le serveur C&C et les bots. Les canaux IRC (Internet Relay Chat) et HTTP représentent les formes de communications les plus communes chez les *botnets*.

Etant données ces caractéristiques, afin d'identifier si la technologie de communication et l'implémentation logicielle d'un *botnet* ont une influence sur le comportement de son activité système, nous avons reproduit notre campagne d'expérimentation avec deux *botnets* différents. Ces derniers sont (1) Hybrid_V1.0[47] programmé en PERL, un langage de haut niveau et communicant via le protocole HTTP, et (2) Kaiten[48] programmé en C, un langage de bas niveau et qui utilise le protocole IRC pour les communications.

Nous avons réalisé au final quatorze expérimentations. Ces dernières impliquent des attaques UDP Flood avec différents débit ; de 8Mb/s à 80 Mb/s par source atteignant des débits agrégés entre 328 Mb/s et 3.125 Gb/s, ainsi que des attaques TCP SYN Flood avec des débits entre 159 paquets/s et 420 paquets/s atteignant des débits agrégés entre 6500 paquets/s et 20000 paquets/s. Les débits par source[49] ont été choisi dans un intervalle allant d'un débit faible jusqu'à atteindre un débit très élevé dans le but d'étudier l'impact de ce débit d'attaque sur l'empreinte système du *botcloud*. Aussi, pour chaque expérimentation, nous avons capturé l'activité système de plus de 100 *tenants* légitimes afin de pouvoir les confronter à celle du *bot*, et ce à différentes étapes du travail présenté

[47] http ://security-sh3ll.blogspot.com/2010/01/hybrid-botnet-system-v10-released.html

[48] http ://www.hackforums.net/showthread.php ?tid=974543

[49] Dans cette étude nous considérons les débits par source car nous nous intéressons aux paramètres système de chaque VM.

Débit d'attaque	#Serveurs physiques	#Tenants (attaquant incl.)	#VMs (attaquant incl.)	# VMs attaquantes	Débit d'attaque agrégé
Experimentations UDP Flooding					
8 Mb/s	43	125	1625	43	344 Mb/s
16 Mb/s	43	122	1511	43	688 Mb/s
40 Mb/s	43	123	1528	43	1,68 Gb/s
56 Mb/s	43	123	1528	43	2,35 Gb/s
80 Mb/s	43	128	1610	43	3.36 Gb/s

TABLE 3.2 – Paramètres numériques des expérimentations menées avec le *bot* Kaiten

dans ce manuscrit : l'analyse du comportement, la conception du détecteur et enfin sa validation.

Enfin, le scénario de chaque expérimentation est le suivant. Chaque expérimentation est divisée en trois étapes, chacune dure une heure. Ces dernières sont : (1) une première étape d'état normal où le bot est déployé, actif et prêt à l'attaque. (2) une seconde étape d'attaque contre une tierce cible et (3) une dernière étape où l'attaque s'arrête et le système revient à son état normal. Les tableaux 3.1 et 3.2 décrivent les différents paramètres et statistiques de ces expérimentations. La collecte des données des VMs, légitimes et attaquantes, est effectuée toutes les minutes via l'outil Slicestat[50].

3.2.4 Méthodologie d'analyse

Lors des expérimentations réalisées, nous avons collecté 31.15 GBytes de fichiers journaux. Afin d'étudier ces données et mettre en lumière des points tangibles du comportement des *botclouds*, nous avons choisi une méthode d'analyse de données multidimensionnelles qui permet de mettre en exergue les relations de corrélation, quelles que soient leur nature. L'idée sous tendue par ce choix consiste à mettre en évidence les relations entre paramètres mais aussi celles entre les VM. Pour ce faire, nous avons retenu l'Analyse en Composante Principale que nous présentons ci-après.

3.2.4.1 Analyse en Composante Principale des données

L'Analyse en Composantes Principales (ACP) [116] représente une méthode d'analyse de données appartenant à la famille de la statistique multidimensionnelle et plus spécifiquement des méthodes factorielles. Elle vise à faciliter l'exploration et l'analyse de vecteurs de grande dimension de données par la réduction de leurs dimensions ainsi que l'extraction de certaines caractéristiques majeures. Etant donné une matrice de données composée de n observations, également appelés individus et décrites par p variables, l'ACP décrit la structure de variance-covariance ou de corrélation de l'ensemble des variables à travers de nouvelles variables, appelées composantes ou facteurs principaux.

[50] codeen.cs.prinston.edu/slicestat/

Les composantes principales représentent des fonctions des variables initiales. Plus spécifiquement, elles représentent des combinaisons linéaires des p variables ayant des propriétés importantes : les premières composantes principales (≤ 3), ont respectivement les plus grandes variances et elles représentent ainsi le plus significativement les données initiales dans un espace de dimensions réduites, permettant ainsi de mettre en évidence leurs relations linéaires. De plus, les composantes principales ne sont pas corrélées et leur variance totale est égale à la variance totale de toutes les variables originales, ce qui permet une représentation fidèle des données initiales.

3.2.4.2 Formalisation de la méthode

Une ACP a pour objectif la description de données contenues dans un tableau a n individus mesurés sur p variables. Lorsque p est égal à 2 (x^1 et x^2), il est facile de représenter sur un plan l'ensemble des données. Chaque individu e_i est alors représenté par un point de coordonnées x_i^1, x_i^2 et le simple examen visuel du nuage de points permet d'interpréter la liaison entre x^1 et x^2 et de repérer les individus ou groupes d'individus présentant des caractéristiques voisines. Lorsque p est égal à 3, l'étude visuelle est toujours possible, en utilisant la géométrie dans un espace à trois dimensions. Toutefois, dès que le nombre p de variables est supérieur ou égal à 4, cela devient impossible.

Supposons la représentation de n individus sur p variables où $p \geq 4$ sur un plan. Ce qui sera visible sur le graphique sera une représentation déformée de la configuration initiale. En effet, les distances entre les n points sur le plan ne peuvent pas être toutes égales aux distances entre les même n individus dans l'espace complet à p dimensions, à moins qu'il existe $p-2$ relations linéaires exactes entre les variable [117]. Il y aura donc forcément des distorsions qu'il faut rendre minimes. Géométriquement, le graphique représente la projection des point individus $e_1, e_2, \ldots, e_n$ sur un plan. Par conséquent, il s'impose de choisir le plan de projection sur lequel les distances seront le mieux conservées. Etant donné que l'opération de projection raccourcie toujours les distances $d(f_i; f_j) \leq d(e_i; e_j)$ où $(f_1, f_2, \ldots, f_n)$ représentent les projections des $(e_1, e_2, \ldots, e_n)$, le critère de choix du plan est de rendre maximale la moyenne des carrés des distances entre les projections $f_1; f_2; \ldots; f_n$.

Selon Saporta *et al.* [117], afin de déterminer ce plan appelé plan principal, il suffit de trouver deux droites Δ_1 et Δ_2. Si Δ_1 et Δ_2 sont perpendiculaires alors :

$$d^2(f_i; f_j) = d^2(\alpha_i; \alpha_j) + d^2(\beta_i; \beta_j) \tag{3.1}$$

où les α_i et les β_i représentent les projections des e_i (et des f_i) sur Δ_1 et Δ_2 respectivement.

D'après l'équation 3.1, la moyenne des carrés des distances entre les f_i est donc égale à la moyenne des carrés des distances entre les α_i plus la moyenne des carrés des distances entre les β_i. La méthode consiste ainsi à chercher tout d'abord Δ_1 qui maximise la moyenne des $d^2(\alpha_i; \alpha_j)$ puis Δ_2 qui maximise la moyenne des $d^2(\beta_i; \beta_j)$.

Le calcul ne se limite pas à un seul plan (2 dimensions), mais il peut se poursuivre, conduisant à $\Delta_3, \Delta_4, \ldots, \Delta_p$ perpendiculaires entre elles. Ces Δ_i représentent les axes principaux. La projection des e_i qui avaient pour coordonnées initiales $(x_i^1, x_i^2, \ldots, x_i^p)$ sur les axes principaux permet l'obtention de nouvelles coordonnées $(c_i^1, c_i^2, \ldots, c_i^p)$. Ainsi, cette méthode permet la construction de nouvelles variables $(C^1, C^2, \ldots, C^p)$ appelées les

composantes principales. Chaque composante C^k représente la liste des coordonnées des n individus sur l'axe Δ_k et représente une combinaison linéaire des variables initiales, comme décrit par l'équation 3.2 :

$$C^k = u_1^k X^1 + u_2^k X^2 + \ldots + u_p^k X^p \tag{3.2}$$

Les coefficients $(u_1^k, u_2^k, \ldots, u_p^k)$ forment le k-ième facteur principal u^k.

La meilleure représentation des données au moyen de seulement q variables ($q < p$) s'obtient alors en ne retenant que les q premières composantes principales. Tel est le schéma de l'ACP qui représente donc une méthode de réduction du nombre de variables permettant des représentations géométriques des individus et des variables. On note que cette réduction n'est possible que si les p variables initiales ne sont pas indépendantes et ont des coefficients de corrélation non nuls.

D'un point de vue méthodologique, la réalisation d'une ACP conduit vers un nouvel espace dans lequel les données sont projetées. Ce nouvel espace est constitué des vecteurs propres qui permettent l'obtention des composantes principales. Chaque composante principale est le résultat de la multiplication des points de la matrice initiale par le vecteur propre sous-jacent. Ainsi, les vecteurs propres représentent la base du nouvel espace vectoriel, autrement dit, ses vecteurs unitaires.

Pour conclure, l'ACP représente une méthode *factorielle* et linéaire car la réduction du nombre de variables ne se fait pas par une simple sélection de certaines d'entre elles, mais par la construction de nouvelles variables synthétiques obtenues en combinant linéairement les variables initiales au moyen de *facteurs* [117].

3.3 Analyse du comportement global du *botcloud* : un premier résultat

Dans cette section, nous présenterons une première caractérisation du comportement globale du *botcloud* hébergé par un *tenant* malicieux. Cette étude porte ainsi sur un sous-ensemble du jeu de données collecté et concerne l'étude des distributions des métriques et de leurs corrélations.

3.3.1 Analyse de la distribution

Les figures 3.2.a et 3.2.b représentent les distributions statistiques des métriques mesurées sur les VMs attaquantes appartenant au *botcloud* Kaiten réalisant l'attaque UDP Flood avec un débit de 56Mb/s par source et le *botcloud* Hybrid_V1.0 réalisant l'attaque TCP SYN Flood avec un débit de 290 paquets/s par source.

Considérons les paramètres un par un. Nous constatons que : pour les deux types d'attaque et concernant les deux implémentations de *botnet*, l'activité CPU est généralement faible. En revanche, la consommation mémoire est clairement mise en forme par une distribution bi-modale, caractérisant les deux phases, hors et pendant l'attaque. Concernant les activités TX et RX, nous observons également une distribution bi-modale. Nous notons que dans les deux cas, une centralisation du premier mode autour de zéro, ceci

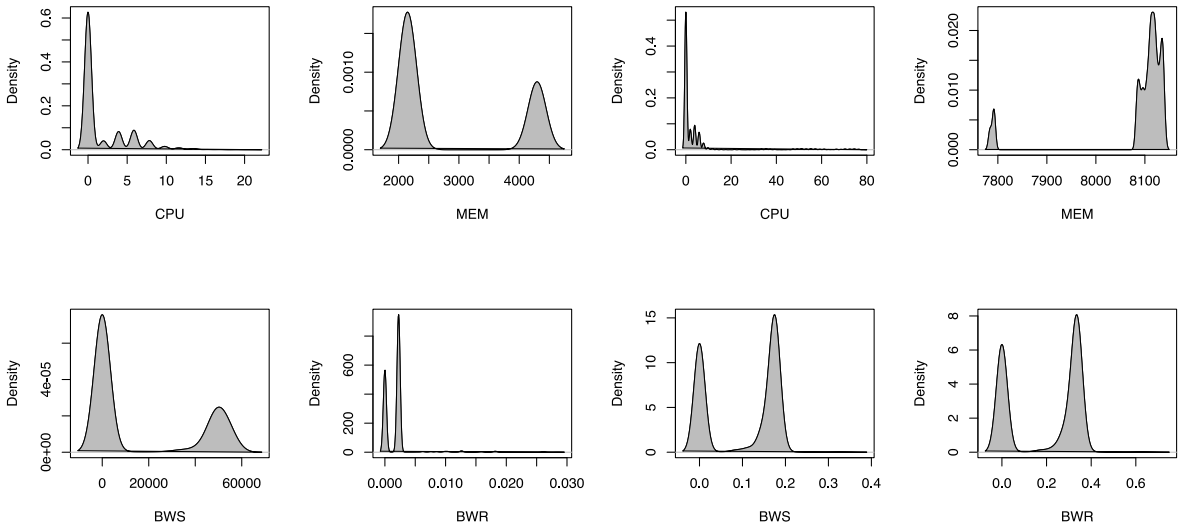


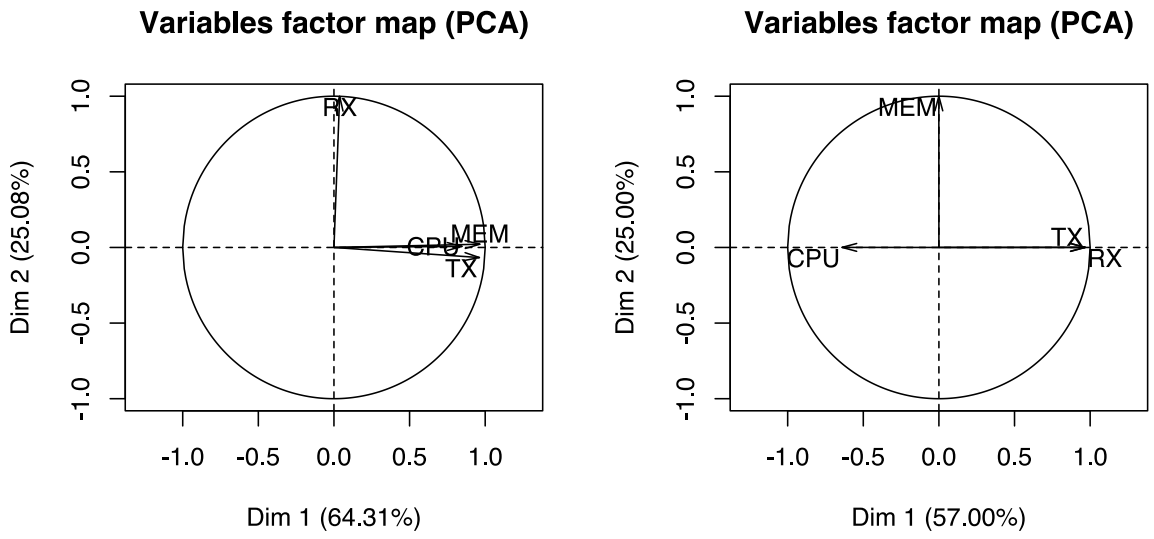
FIGURE 3.2 – Distributions statistiques des métriques ; (a) *botnet* Kaiten, attaque UDP Flood à 56Mb/s ; (b) *botnet* Hybrid_V1.0, attaque TCP SYN Flood à 290 paquets/s

est due à une importante activité réseau nulle, hors les phases d'attaque. Toutefois, selon la direction de la communication, le second mode a une signification différente. En effet, pour RX, le second mode représente le trafic de signalisation reçu par les Bots de la part du *botmaster*, tandis que pour TX, le second mode représente le trafic d'attaque envoyé au cours de la phase d'attaque.

L'annexe A (figure A.1 à figure A.3) fournit tous les résultats obtenus sur l'étude de la distribution des paramètres sur l'ensemble des expérimentations. Pour chaque type d'attaque, les distributions obtenues sont presque invariantes et d'autant plus pour une même implémentation logicielle de *botcloud*. Plus précisément, il apparait que les consommations MEM (par exemple 150 KBytes pour Hybrid_V1.0) et RX sont constantes, quelque soit le débit d'attaque. En revanche CPU et TX augmentent avec le débit d'attaque. Enfin, quelle que soit la métrique considérée, nous notons que toutes les distributions sont fortement concentrées autour de leurs modes dénotant une forte corrélation du comportement de fonctionnement des *bots*. Cette similarité de comportement des *bots* est absente dans les *botnets* traditionnels où les comportements des *bots* sont totalement hétérogènes, puisque ils sont étroitement liés au comportement de leurs utilisateurs.

3.3.2 Analyse des corrélations

Si les résultats présentés dans la section précédente donnent un premier aperçu sur la façon dont un *botcloud* dédié à l'attaque fonctionne, ils ne sont pas suffisants pour (1) comprendre comment les métriques sont liées les unes aux autres de sorte que des caractéristiques supplémentaires puissent être extraites et (2) construire une vision synthétique



(a) (b)

FIGURE 3.3 – Cercles de corrélations des ACP réalisées sur ; (a) *botnet* Kaiten, attaque UDP Flood à 56Mb/s ; (b) *botnet* Hybrid_V1.0, attaque TCP SYN Flood à 290 Paquets/s

du comportement de l'attaquant en se basant sur les données recueillies, comportement qui puisse être comparé à l'activité des autres *tenants* dans le but de les discriminer. A cette fin, nous avons effectué des ACP sur les différentes données collectées sur l'ensemble des expérimentations. Etant donné que toutes les variables ne sont pas homogènes, nous avons centré et réduit chacune d'entre elles avant de les traiter. On note enfin que dans cette section nous présenterons uniquement les résultats correspondant aux deux expérimentations décrites dans la section précédente.

La figure 3.3.a représente le cercle de corrélations de l'ACP réalisée sur le *botcloud* Kaiten réalisant l'attaque UDP Flood à 56 Mb/s. Ce type de graphique décrit les corrélations entre les différentes métriques étudiées et entre ces dernières et les nouvelles composantes principales. Chaque métrique est représentée par un vecteur sur le plan construit avec les deux premières composantes principales. L'axe des x représente toujours la première composante principale, c'est à dire celle qui possède la plus grande variance des données initiales. Le coefficient de corrélation entre deux vecteurs ou entre un vecteur et un axe du plan est défini par le cosinus de l'angle formé par ces deux derniers.

Pour ce premier cas, nous constatons ainsi que les métriques CPU, MEM et TX sont fortement corrélées entre elles et avec le premier facteur avec un coefficient de corrélation entre chaque paire très proche de 1. Nous constatons également une corrélation nulle entre RX et le reste des métriques. En revanche, cette dernière est fortement corrélé avec le deuxième facteur. Nous attirons aussi l'attention sur le fait que les deux premières composantes principales restaurent 93.71%(68.67% + 25.04%) de la variance totale des données initiales, ce qui indique que 2 composantes sont suffisantes pour représenter les données mesurées.

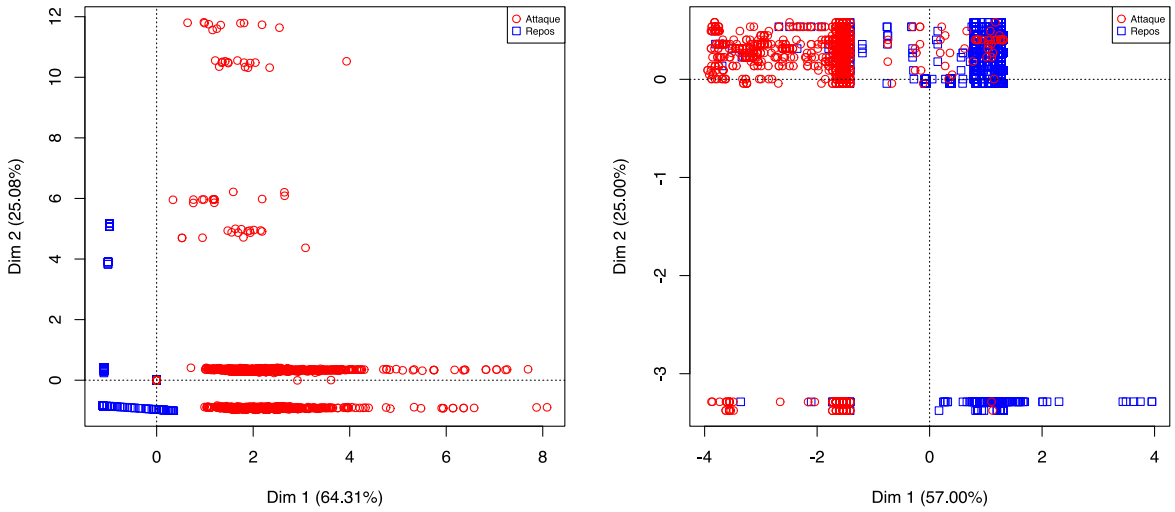
L'annexe A (figure A.4 à figure A.5) presente l'ensemble des cercles de corrélation des ACP réalisées sur l'ensemble des expérimentations impliquant une attaque UDP Flood. La lecture des corrélations de ces paramètres indique que les comportements des deux *botcloud* réalisant une attaque UDP Flood sont similaires, à deux différences mineures près. La première différence porte sur la métrique RX qui est fortement et négativement corrélé avec TX et CPU dans le cas de Hybrid_V1.0 (le coefficient de corrélation est quasiment égal à -1 pour toutes les expérimentations). En revanche, dans le cas de Kaiten, le coefficient de corrélation de RX avec les autres paramètres est toujours proche de 0, indiquant donc une absence de corrélation. Ceci s'explique par l'arrêt de l'envoi de traffic de signalisation de la part du *botmaster* pendant l'attaque dans le cas de Hybrid_V1.0 et sa continuité dans le cas de Kaiten. La deuxième différence réside dans l'utilisation de la mémoire. En effet, la métrique MEM est fortement et positivement corrélée avec CPU et TX dans le cas de Kaiten (coefficient de corrélation très proche de 1 pour toutes les expérimentations), alors qu'il est très proche ou égal à 0 dans le cas de Hybrid_V1.0. Cette différence est expliqué par une meilleure gestion de la mémoire de la part de Kaiten car Hybrid_V1.0 ne libère pas complètement la mémoire après son utilisation.

La figure 3.3.b représente le cercle de corrélation de l'ACP réalisée sur le *botcloud* Hybrid_V1.0 réalisant l'attaque TCP SYN Flood avec un débit de 290 Paquets/s. Nous constatons que les deux premiers facteurs restaurent 82 % de la variance des données initiales, ce qui est, comme pour le cas précédent, suffisant pour les représenter. Concernant les métriques, nous notons une forte corrélation de TX et RX et le premier facteur avec un coefficient de corrélation égal à 1. Toutefois, ces derniers sont parfaitement et négativement corrélés avec la métrique CPU avec un coefficient de corrélation égal à -1. La métrique MEM présente une corrélation nulle avec ces trois derniers paramètres, ainsi qu'une corrélation parfaite et positive avec le deuxième facteur (coefficient de corrélation égal à 1). L'Annexe A (figure A.6) comprend également tous les cercles de corrélation des ACP réalisées sur l'ensemble des expérimentations impliquant une attaque TCP SYN Flood. Quelque soit le débit d'attaque, les paramètres agissent comme pour le cas de l'attaque à 290 P/s décrit ci-dessus. Ainsi, nous concluons que le comportement système global d'un *botcloud* est peu variable avec le changement du débit d'attaque.

3.3.3 Comportement des individus

Dans la précédente section, nous avons décrit, via l'ACP, le comportement des métriques mesurées et leurs corrélations. Dans cette section, nous décrivons le comportement des individus. Comme il est décrit dans la section 3.2.4.2, l'ACP crée de nouveaux individus, où chacun de ces derniers représente une combinaison linéaire des quatre paramètres collectés dans le même instant donné.

La figure 3.4.a et la figure 3.4.b représentent les projections des individus des deux expérimentations décrites dans les sections précédentes, sur les deux premiers facteurs résultat de leurs ACP respectives. Dans ces graphiques, un individu prélevé en période d'attaque est représenté par un rond rouge, et un individu prélevé en période de repos est représenté par un carré bleu. Pour toutes les expérimentations, nous notons une séparation claire entre la période d'attaque et la période de repos matérialisée par l'axe représentant le deuxième facteur. Pour le cas des expérimentations impliquant une attaque UDP Flood,



(a) (b)

FIGURE 3.4 – Projections des individus des ACP réalisées sur ; (a) *botnet* Kaiten, attaque UDP Flood à 56Mb/s ; (b) *botnet* Hybrid_V1.0, attaqueTCP SYN Flood à 290 Paquets/s

les individus représentant la période d'attaque sont situés à droite du deuxième axe et ceux représentant la période de repos à sa gauche. En revanche cette position est inversé dans le cas de l'attaque TCP SYN Flood. Les quelques individus de la période d'attaque se trouvant à gauche du deuxième axe ou ceux de la période de repos se trouvant à sa droite dans le cas d'une attaque UDP Flood (ou inversement dans le cas d'une attaque TCP SYN Flood), représentent les phases de transition des paramètres système, à savoir de la phase de repos vers la phase d'attaque et vice versa. L'annexe A (figure A.7 à figure A.9) présente les projections d'individus résultant des ACP réalisées sur toutes les expérimentations.

3.4 Caractérisation du *botcloud* durant la période d'attaque

Dans la Section 3.3 nous avons décrit le comportement système global d'un *botcloud* et des VMs qui le composent. Dans cette section, nous nous concentrons sur la seule phase d'attaque afin de comprendre de façon plus approfondie le comportement des différentes métriques étudiées. En d'autres termes, contrairement aux résultats présentés précédemment qui concernent toutes les phases du scénario d'expérimentation, nous avons réalisé des ACP sur les données collectées durant les seules périodes d'attaque et ce pour toutes les expérimentations réalisées. Afin de comprendre plus finement le comportement des métriques étudiées nous nous intéressons à leurs contributions dans la construction des composantes principales. L'objectif de cette étude et d'identifier les potentiels invariants

comportementaux d'un *botcloud* lorsque le type d'attaque, son débit ou l'implémentation utilisée varient.

3.4.1 Impact du débit d'attaque

Dans cette section, nous étudions l'impact du débit d'attaque sur le comportement système d'une implémentation de *botcloud*. Pour ce faire, nous considérons les cinq débits d'attaques couverts dans notre plage de mesure qui sont choisis pour induire une faible empreinte système, pour le plus faible débit, jusqu'à une empreinte forte et visible, pour le plus fort débit.

3.4.1.1 Cas de l'implémentation Kaiten

Le tableau 3.3 décrit l'évolution des contributions des paramètres CPU, MEM, TX et RX dans la construction des facteurs principaux résultats des ACP réalisées sur les données collectées sur l'ensemble des VMs du *botcloud* Kaiten réalisant les attaques UDP Flood. Il met au premier plan les similitudes entre les contributions des différentes métriques, mais montre également les détails de la construction de chaque facteur. Si on considère les facteurs un par un, on note pour les cinq expérimentations que le premier facteur est caractérisé par (1) une grande contribution de TX avec 47.34 % de moyenne. Cette dernière est stable avec le changement du débit d'attaque avec un écart type de 1% ; (2) une grande contribution de CPU avec 40.92 % de moyenne ; (3) une contribution insignifiante (très proche de 0) de MEM et (4) une petite contribution de RX. Le deuxième facteur est caractérisé par (1) une très grande contribution de RX avec 73.88 % de moyenne et qui est décroissante avec l'augmentation du débit d'attaque ; (2) une contribution très variable de CPU selon le débit d'attaque (le seul cas constaté) ; (3) une petite contribution de TX et (4) une contribution insignifiante de MEM. Le troisième facteur est caractérisé par (1) une grande contribution de TX avec 47.64 % de moyenne ; (2) une grande contribution de CPU avec 37.95 % de moyenne. Cette dernière décroit avec l'augmentation du débit d'attaque ; (3) une petite contribution de RX qui est croissante avec l'augmentation du débit d'attaque et (4) une contribution insignifiante de MEM. Enfin, le quatrième facteur est caractérisé par une contribution insignifiante des quatre paramètres.

Le tableau 3.3 montre ainsi que pour chaque facteur, la valeur de la moyenne calculée sur les pourcentages de contributions respectifs aux cinq débits d'attaques et pour chaque paramètre est très proche des valeurs initiales et son écart type associé est très faible aussi. Cela témoigne d'une forte similitude des pourcentages de contribution de ces paramètres dans la construction des facteurs. En d'autre terme, de façon générale, quelque soit le débit d'attaque, le comportement système du *botcloud* Kaiten est très peu variable. On note que le cas de l'expérimentation de l'attaque à 40 Mb/s représente une exception car certains pourcentages des contributions des paramètres sont différents de ceux des autres expérimentations. Nous attribuons cette différence à l'état opérationnel de notre infrastructure de test au moment des mesures, qui nous le rappelons est mutualisée et non isolée vis à vis de l'allocation de ressources.

	8 Mb/s	16 Mb/s	40 Mb/s	56 Mb/s	80 Mb/s	μ	σ
Fact. 1	49.85	51.38	17.66	51.97	33.76	40.92	15.02
Fact. 2	2.54e-02	2.23	69.56	1.41	32.34	21.11	30.25
Fact. 3	50.12	46.38	12.77	46.60	33.88	37.95	15.36
Fact. 4	2.61e-65	9.31e-68	0	0	0	5.23e-66	1.16e-65

(a) Contribution de CPU dans la construction des vecteurs propres de l'ACP

Fact. 1	1.23e-30	1.23e-30	7.88e-29	4.43e-29	2.77e-28	8.05e-29	1.14e-28
Fact. 2	2.03e-29	8.13e-31	9.98e-29	5.69e-27	1.49e-27	1.46e-27	2.44e-27
Fact. 3	4.93e-30	1.23e-30	1.77e-28	3.08e-29	4.44e-28	1.31e-28	1.88e-28
Fact. 4	6.67e-32	9.30e-34	1.41e-33	1.33e-31	1.43e-33	4.06e-32	5.88e-32

(b) Contribution de MEM dans la construction des vecteurs propres de l'ACP

Fact. 1	48.53	47.53	47.68	45.09	47.90	47.34	1.31
Fact. 2	2.67	8.85	1.47e-01	13.19	9.56e-02	4.99	5.80
Fact. 3	48.78	43.60	52.16	41.70	51.99	47.64	4.80
Fact. 4	8.22e-64	1.14e-65	2.79e-64	1.10e-60	6.36e-64	2.20e-61	4.91e-61

(c) Contribution de TX dans la construction des vecteurs propres de l'ACP

Fact. 1	1.61	1.08	34.65	2.92	18.32	11.71	14.68
Fact. 2	97.30	88.91	30.28	85.38	67.55	73.88	26.68
Fact. 3	1.08	10.00	35.06	11.69	14.11	14.38	12.56
Fact. 4	5.13e-65	2.58e-65	3.93e-63	7.36e-60	3.24e-62	1.47e-60	3.28e-60

(d) Contribution de RX dans la construction des vecteurs propres de l'ACP

TABLE 3.3 – Contribution des paramètres dans la construction des vecteurs propres des ACP réalisées sur les données de l'expérimentation impliquant Kaiten réalisant les attaques UDP Flood

3.4.1.2 Cas de l'implémentation Hybrid_V1.0

Comme pour le cas des expérimentations impliquant Kaiten, le tableau 3.4 représente les contributions des métriques CPU, MEM, TX et RX dans la construction des facteurs propres, résultat des ACP réalisées sur les données collectées sur l'ensemble des VMs du *botcloud* Hybrid_V1.0 réalisant les attaques UDP Flood. Si l'on considère un par un les facteurs obtenus par les ACP, on constate que le premier facteur est toujours caractérisé par (1) une grande contribution de TX avec 48.93 % de moyenne. Cette contribution est stable avec le changement du débit d'attaque avec un écart type de 5 % ; (2) une grande contribution de CPU avec 39.24 % de moyenne. Cette dernière est croissante avec l'augmentation du débit d'attaque ; (3) une petite contribution de MEM qui décroit avec l'augmentation du débit d'attaque et (4) une contribution nulle de RX. Le deuxième facteur est caractérisé par (1) une très grande contribution de MEM qui est fortement croissante avec l'augmentation du débit d'attaque, atteignant les 92 % ; (2) une contribution moyenne de CPU, qui décroit avec l'augmentation du débit d'attaque ; (3) une petite contribution de TX et (4) une contribution nulle de RX. Le troisième facteur est caractérisé par (1) une grande contribution de TX avec une moyenne de 47.87 %. Cette contribution est stable avec le changement du débit d'attaque avec un écart type de 5 % ;

	8 Mb/s	16 Mb/s	40 Mb/s	56 Mb/s	80 Mb/s	μ	σ
Fact. 1	29.68	32.21	36.18	47.64	50.52	39.24	9.32
Fact. 2	38.92	37.38	28.86	1.05	2.77	21.79	18.56
Fact. 3	31.39	30.39	34.95	51.302	46.69	38.94	9.47
Fact. 4	0	0	0	0	0	0	0

(a) Contribution de CPU dans la construction des vecteurs propres de l'ACP

Fact. 1	15.34	25.43	9.68	8.45	0.14	11.80	9.35
Fact. 2	60.86	62.01	70.42	89.45	92.32	75.01	14.98
Fact. 3	23.78	12.54	19.88	2.09	7.52	13.16	8.84
Fact. 4	0	0	0	0	0	0	0

(b) Contribution de MEM dans la construction des vecteurs propres de l'ACP

Fact. 1	54.96	42.34	54.13	43.90	49.32	48.93	5.75
Fact. 2	0.21	0.59	0.70	9.49	4.89	3.17	4.01
Fact. 3	44.81	57.06	45.15	46.60	45.77	47.87	5.17
Fact. 4	0	0	0	0	0	0	0

(c) Contribution de TX dans la construction des vecteurs propres de l'ACP

Fact. 1	0	0	0	0	0	0	0
Fact. 2	0	0	0	0	0	0	0
Fact. 3	0	0	0	0	0	0	0
Fact. 4	0	0	0	0	0	0	0

(d) Contribution de RX dans la construction des vecteurs propres de l'ACP

TABLE 3.4 – Contributions des paramètres dans la construction des facteurs résultats des l'ACP réalisées sur les données des expérimentations impliquant le Hybrid_V1.0 réalisant les attaques UDP Flood

(2) Une grande contribution CPU avec 38 % de moyenne. Cette dernière est croissante avec l'augmentation du débit d'attaque ; (3) une petite contribution de MEM et (4) une contribution nulle de RX. Le quatrième facteur représente un facteur nul.

Ainsi, pour ce cas également, nous notons une forte similitude des pourcentages de contribution des paramètres dans la construction des quatre facteurs, et ce pour les cinq débits d'attaque. En effet, les moyennes des contributions sur l'ensemble des facteurs et concernant chacun des paramètres sont très proches des valeurs initiales et les écarts type associés sont très petits.

Le tableau 3.5 décrit les contributions des paramètres dans la construction des facteurs, résultat des ACP réalisées sur les données collectées sur l'ensemble des VMs du *botcloud* Hybrid_V1.0 réalisant les attaques TCP SYN Flood. Comme pour les deux précédents cas, la valeur de la moyenne calculée pour chaque facteur sur les cinq débits d'attaque et pour chacun des paramètres est très proche des valeurs initiales. De même, les écarts type de ces dernières sont très proches de 0 et distribués sur l'intervalle $[0.01, 4.78]$. Concernant les facteurs obtenus via les ACP, nous notons que le premier facteur est caractérisé par une grande contribution de CPU d'une moyenne de 24.14%. Ces valeurs de contribution baissent avec la croissance du débit d'attaque. Les contributions de TX et RX sont exac-

	6500 p/s	8000 p/s	12500 p/s	20000 p/s	μ	σ
Fact. 1	28.52	27.22	17.98	22.86	24.14	4.76
Fact. 2	2.11e-03	2.25e-03	1.19e-05	3.56	0.87	1.74
Fact. 3	71.47	72.77	82.01	73.57	74.95	4.78
Fact. 4	5.59e-05	1.32e-05	1.21e-05	3.11e-08	2.03e-05	2.44e-05

(a) Contribution de CPU dans la construction des vecteurs propres de l'ACP

Fact. 1	0.001	4.26e-04	1.07e-05	0.03	0.007	0.01
Fact. 2	99.99	99.99	99.99	95.76	98.93	2.11
Fact. 3	7.51e-04	0.004	2.85e-05	4.19	1.04	2.09
Fact. 4	1.40e-04	9.72e-05	5.53e-05	1.94e-07	7.31e-05	5.96e-05

(b) Contribution de MEM dans la construction des vecteurs propres de l'ACP

Fact. 1	35.72	36.38	41.00	38.55	37.91	2.38
Fact. 2	5.53e-04	9.42e-04	2.42e-06	0.33	0.08	0.16
Fact. 3	14.30	13.63	9.00	11.11	12.01	2.43
Fact. 4	49.96	49.98	49.98	49.99	49.98	0.01

(c) Contribution de TX dans la construction des vecteurs propres de l'ACP

Fact. 1	35.74	36.39	41.01	38.55	37.92	2.38
Fact. 2	4.58e-05	1.98e-03	8.02e-05	0.33	0.08	0.16
Fact. 3	14.21	13.58	8.97	11.11	11.96	2.40
Fact. 4	50.03	50.01	50.01	50.00	50.01	0.01

(d) Contribution de RX dans la construction des vecteurs propres de l'ACP

TABLE 3.5 – Contribution des métriques dans la construction des vecteurs propres de l'ACP réalisée sur les données de l'expérimentation impliquant le Hybrid_V1 et l'attaque TCP SYN Flood

tement identiques avec respectivement 37.91 % et 37.92% de moyenne. Ces dernières sont croissantes lors de la croissance du débit d'attaque. Finalement, MEM a une contribution négligeable. Le deuxième facteur est caractérisé par une très grande contribution de MEM avec 98.93%. Le pourcentage restant est partagé entre CPU, TX et RX. Le troisième facteur est caractérisé par la grande contribution de CPU avec 74.95% de moyenne. Cette contribution est croissante avec l'augmentation du débit d'attaque. TX et RX ont des contributions moyennes et quasi identiques avec 12.01% et 11.96% de moyenne. Ces dernières sont décroissantes en fonction de la croissance du débit d'attaque. Finalement, MEM a une contribution négligeable de 1.04% de moyenne. Le quatrième facteur est caractérisé par la domination de TX et RX avec 49.98% et 50.01% de moyenne. CPU et MEM ont une contribution négligeable.

Pour conclure, d'après les expérimentations réalisées sur les deux types d'attaques, nous notons que quelque soit le débit d'attaque, le comportement système du *botcloud* Hybrid_V1.0 est très peu variable.

3.4.2 Impact de l'implémentation logicielle du *botcloud*

Etant donné que nous possédons pas de données relatives au *botcloud* Kaiten réalisant des attaques TCP SYN Flood, dans cette section nous nous concentrons uniquement sur

les attaques de type UDP Flood.

En se basant sur les tableaux 3.3 et 3.4, nous remarquons que les comportements des deux *botcloud* sont très proches malgré leur source et leur implémentation différente. En effet, l'implication des paramètres dans la construction des facteurs est très similaire pour les deux implémentations Kaiten et Hybrid_V1.0. Afin de mettre en exergue ces similarités, le tableau 3.6 résume les similitudes et les quelques différences entre ces derniers, en les comparant qualitativement facteur par facteur et paramètre par paramètre. Pour chaque paramètre, nous avons attribué une note ; à savoir + pour similaire et – pour différent, afin de déterminer si le comportement de la contribution de ce paramètre est le même pour les deux implémentations de *botcloud* ou non. Pour l'ensemble des seize métriques étudiées, nous avons constaté que 13 sont similaires (+) alors que seulement 3 sont différentes (–). Cela démontre que la contribution des paramètres dans la construction des facteurs est très peu variable. En d'autres termes, nous concluons que, le comportement système d'un *botcloud* réalisant une attaque UDP Flood est peu variable et cela indépendamment du débit d'attaque ou de l'implémentation logicielle du *botcloud*, caractérisée par son langage de programmation, sa structure algorithmique et son protocole de signalisation.

3.5 Conclusion

L'étude détaillée du comportement d'un *botcloud* représente une étape clé dans le la conception d'un système de détection. Dans ce chapitre nous avons décrit et expliqué une première caractérisation globale du comportement système d'un *botcloud* réalisant des attaques DDoS de type Flooding ; TCP SYN Flood et UDP Flood. Ainsi, un *botcloud* dédié à l'attaque est caractérisé par la forte similitude du comportement des VMs le composant, en attestent les distributions statistiques des métriques MEM, TX et RX qui sont caractérisées par une distribution bi-modale avec une grande concentration autour des modes. A l'inverse, celle du CPU varie et augmente selon le débit d'attaque, mais elle reste faible dans la majorité des cas.

Par la méthode d'ACP, nous avons décrit et mis en évidence le comportement et les corrélations des paramètres systèmes d'un *botcloud*. Par la projection des individus résultats de ces ACP, nous avons pu détecter et séparer les périodes d'attaque de celles de repos pour toutes les expérimentations menées. Par la suite, nous nous somme intéressé à la seule phase d'attaque et nous avons produit une caractérisation approfondie de cette dernière. Nous avons démontré que le comportement système d'un *botcloud* est quasi constant quelque soit le débit d'attaque. De plus, en utilisant deux *botclouds* différents, nous avons démontré qu'il est également très peu variable quelque soit l'implémentation utilisé.

Etant données ces caractéristiques fortes du comportement système d'un *botcloud* durant la phase d'attaque, nous pouvons les exploiter afin de construire une référence qui serve de signature pour la détection de ce type d'attaque et c'est le sujet abordé dans le chapitre suivant.

		Hybrid_V1.0	Kaiten	Similarité
Fact. 1	**CPU**	- Grande contribution - Contribution croissante avec l'augmentation du débit d'attaque	- Grande contribution	+
	MEM	- Petite contribution - Décroissante avec l'augmentation du débit d'attaque	- Très petite contribution	+
	TX	- Grande contribution - Stable avec le changement du débit	- Grande contribution - Stable avec le changement du débit	+
	RX	- Contribution nulle	- Petite contribution	+
Fact. 2	**CPU**	- Contribution moyenne - Décroissante avec l'augmentation du débit d'attaque	- Contribution très variable selon le débit d'attaque	–
	MEM	- Très grande contribution	- Très petite contribution	–
	TX	- Très petite contribution - Croissante légèrement avec l'augmentation du débit d'attaque	- Petite contribution	+
	RX	- Contribution nulle	- Très grande contribution	–
Fact. 3	**CPU**	- Grande Contribution - Croissante avec l'augmentation du débit d'attaque	- Grande Contribution - Décroissante avec l'augmentation du débit d'attaque	+
	MEM	- Petite contribution	- Très petite contribution	+
	TX	- Grande contribution - Stable avec le changement du débit	- Grande contribution - Décroissante avec l'augmentation du débit d'attaque	+
	RX	- Contribution nulle	- Petite contribution - Croissante avec l'augmentation du débit d'attaque	+
Fact. 4	**CPU**	- Contribution nulle	- Contributions insignifiante	+
	MEM	- Contribution nulle	- Contributions insignifiante	+
	TX	- Contribution nulle	- Contributions insignifiante	+
	RX	- Contribution nulle	- Contributions insignifiante	+

TABLE 3.6 – Comparaison qualitative des contributions des métriques dans Hybrid_V1.0 et Kaiten

4

Vers une approche de détection à la source des *botclouds*

Sommaire

4.1 Introduction

Il existe de nombreuses propositions qui visent à détecter des attaques produites dans le contexte du *cloud computing*. Cependant, la majorité de ces dernières visent à protéger le *cloud* et ses clients des attaques internes et externes mais elles ignorent le fait que le *cloud* puisse lui même être utilisé comme support pour ces attaques à destination de tout tiers connecté à l'Internet.

En outre, dans la cadre des attaques de déni de service distribuées, bien qu'une attaque puisse être détectée et stoppée au niveau de sa cible, elle peut causer de nombreux dégâts sur le chemin parcouru jusqu'à celle-ci, d'autant plus que son débit est conséquent. C'est pourquoi, afin de limiter toute forme de dommages, directs ou collatéraux, induits par ce type d'attaque, nous proposons une approche de détection à la source. Cette stratégie repose sur le fait qu'à contrario des *botnets*, les machines sollicités pour la mise en oeuvre

d'une attaque sont toutes gérées par un (ou quelques) opérateur(s) de services *cloud*. Ainsi, si par le biais du *cloud*, l'attaquant dispose d'une plate-forme prête à l'emploi pour la mise en oeuvre d'attaques, sa gestion centralisée par un unique opérateur, offre à ce dernier un meilleur contrôle des machines qu'il héberge. On ajoute enfin que pour un opérateur de *cloud* être en mesure de détecter et endiguer les activités malveillantes de son infrastructure est essentiel pour maintenir sa compétitivité et sa crédibilité face à la concurrence et une détection à la source s'avère essentielle dans ce cadre.

Ainsi, nous présenterons dans ce chapitre l'approche de détection à la source des attaques de déni de service distribuées que nous proposons. Dans un premier temps, nous présenterons la méthodologie suivie dans la construction des références (signatures) d'attaque. Par la suite nous détaillerons l'algorithme de détection qui utilise ces références. Finalement nous présenterons une évaluation de la performance de cet algorithme ainsi qu'une évaluation de sa complexité algorithmique.

4.2 Etat de l'art sur les approche de détection fondées sur l'ACP

De nombreuses contributions scientifiques sont fondées sur l'utilisation de l'ACP pour la détection d'intrusions. La majorité des travaux utilise cette méthode dans une approche comportementale pour la détection de données aberrantes. En effet, les données respectives aux attaques et intrusions sont des données qui se distinguent par rapport à l'ensemble des données du comportement normal, apparaissant ainsi comme aberrantes.

Ling *et al.* [118] proposent un IDS distribué qui repose sur cette méthodologie. Ce système est composé de deux parties : (1) un ensemble de noeuds *(Monitors)* ayant un rôle de premier filtre. Chaque noeud stocke une fenêtre glissante des données recueillies. Pour chaque glissement de la fenêtre, le noeud applique une ACP sur ses données. Le nouvel espace (de p dimensions) résultat de cette ACP se divise en deux sous-espaces ; un premier sous espace de k dimensions, représenté par les k premières composantes principales et qui contiennent la majorité de la variance des données initiales ainsi qu'un un autre sous-espace de $p - k$ dimensions. Selon les travaux de Anukool *et al.* [119], les données qui représentent de potentielles intrusions dans les données initiales, sont clairement aberrantes après projection sur ce deuxième sous espace. Les données respectives à ces potentielles intrusions sont remontées au deuxième composant appelé (2) *Coordinator*. La théorie de la perturbation stochastique [120] est ensuite utilisée par ce dernier afin d'analyser les différentes alertes et décider de l'occurrence d'une attaque ou intrusion.

Shyu *et al.* [121] proposent aussi un IDS comportemental. Ce dernier applique une ACP sur les donnés surveillées et utilise deux fonctions de détection. La première est appliquée sur l'espace composé par les premières composantes principales qui totalisent environ 50% de la variance initiale afin de détecter les données aberrantes. La deuxième étant sur les dernières composantes principales qui totalisent moins de 20% de la variance initiale afin de détecter les données qui possèdent une structure de corrélation différente de l'ensemble des données. Dans les deux cas, la distance de Mahalanobis [122] est utilisée pour la détection des données aberrantes. A la différence de la distance euclidienne qui donne le même poids à toutes les variables utilisées dans le calcul, la distance de Mahalanobis

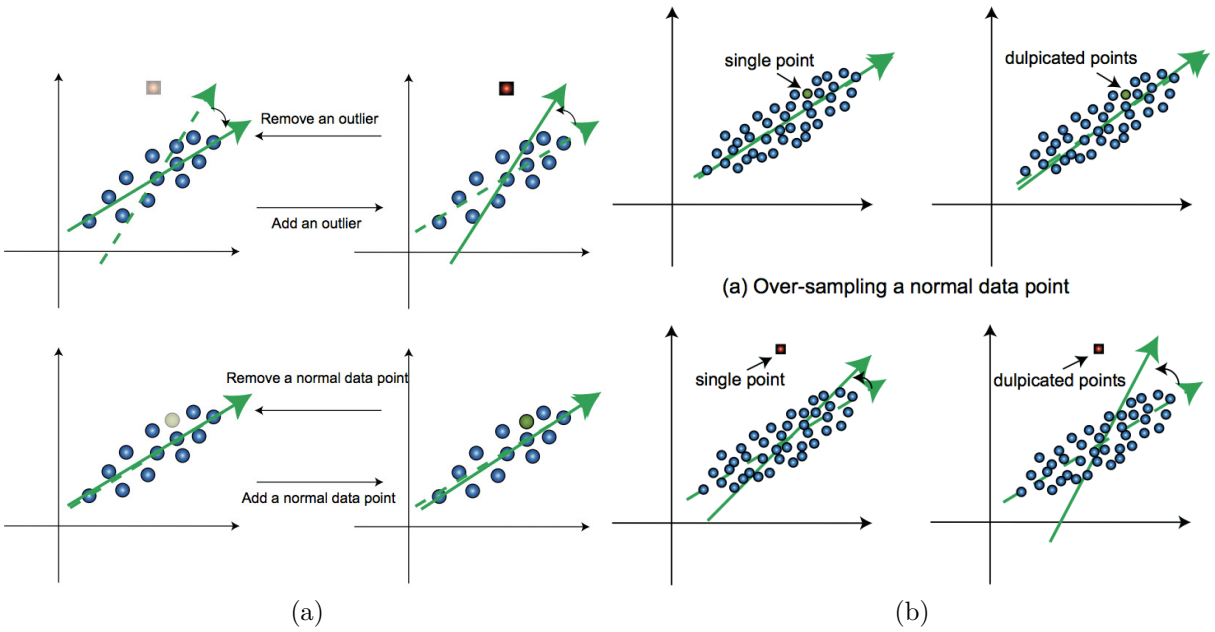


FIGURE 4.1 – Les effets du (a) rajout et suppression (b) sur-échantillonnage ; des données légitimes et aberrantes sur la direction du vecteur propre[7]

considère la variance des variables et donne un poids à chacune dans le calcul. Cette distance est calculée par rapport à une moyenne définie par apprentissage et comparée à un seuil fixé en paramètre.

Lee *et al.* [7] proposent un IDS comportemental qui repose sur le fait que l'ACP est très sensible aux données aberrantes. L'idée de l'approche repose sur la direction du premier vecteur propre et le fait que des données aberrantes peuvent changer cette direction. En effet, comme décrit par la figure 4.1, dupliquer des données de comportement légitime n'affecte pas la direction du premier vecteur propre. Cependant, dupliquer des données aberrantes modifie clairement cette direction. La duplication des données se fait par un sur-échantillonnage[51] effectué sur l'ensemble des données initiales. Le processus de détection se fait en comparant l'angle dessiné par les deux positions du premier vecteur propre avant et après le sur-échantillonnage, par rapport à un seuil prédéfini.

Il existe de nombreuses autres études qui utilisent l'ACP dans un contexte de sécurité : Huizhong *et al.* [123] qui propose un système de défense contre les DDoS qui extrait les caractéristiques du trafic réseau en analysant la dépendance intrinsèque selon plusieurs valeurs d'attributs. La détection de l'occurrence de l'attaque repose sur ces dépendences. Brauckhoff *et al.* [124] propose une version modifiée de l'ACP appliqué à un IDS centralisé.

Au delà, d'autres approches utilisent l'ACP dans d'autres cadres applicatifs tels que la réduction de données et combinent son utilisation avec d'autres méthodes statistiques. Par exemple, Bouzida *et al.* [125] utilisent l'ACP puis les arbres de décision [126] et la

[51]Le sur-échantillonnage représente une technique d'échantillonnage qui consiste à échantillonner les données à une fréquence bien plus élevée que la fréquence donnée par le théorème de Shannon

méthode des K plus proches voisins [127]. L'approche proposée dans [128] utilise l'ACP et les réseaux de neurones [129]. Enfin, les auteurs de [130] utilisent l'ACP et les machines a vecteurs de support [131].

Dans l'ensemble des travaux présentés, les caractéristiques de l'ACP et plus spécifiquement des vecteurs propres de cette ACP sont utilisés dans le processus de détection. Que se soit pour la réduction des dimensions ou pour la détection, dans la majorité des travaux c'est une partie de ces vecteurs propres qui est utilisée. Dans la suite de se chapitre, nous présenterons l'approche que nous proposons pour la détection des attaques DDoS sur la base de mesures du comportement système d'un *botcloud*. Cette dernière repose sur l'ACP mais contrairement aux autres méthodes présentées, nous utilisons tous les vecteurs propres obtenus car le nombre de dimensions initial des données est faible et cela permet d'éviter toute perte d'information.

4.3 Une approche par ACP pour la détection à la source des *botclouds*

4.3.1 Etablissement d'une référence pour les attaques DDoS

Dans cette section, nous présenterons la méthodologie menée pour la construction d'une référence pour les attaques DDoS de type UDP Flood et TCP SYN Flood ainsi que les résultats que nous avons obtenus.

4.3.1.1 Motivations pour une approche à base de signature

Comme décrit dans la Section 4.2 les IDS ayant une approche comportementale reposent sur la modélisation d'un comportement normal et sur le fait que toute activité qui dévie de ce comportement soit considérée comme une potentielle intrusion ou attaque.

Une infrastructure de *cloud* public offrant des service IaaS héberge un très grand nombre de VM. Par exemple, Amazon EC2 comprend plus d'un demi million de serveurs physiques [132] et selon une étude faite en 2009 [133] sur une région d'Amazon EC2 (*us-east-1*) surveillée pendant 24 heures, 50242 requêtes de gestion de VM ont été effectuées, alors que l'adoption du *cloud* n'était qu'à son début. Ces chiffres montrent la taille colossale de ces infrastructures ainsi que la dynamique qu'elles subissent. Dans ce contexte, il apparait très difficile de modéliser un comportement que l'on pourrait qualifier de *normal* pour toutes les VM d'une telle infrastructure notamment car le comportement individuel de chacune est très hétérogène, à l'image de ses utilisateurs et de leurs besoins. Par conséquent, la considération d'une approche comportementale apparait comme un choix complexe pour la détection d'attaques opérées dans un *cloud* public.

A l'inverse, nous avons démontré dans le chapitre précédent qu'indépendamment du débit d'attaque et de l'implémentation du *botcloud* utilisée, le comportement système de ce dernier (et respectivement de ses VM) est peu variable. Ainsi, la construction d'une référence de ce comportement et la considération d'une approche à base de signature représente le meilleur choix.

4.3.1.2 Construction de la référence

	REF	H_8	H_16	H_40	H_56	H_80	K_8	K_16	K_40	K_56	K_80
REF	0.00	1.26	1.23	1.27	1.27	1.34	1.22	1.24	1.40	1.26	1.27
H_8	1.26	0.00	0.22	0.17	0.75	0.77	1.56	1.53	1.45	1.55	1.42
H_16	1.23	0.22	0.00	0.28	0.71	0.80	1.55	1.52	1.45	1.54	1.42
H_40	1.27	0.17	0.28	0.00	0.61	0.60	1.52	1.49	1.48	1.51	1.42
H_56	1.27	0.75	0.71	0.61	0.00	0.35	1.43	1.42	1.68	1.42	1.54
H_80	1.34	0.77	0.80	0.60	0.35	0.00	1.45	1.44	1.65	1.44	1.52
K_8	1.22	1.56	1.55	1.52	1.43	1.45	0.00	0.29	1.24	0.33	0.74
K_16	1.24	1.53	1.52	1.49	1.42	1.44	0.29	0.00	1.10	0.11	0.64
K_40	1.40	1.45	1.45	1.48	1.68	1.65	1.24	1.10	0.00	1.10	0.54
K_56	1.26	1.55	1.54	1.51	1.42	1.44	0.33	0.11	1.10	0.00	0.66
K_80	1.27	1.42	1.42	1.42	1.54	1.52	0.74	0.64	0.54	0.66	0.00

TABLE 4.1 – Matrice de dissimilarité des activités des différents *botclouds*réalisant les attaques UDP Flood (H_ : Hybrid_V1.0 ; K_ : Kaiten)

Dans le chapitre précédent, nous avons montré l'indépendance relative du comportement système d'un *botcloud* face à son débit d'attaque et son implémentation logicielle en étudiant et comparant les contributions des métriques dans la construction des facteurs. Afin de valider la possible utilisation de ce type de données pour la construction d'un détecteur, nous introduisons la notion de dissimilarité, ou distance, entre ces derniers. Nous rappelons qu'étant donné une ACP réalisée sur une matrice de données, la matrice de vecteurs propres de taille $[p \times p]$, résultat de cette ACP, représente un espace factoriel à p dimensions qui représente et décrit l'activité des données composant cette matrice initiale. Ainsi la dissimilarité est déterminée par la distance entre les espaces factoriels. Plus proche de 0 est cette distance, plus proches sont les comportements des deux activités décrites par leur ACPs respectives. La mesure de distance entre deux matrices A et B est calculée via la norme de Frobenius [134] $||A - B||_F$ telle que décrite par l'équation 4.1. Selon cette équation et sachant que $p = 4$, la valeur maximale que peut posséder une distance (dissimilarité) entre deux espaces factoriels est de 4.

$$||A - B||_F = \sqrt{\sum_{i=1}^{p} \sum_{j=1}^{p} |(A - B)_{ij}|^2} \tag{4.1}$$

Le tableau 4.1 à l'exception de sa première ligne et sa première colonne, représente la matrice de dissimilarité des comportements de Hybrid_V1.0 et Kaiten réalisant l'attaque UDP Flood à différents débits. La figure 4.2 représente une boîte à moustaches de toutes les valeurs de dissimilarité obtenues. Ces dernières sont de façon générale petites et dispersées dans l'intervalle $[0.11, 1.68]$. La médiane correspond à 1.2 et les deux groupes séparés par cette dernière correspondent chacun à un *botcloud* réalisant les attaques à différents débits.

Cette similarité et celle des contributions confirment certes le fait qu'un *botcloud* a un comportement système peu variable indépendamment du débit d'attaque et de l'implémentation logicielle utilisée, mais elles montrent aussi que ces similatirités sont quantifiables dans l'espace factoriel résultant. Par conséquent nous définissons un espace factoriel

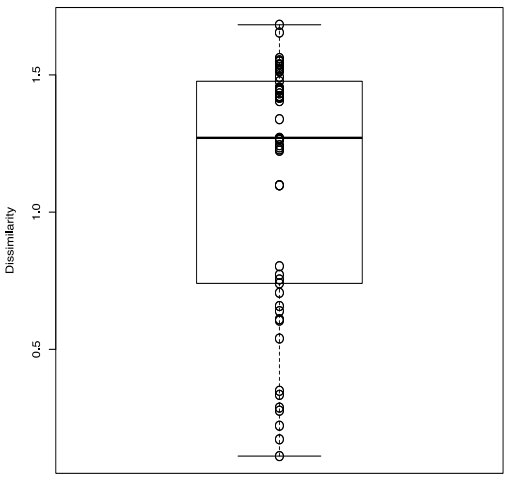


FIGURE 4.2 – Boîte à moustaches des dissimilarités des activités des différents *botclouds* réalisants les attaques UDP Flood

de référence, nommé R_{UDP}, pour les attaques de type UDP Flood. Ce dernier est défini par le centre de gravité de toutes les matrices de vecteurs propres relatives aux expérimentations des attaques UDP Flood et pour les deux *botcloud*. La Table 4.2 décrit cette référence. Suivant la même méthodologie, une référence R_{TCP} pour les attaques TCP est également proposée.

Le tableau 4.1 décrit également la dissimilarité de cette référence R_{UDP} (représentée par la première ligne et la première colonne) des autres espaces factoriels représentants chacune des attaques. On note que la plus grande valeur de dissimilarité entre R_{TCP} et les autres espaces factoriels représentés dans ce tableau est de 1.40 ce qui montre une relative concentration des espaces factoriels autour de cette référence.

	Fact. 1	Fact. 2	Fact. 3	Fact. 4
CPU	0.006067256	0.03234925	0.44652392	-1.310494e-51
MEM	0.164686557	0.43219073	0.03613016	2.035753e-34
TX	0.145103016	-0.03152999	0.63690376	3.752683e-49
RX	0.057728988	0.42302917	0.03150503	-9.960978e-49

TABLE 4.2 – Espace factoriel de référence pour les attaques de type UDP Flood

4.3.1.3 Comparaison avec une charge légitime

Afin de déterminer si l'empreinte système du *botcloud* pourrait être identifiable parmi ceux des autres *tenants*, nous confrontons l'espace factoriel de référence à d'autres activités

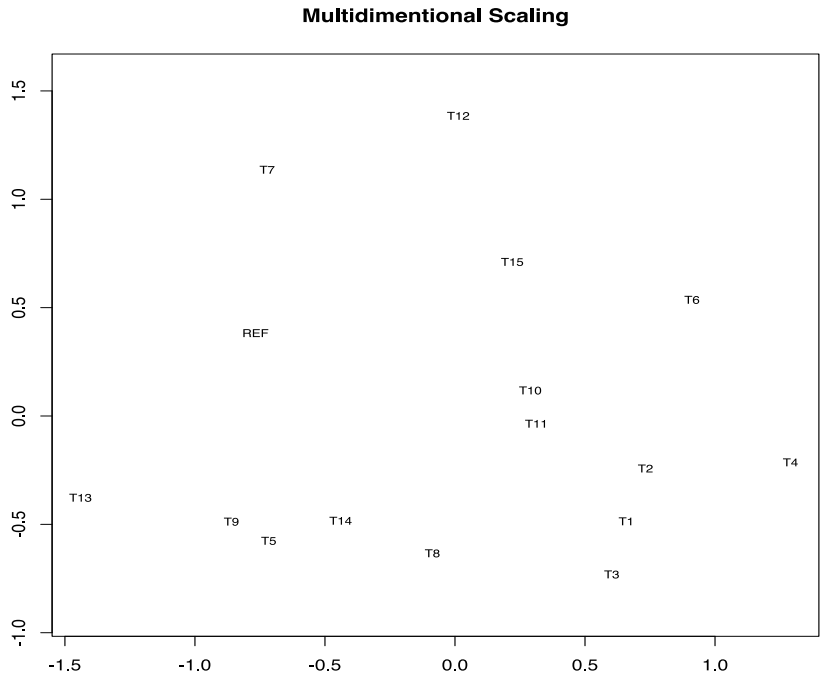


FIGURE 4.3 – MDS de 16 tenants (la Référence UDP + 15 autres tenants)

légitimes. Nous avons ainsi calculé la dissimilarité entre chaque paire d'un ensemble de 16 tenants, comprenant l'espace factoriel de référence de l'attaque UDP. Afin de produire une comparaison équitable des activités des tenants, nous avons sélectionné aléatoirement 15 tenants des 10 jeux de données impliquants l'expérimentation de l'attaque UDP Flood.

La figure 4.3 représente le positionnement multidimensionnel *(Multidimentional Scaling (MDS))* [135] de la matrice de dissimilarité de cet ensemble de tenant. MDS sert à représenter graphiquement une matrice de similarité en affectant une position à chaque individu dans un espace à N dimensions. Cette position est affectée de façon à ce que les similarités soient le plus proche possible de celle de la matrice initiale. En effet, la représentation dans un espace de dimension réduites induit certaines distorsions qui impactent la représentation des distances. MDS propose de faire en sorte que ces distances soient le plus proche possible de celles de la matrice initiale et on mesure le degré de correspondance des distances entre les points d'un graphique MDS et celles de la matrice initiale par une fonction de stress.

Sur la figure 4.3, nous constatons que la référence, représentée par le point REF, est bien loin des autres activités, ce qui prouve une forte dissimilarité, sauf pour T7 qui peut être un potentiel faux positif dans le cadre d'une approche de détection. La valeur de stress du MDS décrit dans ce cadre est de 0.14, ce qui montre que les dissimilarités sont bien représentées.

Comme pour le cas de UDP, nous avons calculé la dissimilarité entre chaque paire d'un ensemble de 16 tenants, comprenant l'espace factoriel référence de l'attaque TCP.

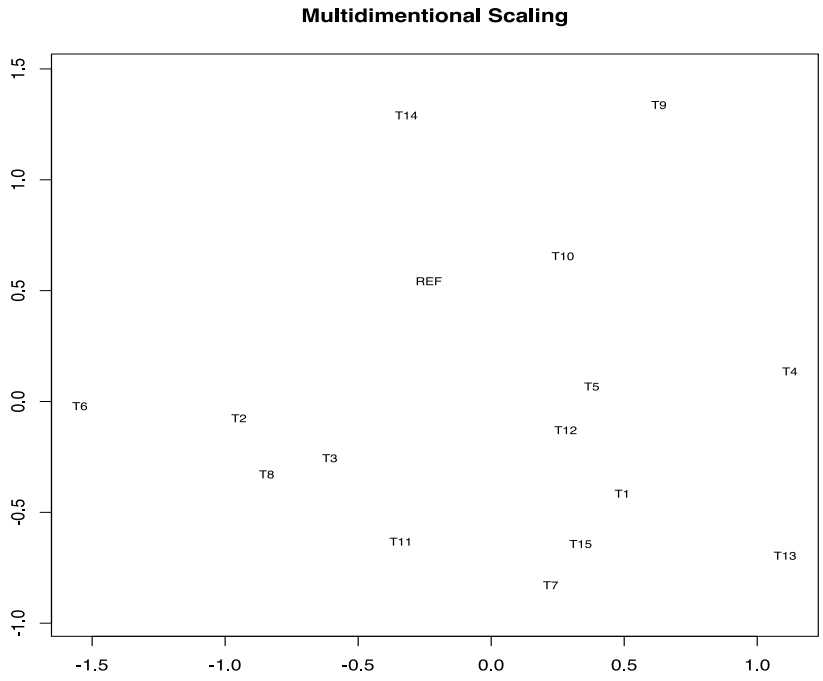


FIGURE 4.4 – MDS de 16 tenants (la Référence TCP + 15 autres tenants)

La figure 4.4 représente le MDS de la matrice de dissimilarité de cet ensemble de tenants. Comme pour le cas de UDP, nous avons pris au hasard 15 tenants des 5 jeux de données impliquant l'expérimentation de l'attaque TCP SYN Flood, afin de produire une comparaison équitable des activités des tenants. Dans ce cas également, nous remarquons la forte dissimilarité de la référence des autres *tenants*, sauf pour T10 qui pourrait représenter un potentiel faux positif.

Pour conclure, dans cette section nous avons démontré que l'activité d'un *botcloud* réalisant une attaque *Flooding*, peut être détectée et discriminée des activités légitimes grâce à son comportement système. Ainsi l'espace de référence déduit forme une signature de cette attaque qui peut être intégrée dans un algorithme de détection que nous décrivons dans la section suivante.

4.3.2 Proposition d'un algorithme de détection

Dans cette section nous présenterons en détails l'algorithme de détection proposé. La construction de cet algorithme repose principalement sur l'utilisation de l'espace factoriel de référence présenté ci-avant comme signature système d'une attaque DDoS. Pour déterminer si une attaque a eu lieu ou pas, nous appliquons une fonction de seuil à sa matrice de dissimilarité.

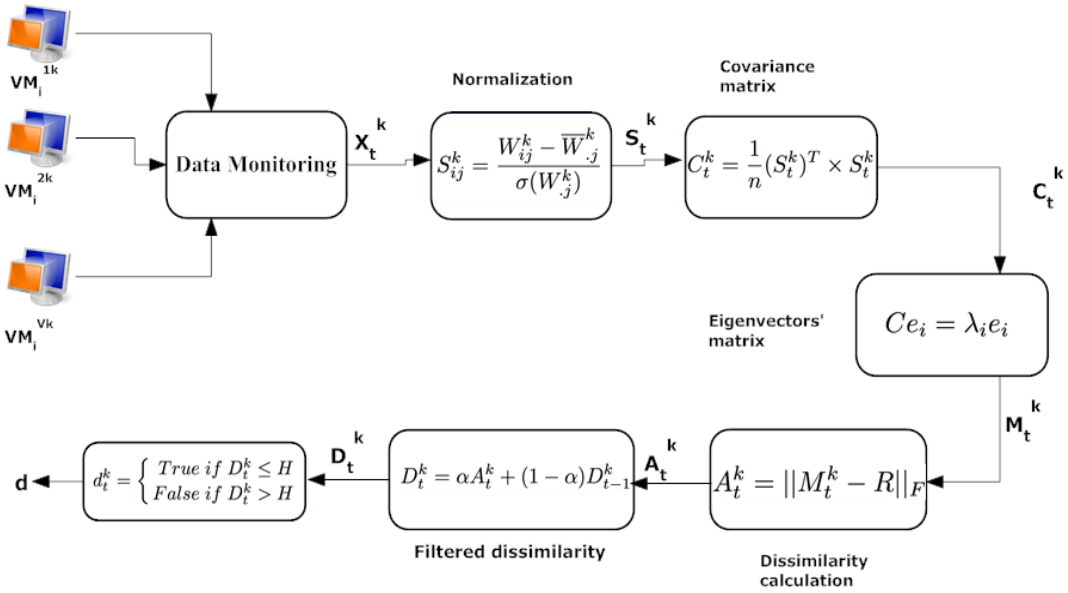


FIGURE 4.5 – Etapes de la détection

4.3.2.1 Formalisation du problème

La figure 4.5 décrit les étapes de l'algorithme de détection et la méthodologie suivie pour effectuer le processus de détection. Le tableau 4.3 décrit les différentes notations utilisées dans la suite de ce chapitre et du manuscrit.

La description détaillée de ces étapes est présentée ci-après. On note qu'elles ne sont données que pour le traitement d'un seul *tenant*. Cependant, l'algorithme gère tous les *tenant* de la même façon. En d'autre termes, ces étapes sont appliquées à tous les *tenant* simultanément.

La première étape du processus de détection représente la surveillance et la collecte de données. En effet, notre algorithme reçoit en entrée les données de toutes les VMs appartenant à un *tenant*. Par conséquent, il reçoit une matrice X_t^k de taille $[v \times p]$ ou v représente le nombre de VMs appartenant au *tenant* k. En d'autre termes, chaque ligne de la matrice représente les données relevées sur une VM du *tenant* k au temps t.

La deuxième étape consiste en la normalisation des données collectées par le calcul de S_t^k, la version centrée réduite de la matrice X_t^k, comme décrit par l'équation 4.2. La raison de la normalisation des données réside dans la différence d'échelle des données et de leur hétérogénéité. Par exemple, une consommation mémoire peut atteindre 8000 KB, le débit de transmission TX 10000 Kb/s pour une consommation CPU de 40%, et chacune de ses grandeur possède sa propre échelle de mesure et unité. Ainsi, transformer ces métriques en grandeurs comparables est nécessaire avant tout traitement.

$$S_{ij}^k = \frac{X_{ij}^k - \overline{X}_{.j}^k}{\sigma(X_{.j}^k)} \tag{4.2}$$

La troisième étape est le calcul de la matrice de variance-covariance, comme décrit

Symbole	Description	Symbole	Description
p	Nombre de métriques de la matrice	n	Nombre de ligne de la matrice (Nombre de VMs du *tenant*)
t	Index de temps	k	Index pour les *tenant*
X_t^k	Matrice $[n \times p]$ de données du *tenant* k au temps t	S_t^k	Normalisation de la matrice X_t^k
H	Seuil de détection	σ	Ecart type
e_{it}^k	$i^{ème}$ vecteur propre du *tenant* k calculé au temps t	λ_{it}^k	$i^{ème}$ valeur propre du *tenant* k calculée au temps t
M_t^k	Matrice $[p \times p]$ de vecteurs propre du *tenant* k calculée au temps t	C_t^k	Matrice $[p \times p]$ de covariance du *tenant* k calculée au temps t
d_t^k	Décision prise pour le *tenant* k au temps t	R	$[p \times p]$ Espace factoriel référence de l'attaque
A_t^k	Valeur de dissimilarité brute de M_t^k et R	D_t^k	valeur de dissimilarité filtrée avec EWMA de M_t^k et R

TABLE 4.3 – Notations utilisées pour la description de l'algorithme de détection

par l'équation 4.3. La matrice de variance-covariance sera utilisée pour le calcul des vecteurs propres $(e_{1t}^k, e_{2t}^k, .., e_{pt}^k)$, liés aux valeurs propres $(\lambda_{1t}^k, \lambda_{2t}^k, , .., \lambda_{pt}^k,)$, comme décrit par l'équation 4.4. Ainsi, $M_t^k = [e_{1t}^k, e_{2t}^k, .., e_{pt}^k]$ représente la matrice de vecteurs propres. Cette étape, coeur de notre approche de détection consiste à calculer une signature unique pour un ensemble de métrique mesurées, fondée sur les relations de variance-covariances entre ces métriques.

$$C_t^k = \frac{1}{v}(S_t^k)^T \times S_t^k \tag{4.3}$$

$$Ce_i = \lambda_i e_i \tag{4.4}$$

Par conséquent, la quatrième étape consiste en la comparaison de M_t^k avec la référence R telle que présentée précédemment. Pour ce faire, nous utilisons la notion de dissimilarité que l'on note A_t^k, comme décrit dans la section 4.3.1.2. Cette dernière est calculée via la norme de Frobenius $||M^{k_t} - R||_F$ dont l'équation 4.1 décrit le calcul.

Dans le chapitre 3, nous avons montré que les paramètres sont plus corrélés dans le cas des attaques franches que dans le cas des attaques à faible débit. Cela est dû à la difficulté de mesurer des activités faibles comme par exemple celle du CPU qui, dans le cas d'attaques à faible débit, passe par des phases de pauses, donnant des valeurs nulles et par conséquent plus de corrélation avec TX[52]. Ainsi et étant donnée que l'ACP est très sensible aux données aberrantes, la comparaison directe entre les espaces factoriels mesurés pour des attaques à faible débit et celui de référence engendrerait un taux très élevé de faux négatifs.

[52]dans le cas de l'attaque UDP Flood

Afin de remédier à ce problème, et sachant que les attaques durent un certain temps pour être efficaces, nous utilisons la moyenne mobile exponentielle pondérée *(Exponentially-Weighted Moving Average)* EWMA [136] comme un post-filtre. De cette façon, nous lions temporellement les données et limitons les changements brusques et par conséquent les détections de données aberrantes.

Comme décrit par l'équation 4.5, EWMA est une méthode statistique qui représente un filtre dans lequel une pondération exponentielle est utilisée pour estimer les valeurs futures d'une variable à partir d'un historique de données. En d'autres termes, c'est un procédé qui calcule la moyenne des données de façon à donner un poids à ces dernières. Ce poids diminue au fur et à mesure que les données sont plus éloignés dans le temps.

$$D_t^k = \alpha A_t^k + (1-\alpha) D_{t-1}^k \tag{4.5}$$

Le paramètre α représente la constante de lissage qui contrôle le degré de décroissance des poids applicables à chaque observation participant à la moyenne. Elle est comprise entre 0 et 1. Une petite valeur de α (proche de 0) donne plus de poids aux plus anciennes données. Par conséquent, les données sont plus liées et la détection pourrait être plus stable dans le temps. Cependant, plus petite est cette valeur, plus lente est la réactivité de l'algorithme, ce qui engendrera beaucoup d'erreur lors du changement du comportement des VMs attaquantes. A l'opposé, une grande valeur de α (proche de 1), attribut plus de poids aux données les plus récentes. Par conséquent, le système est plus réactif aux attaques. Cependant, cela ne résoudra pas complètement le problème des changements brusques et par conséquent des faux négatifs. Afin d'avoir un système assurant une détection stable tout en gardant sa réactivité, dans la suite de cette étude, nous choisirons une valeur moyenne $\alpha = 0.4$.

Finalement, nous fixons un seuil nommé H, afin décider si la valeur de dissimilarité D_t^k représente ou non une attaque DDoS. En d'autres termes, toute valeur de dissimilarité en dessous du seuil représentera une attaque. La règle de décision d_t^k est décrite par l'équation 4.6.

$$d_t^k = \begin{cases} True & si \quad D_t^k \leq H \\ False & si \quad D_t^k > H \end{cases} \tag{4.6}$$

L'algorithme 1 donné dans la section 4.4.4 synthétise le déroulement du processus de détection pour un *tenant* et pour un cycle de détection.

4.4 Evaluation de l'approche de détection

Afin de valider l'algorithme proposé, nous l'avons appliqué au jeu de données collectés lors de notre campagne de mesure présentée dans le chapitre 3. Nous avons alors mesuré sa performance, exprimée principalement par sa capacité à minimiser les fausses alarmes. Dans cette section, nous présentons les différents résultats obtenus dans ce cadre.

4.4.1 Cadre d'évaluation

Notre méthodologie d'évaluation de performance repose sur la simulation. Pour ce faire, nous avons implémenté l'algorithme de détection que nous proposons dans l'envi-

Débit d'attaque	#Tenants (attaquant incl.)	#VMs (attaquantes incl.)	# VMs attaquantes	Débit d'attaque agrégé
Expérimentation des attaques UDP Flood (Hybrid_V1.0)				
8 Mb/s	22	562	41	328 Mb/s
16 Mb/s	22	562	41	656 Mb/s
40 Mb/s	22	564	43	1,68 Gb/s
56 Mb/s	22	562	41	2,25 Gb/s
80 Mb/s	22	561	40	3,12 Gb/s
Expérimentation des attaques UDP Flood (Kaiten)				
8 Mb/s	22	564	43	344 Mb/s
16 Mb/s	22	564	43	688 Mb/s
40 Mb/s	22	564	43	1,68 Gb/s
56 Mb/s	22	564	43	2,35 Gb/s
80 Mb/s	22	564	43	3.36 Gb/s

TABLE 4.4 – Paramètres numériques des jeux de données utilisées dans les simulations

ronnement R[53]. Cet algorithme reçoit en entrée les fichiers journaux collectés lors de des campagnes de mesure décrites dans la section 3.2.3 du chapitre 3. Par conséquent, nous évaluons l'efficacité de l'algorithme dans un contexte réaliste où l'activité malveillante diluée dans une charge légitime conséquente. On note toutefois que pour des raisons de temps de simulation, nous utilisons uniquement quelques *tenant* choisis au hasard et non l'ensemble des données collectées.

Le tableau 4.4 décrit les jeux de données utilisés lors des simulations. Ces derniers concernent uniquement les expérimentations de l'attaques UDP Flood. En effet, les attaques TCP SYN Flood ont été réalisées sur une seule implémentation logicielle de *botcloud*. Ainsi, l'évaluation de l'approche en utilisant une référence issue d'une seule implémentation logicielle et sur les mêmes jeux de données à partir desquels elle a été conçue ne représente pas un cas d'usage qui permette une véritable évaluation. Au contraire, étant donné que la référence des attaques UDP Flood a été créée à partir de deux implémentations logicielles différentes, elle ne correspond à aucune de ces dernières spécifiquement. Par conséquent, l'évaluation de l'approche en utilisant cette référence ainsi que les jeux de données collectées lors des dix expérimentations de l'attaque UDP Flood représente un cas d'usage moins trivial.

Pour terminer, on note que lors de cette évaluation, nous faisons varier le seuil H de 1.40 jusqu'à 2.12 via des pas de 0.03. 1.40 représente la borne inférieure pour H. En d'autres termes cette valeur représente la valeur de dissimilarité la plus élevée entre la référence et les *botclouds* réalisant les attaques à différents débits. En ce qui concerne la limite supérieure, nous avons fait varier H jusqu'à atteindre un taux très élevé et invariant de faux positifs, ce qui correspond à 2.12.

[53] www.r-project.org

4.4.2 Métriques de performance

Nous avons calculé les matrices de confusion de toutes les simulations réalisées et correspondant au cas de UDP Flood. Une matrice de confusion contient des informations sur les classifications réelles et prédites effectuées par un algorithme de détection ou de classification : faux positifs (fp), faux négatifs (fn), vrais positifs (vp) et vrais négatifs (vn). A partir de cette dernière, nous avons calculé différents indicateurs statistiques tels que : les courbes *Receiver Operating Characteristic* (ROC), l'exactitude *(Accuracy*[54]*)* (*ACC*) et le Coefficient de Correlation de Matthews (*MCC*).

La courbe ROC représente une mesure de la performance d'un classificateur binaire. Graphiquement, cette dernière est représentée sous la forme d'une courbe qui donne le taux de vrais positifs en fonction du taux de faux positifs. Ainsi elle exprime et met au premier plan le coté positif de la détection comme décrit par la figure 4.6.

Area Under Curve (AUC) ou aire sous la courbe est la mesure de l'aire de la surface située sous le tracé d'une fonction mathématique dessinée dans un repère. Formellement, cette valeur correspond à l'intégrale de cette fonction. Dans le cas de la courbe ROC, l'AUC donne un indicateur numérique à cette courbe qui n'a qu'une interprétation qualitative. Une courbe ROC parfaite a une $AUC = 1$. Par conséquent, plus proche est la valeur de la AUC de 1 meilleure est la détection.

L'exactitude (*ACC*) d'une expérience est une mesure de la façon dont les résultats expérimentaux sont conformes à une valeur vraie. Ainsi, ça représente la proportion de résultats de type vrai (positif et négatif) parmi le nombre total d'observations, comme décrit par la figure 4.6. L'équation 4.7 décrit le son calcul.

$$ACC = \frac{VP + VN}{VP + FP + FN + VN} \tag{4.7}$$

Le *MCC* [137] est utilisé comme une mesure de la qualité globale d'une classification binaire. il représente un coefficient de corrélation entre les classifications binaires réelles et prédites ; il renvoie une valeur comprise entre -1 et +1. Un coefficient de 1 représente une détection parfaite, 0 une détection aléatoire et -1 indique un désaccord total entre la détection et la réalité. A la différence avec les courbes ROC, ou *ACC*, *MCC* donne un poids égal aux quatre mesures données par la matrice de confusion utilisées comme décrit par l'équation 4.8 et illustré par la figure 4.6.

$$MCC = \frac{VP * VN - FP * FN}{\sqrt{(VP + FP)(VP + FN)(VN + FP)(VN + FN)}} \tag{4.8}$$

Ainsi, grâce à ces trois indicateurs de performance, nous évaluerons (1) le taux de détection de l'occurrence d'une attaque ; (2) l'exactitude de ces détections et donc la capacité de générer des alertes de type vrai ; (3) la performance globale du détecteur en prenant en considération les quatre métriques étudiées.

4.4.3 Résultats de l'évaluation

Dans la suite de cette section nous présentons les résultats obtenus lors des évaluations. Pour chaque cas étudié, nous présenterons deux résultats : (1) sans le post filtre EWMA et

[54] On note que cet indicateur est différent de la précision statistique

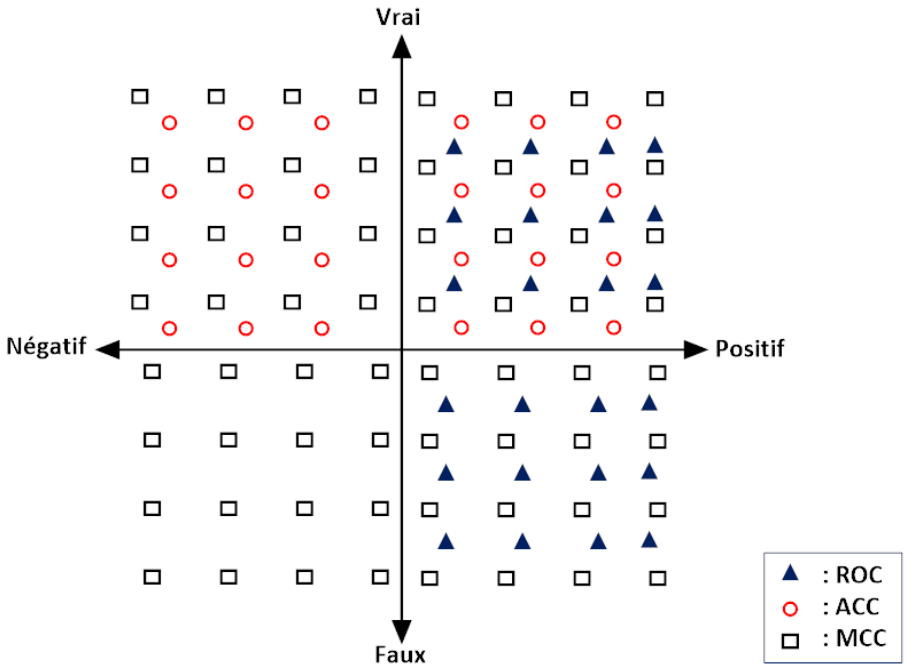


FIGURE 4.6 – Couverture des métriques d'évaluation

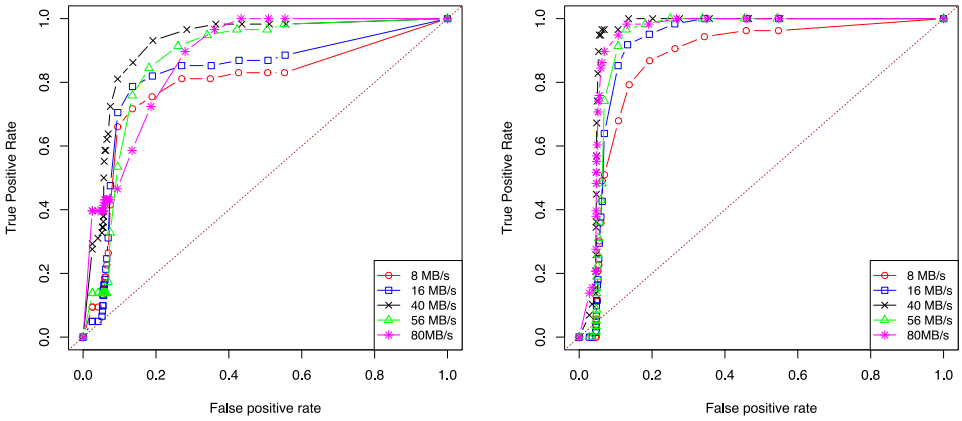
(2) avec le post filtre EWMA. La comparaison de ces derniers nous permettra de conclure quant à son efficacité.

4.4.3.1 Cas de Hybrid_V1.0

La figure 4.7.a représente les courbes ROC des expérimentations impliquant le *botcloud* Hybrid_v1.0 réalisant des attaques UDP Flood sans post filtre. Pour toutes les expérimentations, les courbes montrent une bonne qualité de détection de l'occurrence de l'attaque. Toutes les courbes sont largement au dessus de la ligne bissectrice. Nous remarquons aussi que plus le débit d'attaque est élevé, meilleure est sa détection, sauf pour le cas de 80 Mb/s où nous attribuons cette différence à l'état opérationnel de notre infrastructure de test au moment des mesures.

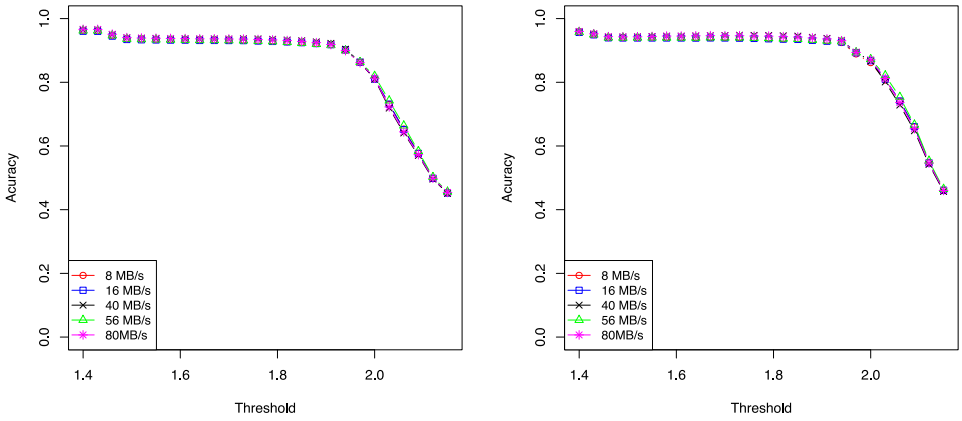
La figure 4.7.b représente les courbes ROC des expérimentations impliquant le *botcloud* Hybrid_v1.0 réalisant des attaques UDP Flood avec le post filtre. Pour toutes les expérimentations, les courbes montrent aussi une très bonne qualité de détection de l'occurrence de l'attaque, et largement meilleure que celle sans le post filtre, en atteste le tableau 4.5 qui décrit les AUC des courbes de l'ensemble des expérimentations et qui permet ainsi leur comparaison.

Les figures 4.8.a et 4.8.b représentent l'évolution de la valeur de ACC, des résultats produits par l'algorithme sans et avec l'application du post filtre EWMA, pour chaque valeur de H. Nous remarquons que quasiment toutes les valeurs de ACC sont au dessus de 90% pour un seuil $1.40 \leq H < 1.9$ et au dessus de 80% pour $1.9 \leq H \leq 2.0$. Au delà, cette valeur commence à baisser considérablement avec l'évolution du seuil. L'application



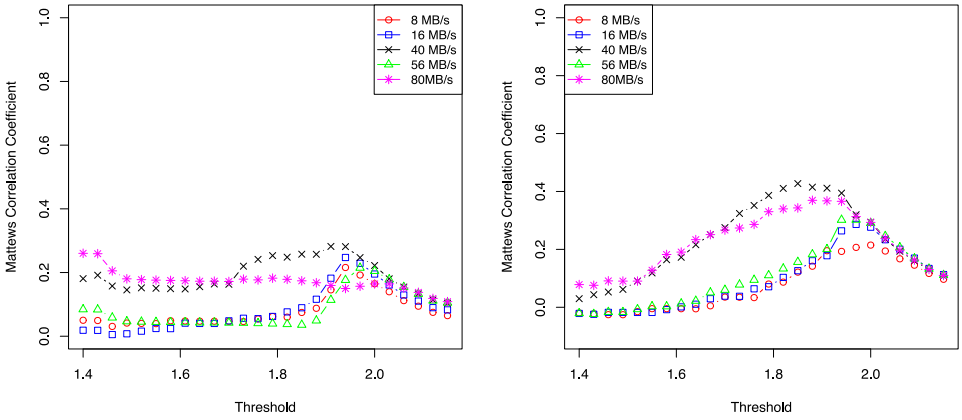
(a) (b)

FIGURE 4.7 – Courbes ROC (a) sans post filtre ; (b) avec post filtre



(a) (b)

FIGURE 4.8 – ACC (a) sans post filtre ; (b) avec post filtre



(a) (b)

FIGURE 4.9 – MCC (a) sans post filtre ; (b) avec post filtre

Débit d'attaque	8 Mb/s	16 Mb/s	40 Mb/s	56 Mb/s	80 Mb/s
AUC après EWMA	0.873	0.919	0.952	0.930	0.947
AUC sans EWMA	0.792	0.825	0.918	0.873	0.878
ACC après EWMA	0.861	0.892	0.941	0.928	0.940
ACC sans EWMA	0.862	0.901	0.922	0.863	0.966
MCC après EWMA	0.214	0.286	0.427	0.302	0.370
MCC sans EWMA	0.192	0.247	0.282	0.215	0. 260

TABLE 4.5 – Valeurs de AUC, MCC et ACC des expérimentation réalisées par Hybrid_V1.0

du post filtre améliore légèrement l'évolution de cette métrique.

La figure 4.9.a représente l'évolution de la valeur de MCC en fonction des valeurs de H, dans le cas de l'évaluation sans post filtre. Les meilleures valeurs obtenues pour le MCC sont 0.192, 0.247, 0.282, 0.215 et 0.260, respectivement pour les expérimentations de 8 Mb/s, 16 Mb/s, 40 Mb/s, 56 Mb/s et 80 Mb/s. Ces résultats reflètent une détection moyenne. En effet, en se reposant sur les deux dernières métriques, nous constatons que l'attaque est bien détectée, mais avec un surcoût de faux négatifs, démontré par les valeurs du MCC.

La figure 4.9.b représente l'évolution du MCC des mêmes expérimentations après l'application du post filtre dans le processus de détection. Les nouvelles meilleures valeurs obtenues pour MCC sont 0.214, 0.286, 0.427, 0.302 et 0.370 respectivement. Ces valeurs reflètent de meilleurs résultats de détection et témoignent de l'amélioration de la qualité par rapport à la version sans post filtre. La table 4.5 permet la comparaison de ces différents résultats.

L'étude de MCC permet également de comprendre l'impact du seuil de détection H. En effet, la valeur permettant la meilleure détection (valeur optimale) de H est celle où le MCC présente la plus grande valeur. Par conséquent, les valeurs des seuils optimales et qui correspondent au valeurs de MCC sont respectivement : 1.97, 1.94, 1.94, 1.97 et 1.40 pour les détections réalisées sans post filtre EWMA et 1.97, 1.97, 1.88, 1.94 et 1.88 pour celles réalisées avec le post filtre. Nous notons que la valeurs optimale de H se trouve dans un intervalle très étroit $[1.88, 1.97]$.

4.4.3.2 Cas de Kaiten

Comme pour le cas des expérimentations réalisées par Hybrid_V1.0, les figures 4.10.a et 4.10.b représentent les courbes ROC des expérimentations impliquant le *botcloud* Kaiten avec et sans l'application du post filtre EWMA dans le processus de détection. Pour toutes

les expérimentations, les courbes montrent une bonne qualité de détection de l'occurrence de l'attaque ainsi qu'une amélioration claire de ces dernières après l'application du post filtre. Le tableau 4.6 décrit les différentes AUC obtenues par ces courbes.

Les figures 4.11.a et 4.11.b décrivent l'évolution des valeurs de ACC des expérimentations décrite ci-dessus. Comme pour le cas de Hybrid_V1.0, toutes les valeurs de ACC sont au dessus de 90% pour un seuil $1.40 \leq H < 1.9$ et au dessus de 80% pour $1.9 \leq H \leq 2.0$. Au delà, cette valeur commence à baisser avec l'évolution du seuil. Egalement, l'application du post filtre améliore légèrement l'évolution de cette métrique. Ces résultats témoignent de la capacité du détecteur à détecter les alertes de type vrai.

Les figures 4.12.a et 4.12.b décrivent l'évolution du MCC selon les valeurs de H dans les cas sans et avec post filtre. Les meilleures valeurs obtenues pour le MCC sont respectivement pour chaque débit d'attaque 0.260, 0.253, 0.252, 0.235 et 0.285 pour les expérimentations sans post filtre et 0.416, 0.396, 0.381, 0.353 et 0.325 pour les expérimentations avec post filtre. Ces dernières reflètent encore un bon résultat de détection et témoignent de l'amélioration de sa qualité par rapport à la version sans le post filtre. Nous notons également que le détecteur réalise de meilleurs résultats pour la détection des attaques générées par Kaiten que celles générées par Hybrid_V1.0.

Le tableau 4.6 décrit ces différents résultats et permet leur comparaison. Les valeurs optimales de H et correspondant aux valeurs décrites de MCC sont respectivement : 1.91, 1.94, 1.88, 1.94 et 1.91 pour les expérimentation sans le post filtre et 1.85, 1.85, 1.85, 1.82 et 1.94 pour les expérimentations avec le post filtre. Dans ce cas encore, la valeur optimale de H est très peu variable, et se trouve dans un petit intervalle, sauf pour le cas de l'attaque à 80 Mb/s où $H = 1.94$.

Lors des expérimentations réalisées sur Kaiten, l'intervalle contenant les valeurs optimales de H est un peu éloigné de celui obtenu pour les expérimentations impliquant Hybrid_V1.0.

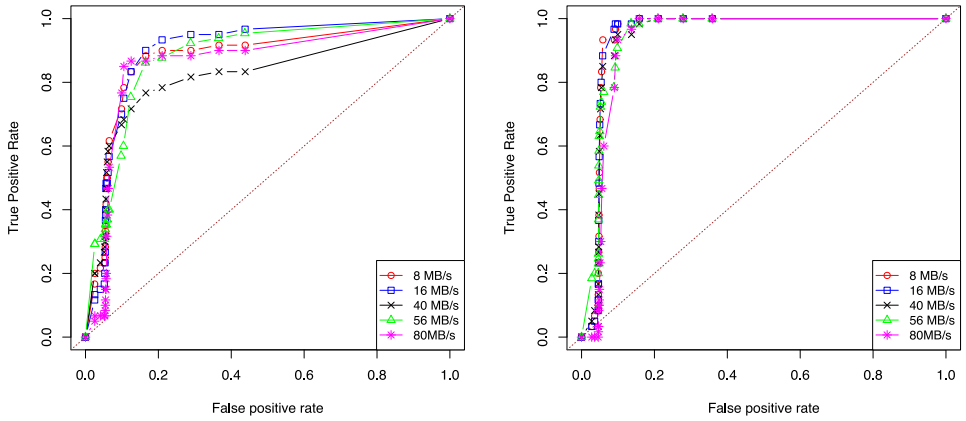
Pour résumer, l'approche proposée réalise de bon résultats pour la détection de l'occurrence de l'attaque et cela indépendamment de l'implémentation logicielle du *botcloud*avec une meilleure détection des attaques à hauts débits. Cependant, elle souffre d'un surcoût de faux négatifs.

4.4.4 Evaluation de la complexité de l'algorithme de détection

Dans cette section, nous présenterons une évaluation de la complexité temporelle de notre proposition de détecteur. Celle-ci repose sur l'algorithme 1 pour lequel, nous estimons la complexité de chaque étape. Finalement, nous déduisons le coût total de l'approche en termes de cycles de calcul, et cela pour un cycle de détection et un *tenant*.

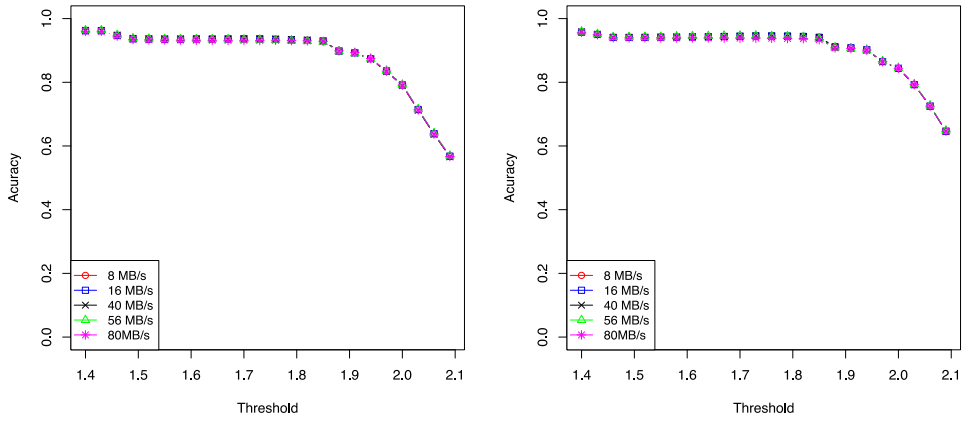
Avant tout, nous rappelons que l'algorithme de détection traite une matrice de données de taille $[v \times p]$, où v est le nombre de machines virtuelles en activité pour un *tenant* et p le nombre de paramètres collectés par le système de surveillance au niveau hyperviseur. Nous formulons aussi l'hypothèse que $p << v$. Sur cette base, la compléxité de chaque étape de l'algorithme est la suivante :

Centrage et réduction : Le centrage et réduction nécessite le calcul de la moyenne et de l'écart type sur chaque paramètre p_i



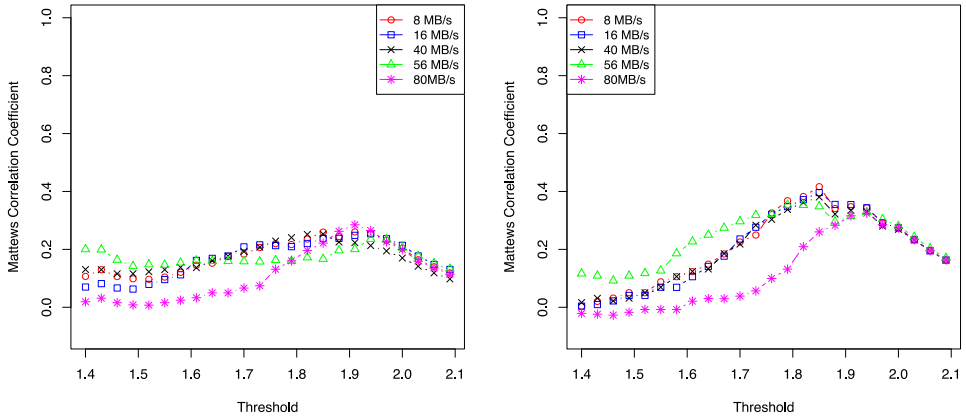
(a) (b)

FIGURE 4.10 – Courbes ROC (a) sans post filtre ; (b) avec post filtre



(a) (b)

FIGURE 4.11 – ACC (a) sans post filtre ; (b) avec post filtre



(a) (b)

FIGURE 4.12 – MCC (a) sans post filtre ; (b) avec post filtre

Débit d'attaque	8 Mb/s	16 Mb/s	40 Mb/s	56 Mb/s	80 Mb/s
AUC après EWMA	0.948	0.948	0.943	0.945	0.931
Anciennes AUC	0.877	0.897	0.820	0.881	0.861
MCC après EWMA	0.416	0.396	0.381	0.353	0.325
Anciens MCC	0.260	0.253	0.252	0.235	0.285
ACC après EWMA	0.941	0.941	0.940	0.942	0.901
Anciennes ACC	0.892	0.874	0.929	0.873	0.893

TABLE 4.6 – Valeurs de AUC, MCC et ACC des expérimentation réalisées par Kaiten

Algorithm 1: Detection algorithm

Declaration :
R : $[p][p]$ Double ▷ Attack reference eigenvector matrix
TRD : $[v][p]$ Double ▷ Matrix of all monitored data
S : $[v][p]$ Double ▷ standard score matrix
C : $[p][p]$ Double ▷ Covariance matrix
EvM : $[p][p]$ Double ▷ Eigenvector matrix
A : Double ▷ Distance of M from R
d : Boolean ▷ Attack decision
H : Integer ▷ Detection threshold

1: **function** DETECTION
2: $S \leftarrow$ ScaleMatrix(TRD)
3: $C \leftarrow$ CovarianceMatrix(S)
4: $EvM \leftarrow$ EigenvectorMatrix(C)
5: $A \leftarrow$ FrobeniusDistance(EvM, R)
if $A < H$ **then**
6: $d \leftarrow True$
else
7: $d \leftarrow False$
8: **return** d
9: **end function**

— **Moyenne :** Pour un paramètre p_i la complexité du calcul de la moyenne est de $O(v)$. Considérant les p paramètres, cette complexité devient $O(p \times v)$.
— **Ecart type :** la complexité du calcul de l'écart type est de $O(p \times v)$.

Etant donné que les étapes du calcul du centrage et de la réduction se font séquentiellement, la complexité globale de la fonction $ScaleMatrix()$ est l'étape ayant la plus grande complexité donc $O(p \times v)$.

Matrice de covariance : La fonction $CovarianceMatrix()$ utilise l'équation 4.3 où S représente la matrice centrée réduite, par conséquent la complexité de cette dernière n'est pas considérée. Nous rappelons que S est de taille $[v \times p]$. Etant donné que l'équation 4.3 représente une multiplication matricielle, la multiplication entre S^T et S nécessite la multiplication de chaque ligne de S^T (p lignes) par les colonnes de S (p colonnes), s'ajoute à cela la somme des multiplications terme à terme des éléments des lignes et colonnes en question. Ainsi, la complexité de cette étape est de l'ordre de $O(p^2v)$.

Matrice des vecteurs propres : La fonction $EigenVectorMatrix()$ qui a pour rôle le calcul de la matrice des vecteurs propres se décompose en deux étapes :

1. La résolution de l'équation 4.9 caractéristique :

$$det(C - \lambda I) = 0 \tag{4.9}$$

   permet de déterminer les valeurs propres λ_i auxquelles sont associés les vecteurs propres e_i. Nous rappelons que les matrices C et I sont toutes les deux de taille $[p \times p]$. Ainsi cette étape nécessite la résolution d'un système d'équation en $O(p^3)$.

2. Une fois les λ_i déterminés, nous pouvons déduire les vecteurs propres en utilisant l'équation 4.4. Ceci conduit encore à un système d'équations en $O(p^3)$ qui doit être résolu pour chaque valeur de i, donc p fois. Par conséquent la complexité de cette phase est de l'ordre de $O(p^4)$.

Ainsi, la complexité globale de l'étape du calcul de la matrice des vecteurs propres est de l'ordre de $O(p^4)$.

Norme de Frobenius : Cette étape nécessite le parcours de deux matrice de taille $[p \times p]$. Ainsi, elle a une complexité de l'ordre de $O(p^2)$.

Complexité globale de l'algorithme : Etant donné que les étapes décrites sont séquentielles, la complexité de l'algorithme correspond à la plus grande complexité des étapes qu'il contient. Sachant que nous formons l'hypothèse que $p << v$, il apparait que la complexité algorithmique dépend majoritairement de v et est ainsi de l'ordre $O(p^2v)$.

Cette complexité peux facilement être gérée, compte tenu de la performance des systèmes actuels [138], mais uniquement pour un nombre réduit de VM surveillées. Cependant, considérant le volume et le nombre de VM qu'une infrastructure *cloud* peut contenir, la considération d'une approche centralisée ayant une telle complexité représente un véritable goulot d'étranglement.

4.5 Conclusion

Dans ce chapitre nous avons présenté l'approche de détection à la source que nous proposons contre la mise en oeuvre d'attaques DDoS dans un *cloud*. Nous avons tout

d'abord présenté la méthodologie et les étapes de construction des références et montré que cette référence discrimine clairement l'activité malicieuse de l'activité légitime. Nous avons ensuite proposé un algorithme de détection basé sur l'ACP qui propose, sur la base de la signature définie par l'empreinte système des VMs impliquées dans ce type d'attaque, d'évaluer la distance d'une fenêtre d'activité d'un *tenant* face à cette signature. Par la suite nous avons procédé à l'évaluation de cet algorithme et nous avons démontré que ce dernier réalise une détection performante et cela indépendamment de l'implémentation logicielle du *botcloud*avec une meilleure détection des attaques à hauts débits, mais avec un surcoût de faux négatifs. Finalement nous avons produit une évaluation de la complexité temporelle de notre algorithme et montrer que sa complexité linéaire en font une solution déployable dans un contexte réel.

Cependant, un *cloud* public représente une infrastructure largement distribuée et hautement évolutive qui héberge de nombreuses activités. Par conséquent, implémenter une approche centralisée telle que celle proposée dans ce chapitre pourrait présenter des limites en termes de support du facteur d'échelle pour une détection en mode flux, et au delà, agir comme un point de défaillance unique.

Afin de permettre à notre solution de réaliser une détection performante et en temps voulu, il est nécessaire que cette dernière soit distribuée. Dans le chapitre suivant, nous proposons une évolution de notre solution, qui repose sur l'approche décrite dans ce chapitre, tout en distribuant ses opérations.

5

Une solution de distribution pour l'approche proposée

Sommaire

5.1 Introduction

Concevoir un détecteur dont la performance, exprimée par sa capacité de détection des attaques tout en minimisant les fausses alarmes, est haute, est une première étape vers la mise en oeuvre d'une solution de sécurité. Toutefois, ce détecteur doit aussi être en mesure de fournir son résultat sous un temps qui soit le plus court possible, tout en nécessitant le moins de ressources de calcul et communication possibles. La solution présentée dans le chapitre précédent satisfait à ce premier critère mais sa complexité fait qu'elle ne peut supporter la prise en charge d'un grand nombre de machines virtuelles tout en fournissant une forte réactivité.

Dans ce chapitre, nous présenterons ainsi une solution distribuée qui repose sur l'approche centralisée décrite dans le chapitre 4. Mais contrairement à cette dernière, au lieu d'effectuer un traitement centralisé pour tous les *tenant* surveillés, ce traitement est distribué sur plusieurs noeuds d'une architecture constituée de multiples réseaux *overlay* pair-à-pair organisés sous forme hiérarchique. Nous présentons ici le principe de cette ap-

proche ainsi qu'une évaluation de ses performances, fondée sur les mêmes indicateurs que ceux utilisés précédemment.

5.2 Etat de l'art sur les IDS pair-à-pair

Les réseaux et protocoles pair-à-pair sont couramment utilisés dans les IDS afin d'assurer la collaboration entre les différents noeuds composant ces dits systèmes. Dans cette section nous en présentons quelques uns et en particulier ceux qui sont fondés sur les tables de Hachage Distribuée (DHT).

Indra [139] fut l'une des premières propositions d'IDS collaboratif. Indra représente une solution complètement distribuée qui vise la protection des réseaux LAN. Les noeuds (IDS) participants sont implémentés au niveau des machines du réseau LAN protégé et sont interconnectés via un réseau pair-à pair. Chaque IDS surveille ses voisins et s'il détecte une intrusion ou attaque il diffuse une alerte.

Dans [140], Kholidy *et al.* proposent un IDS développé pour les environnements de *cloud computing*. Son infrastructure est distribuée et repose sur un réseau pair-à-pair. Afin de couvrir un large éventail d'attaques, le système intègre des techniques de détection comportementales et à base de signatures. Chaque noeud comprend deux composants appelés CIDS pour la corrélation des alertes et HIDS pour l'audit réalisé au niveau de l'hyperviseur.

NetShield [141] est un IDS distribué qui repose sur la DHT Chord [63]. Dans ce système, les IDS participant contribuent et récupèrent les données via l'*overlay* pair-à-pair. Chaque IDS maintient une table de prévalence locale pour enregistrer le nombre d'occurrences de chaque signature de bloc de contenu ainsi que son adresse source et adresse de destination. Une mise à jour est déclenchée si la prévalence locale du bloc de contenu dépasse un seuil local. Si la prévalence globale est supérieure à un seuil donné et si la dispersion d'adresse dépasse un certain seuil également, une alarme est soulevée concernant le bloc de contenu correspondant. Netshield cible les attaques de type *worm outbreaks* et DoS. Cependant, l'utilisation de blocs de contenu pour l'identification d'attaques n'est pas efficace contre les vers polymorphes *(polymorphic worms)* [142]. En outre, NetShield suppose que tous les participants sont honnêtes, ce qui le rend vulnérable aux attaques par collusion *(collusion attacks)* [143] et aux nœuds malveillants.

CIDS [144] représente un autre IDS collaboratif complètement distribué qui utilise la DHT Chord pour organiser les IDS participant sous forme d'un réseau pair-à-pair. Chaque IDS partage sa liste noire *blacklist* d'adresses IP avec les autres via le réseau *overlay*. Si une adresse IP suspecte est rapportée plus d'un seuil N, alors une alerte concernant cette adresse est levée. CIDS est considéré comme évolutif et robuste car il est construit sur une architecture totalement décentralisée. Cependant, la limitation de ce système réside dans l'identification des intrus potentiels par leur adresse IP. Ainsi, il n'est pas efficace contre les attaque qui ont un degré de distribution inférieur à N.

Cyber Disease Distributed Hash Table (CDDHT) [145] est un IDS collaboratif distribué qui repose sur une infrastructure pair-à-pair basée sur une DHT. Dans son architecture, chaque nœud est un IDS local qui tente de détecter localement les attaques et générer des alertes correspondantes. A chaque alerte est attribuée une clé *(disease key)* ba-

sée sur l'intrusion correspondante. L'alerte est envoyée à un noeud ayant un rôle de centre de fusion SFC *(Sensor Fusion Center)*. Les SFC représentent des noeuds sélectionnés en fonction de leur capacité et leurs ressources. Le but de ce système est d'éviter le problème de goulots d'étranglement inhérents aux IDS centralisés et d'utiliser des techniques de catégorisation d'alertes pour équilibrer la charge entre les SFC.

5.3 Architecture fonctionnelle du système

Les travaux présentés dans la section précédente reflètent la grande capacité des réseaux pair-à-pair à répondre aux besoins des IDS et leur distribution et cela pour les différentes méthodes de détection sous jacentes. Dans la majorité des cas, ces réseaux assurent l'acheminement des données vers les unités de corrélation. Cependant, souvent, ils n'assurent pas la fonction d'agrégation.

Dans cette section nous présenterons l'architecture distribuée que nous proposons ainsi que les différentes fonctions de ses composants. Cette dernière repose sur un réseau pair-à-pair qui assure également l'agrégation des données, contrairement aux approches présentées précédemment.

5.3.1 Architecture globale

Afin de concevoir un système de détection adapté à une infrastructure telle que celle du *cloud computing*, ce dernier doit répondre à certaines caractéristiques qui permettent de garantir son efficacité de traitement et son bon fonctionnement.

Premièrement, le système proposé doit comprendre une infrastructure distribuée et extensible afin d'être capable de supporter la haute distribution et l'évolutivité du *cloud*, ainsi que les potentielles centaines de milliers de VMs qu'un fournisseur de services peut héberger. Deuxièmement, le système de détection doit gérer le taux très élevé de *churn* des VMs qui peuvent être créées, détruites et déplacées d'un hyperviseur à un autre de manière très dynamique [146]. Troisièmement, l'absence de tout point central de traitement qui pourrait représenter un goulot d'étranglement ou point de défaillance unique, est un point important pour assurer la résilience du système de détection. Quatrièmement, étant donné que le *cloud* héberge une charge volumineuse, l'architecture de l'IDS doit avoir une structure intelligente afin d'éviter une surcharge et un surcoût en terme de communications réseau. Enfin, l'idée sur laquelle doit reposer l'approche distribuée réside dans la simplification des traitements. En d'autres termes, au lieu d'effecter un traitement centralisé pour tous les *tenants* surveillés, ce traitement doit être distribué sur plusieurs noeuds de l'IDS puis agrégé pour fournir un résultat global.

Afin de satisfaire à toutes ces caractéristiques, nous proposons une solution fondée sur un réseau pair-à-pair qui repose sur une architecture composée de multiples structures hiérarchiques. Ces dernières sont maintenues par des tables de hachage distribuées (DHT) telles que celles fondées sur l'algorithme de Plaxton [147]. Au delà, les DHT sont bien adaptées au cadre applicatif du *cloud* et sont déployées dans ce cadre pour différentes applications telles que l'allocation de ressources et la répartition de charge [148], la fédération d'infrastructures pour la fourniture de services applicatifs à grande échelle

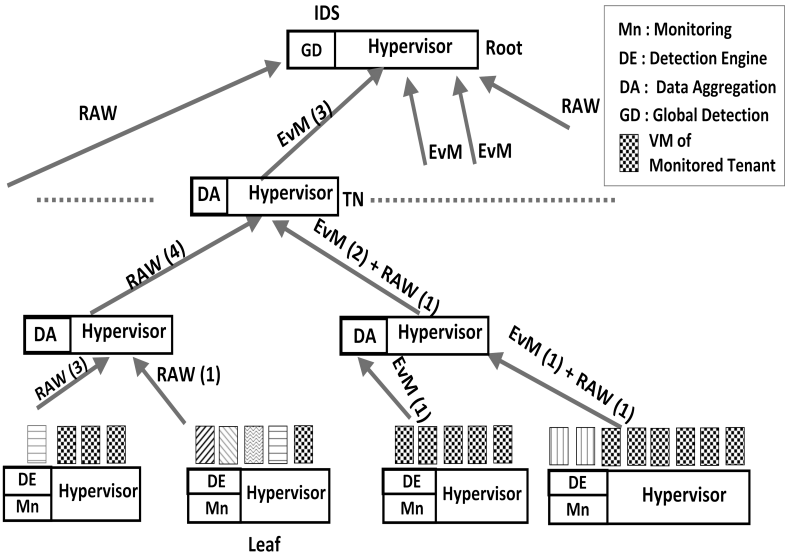


FIGURE 5.1 – Architecture du système et scenario de détection pour $n = 5$

[149] ou la mise en oeuvre de services multimédia [150].

Ainsi, l'approche de détection hiérarchique proposée repose sur une architecture ayant une topologie maillée, résultat du chevauchement de multiples réseaux *overlay* organisés sous forme d'arbres. En son sein, chaque *tenant* est considéré séparément et un arbre est construit pour chacun. La figure 5.1 décrit l'architecture du système et le scénario de détection pour un *tenant*. Le processus de détection appliqué est celui présenté dans l'approche centralisée présentée dans la section 4.3.2.1 et repose sur le calcul de la dissimilarité entre l'espace factoriel décrivant l'activité d'un *tenant* et R. Etant donnée les limites de l'approche centralisée, l'approche distribuée repose sur l'idée de (1) appliquer le calcul de l'ACP sur de petits ensembles de VMs ; un ensemble ayant le nombre requis de VMs est nommé *Clique* (Q) et a une taille n qui est fixée en tant que paramètre connu *a priori* (la figure 5.1 considère par exemple un cas où $n = 5$) ; (2) l'agrégation de l'ensemble des résultats afin de prendre une décision sur l'état du *tenant* (attaquant ou non). Une telle approche réalise des réductions considérables des temps d'exécution, des cycles CPU et des consommations mémoire.

Pour atteindre un tel objectif, chaque nœud de l'architecture représente une instance d'IDS local, implémenté au niveau de l'hyperviseur et a une tâche spécifique dont la description est donnée ci-dessous. Selon sa position dans l'arbre, un nœud peut être de l'un des trois types suivants :

— *Root* : représente le nœud ayant le plus haut niveau d'un arbre donné ; il est le responsable des décisions de détection.
— *Transition Nodes (TN)* : représentent les nœuds assurant la transition et l'agrégation de données entre les *Leaf Nodes* et *Root*.
— *Leaf Nodes (LN)* : représentent les nœuds ayant le niveau le plus bas et qui interagissent directement avec les VMs via la fonction de surveillance. Ils assurent également la transmission des données récupérées.

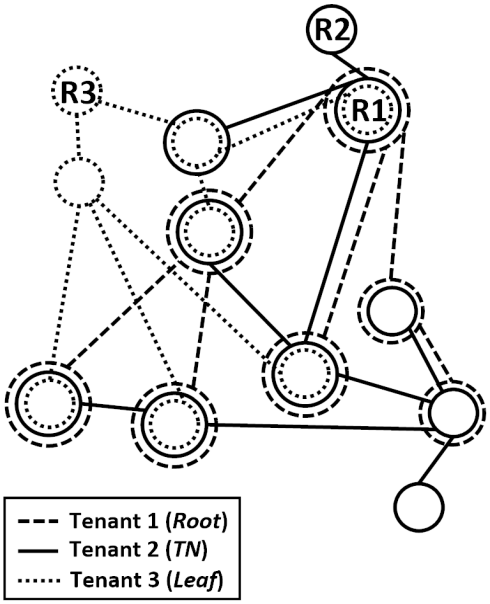


FIGURE 5.2 – La nature multi-fonctions des nœuds

De par la nature pair-à-pair de notre approche, le type d'un nœud donné change d'un *tenant* à l'autre. En d'autres termes, un nœud peut être *Root* pour un *tenant* donné, un *TN* pour un autre *tenant* et un *LN* pour un troisième *tenant*. La figure 5.2 illustre cet aspect. En son sein, le nœud $R1$ représente le *Root* pour le $Tenant1$, un *TN* pour le $Tenant2$ et un *LN* pour le $Tenant3$.

Chaque nœud, à l'exception de la racine, communique uniquement avec son père. En outre, nous supposons que chaque noeud fournit des fonctions primitives d'un protocole de communication, afin d'envoyer et de recevoir des messages de détection. Cette API est décrite dans le pseudo-code 2. Elle comprend au total cinq fonctions : (1) la fonction GETSYSTEMMETRICS() assure la collecte des données système surveillées par un processus d'instrumentation ; (2) la fonction RECEIVERAWDATA(), permet la réception de données brutes non traitées ; (3) la fonction RECEIVEPCA() permet la réception des matrices de vecteurs propres calculées par les fils ; (4) la fonction SENDRAWDATA() assure l'envoi des données brutes au père ; (5) la fonction SENDPCA() assure l'envoi des matrices de vecteurs propres, reçues ou calculées, au père.

Algorithm 2: Opérations basiques des noeuds

```
void : SendPCA(List{[p][p] Real})
void : SendRawData([x][p] Real)
List{[p][p] Real} : ReceivePCA()
[x][p] Real : ReceiveRawData()
[x][p] Real : GetSystemMetrics()
```

5.3.2 Algorithmes des noeuds

Nous rappelons que le coeur de l'approche distribuée que nous proposons repose sur le calcul de la dissimilarité entre l'espace factoriel décrivant l'activité d'un *tenant* et R. Mais à la différence de l'approche centralisée, le traitement se fait sur des *Cliques* de taille n. L'algorithme 3 représente les fonctions qui permettent l'application de l'ACP et le calcul de cet espace factoriel. Ces dernières sont décrites en détail dans la section 3.2.4.2.

Algorithm 3: Détection au niveau local

```
   Declaration :
   S : [n][p] Real                          ▷ Matrice centrée réduite
   C : [p][p] Real                          ▷ Matrice de variance-covariance
   EvM : [p][p] Real                        ▷ Matrice de vecteurs propres
1: function ApplyPCA(X : [n][p] Real)       ▷ X : Données brutes
2:     S ← ScalingMatrix(X)                 ▷ Centrer et réduire la matrice
3:     C ← CovarianceMatrix(S)
4:     EvM ← EigenvectorMatrix(C)
5:     return EvM
6: end function
```

5.3.2.1 Leaf Nodes

Les LN représentent ceux situés au plus bas niveau de l'arbre et représentent les seuls nœuds a avoir une interaction directe avec les VMs via la fonction de collecte. Les données récupérées sont stockées dans une matrice appelée *Table of Raw Data* (TRD). Selon la taille de TRD, qui représente le nombre de VMs que l'hyperviseur héberge pour un *tenant* donné, l'IDS exécute l'une des trois actions suivantes :

- Si $Taille(TRD) < n$: le LN envoie à son père un message contenant un tableau qui comporte les données brutes récupérées auprès de toutes les VMs.
- Si $Taille(TRD) = n$: le LN applique l'algorithme 3 sur les données récupérées. Ensuite, il envoie à son père un message contenant la matrice de vecteurs propres, résultat du dernier algorithme sur la *Clique* surveillée.
- Si $Taille(TRD) > n$: le LN applique l'algorithme 3 sur chacune des *(nombre des VMs / n) Cliques*. Ensuite, il envoie deux messages à son père ; le premier contient une liste de toutes les matrices de vecteurs propres, appelée *List of Eigenvector Matrices (LEM)*, résultat des différentes applications de l'algorithme 3. Le second message contient une table qui comprend les *(nombre des VMs Modulo n)* données brutes restantes.

5.3.2.2 Transition Nodes

Les TN représentent les nœuds intermédiaires entre les LN et *Root*. Ils sont appelées *Transition Nodes* car ils assurent la transition des données ainsi que les opérations d'agrégation. De façon générale, les TN agissent comme les LN. La seule différence avec

les LN réside dans l'acquisition des données. En effet, un LN reçoit uniquement les données brutes par le processus de collecte, alors qu'un TN reçoit des tableaux de données brutes TRD ainsi que les listes LEM, de la part de ses enfants. Ainsi, un TN stocke les différents messages qu'il reçoit de tous ses enfants, LN ou autres TN. Par conséquent, il possède un tableau qui regroupe les données brutes, appelée également TRD et une liste qui regroupe les LEM, appelée aussi LEM. Selon la taille de TRD, trois scénarios sont considérés, comme pour le cas des LN. L'algorithme 4 décrit le fonctionnement des nœuds LN et TN.

Algorithm 4: Agrégation de données par LN et TN

Declaration :
$TYPE$: ▷ Type du noeud
$Tenant$: ID ▷ ID du *tenant*
Q : [n][p] Real ▷ Données d'une clique de VMs
EvM : [p][p] Real ▷ Matrice de vecteurs propres
TRD : [x][p] Real ▷ Table de données brutes
LEM : List{[p][p] Real} ▷ Liste des matrices de vecteurs propres

1: **procedure** DataAggregation
  if $TYPE = Leaf$ **then**
2: $TRD \leftarrow$ GetSystemMetrics($Tenant$) ▷ Surveillance des VMs
  else
3: TRD.concat(ReceiveRawData())
4: $LEM.add$(ReceivePCA())
  switch TRD.size() **do**
  case TRD.size() $< n$
5: SendRawData(TRD)
  case TRD.size() $= n$
6: $EvM \leftarrow$ ApplyPCA(TRD)
7: $LEM.add$(EvM)
8: SendPCA(LEM)
  case TRD.size() $> n$
9: $j \leftarrow 1$
10: **for** $i \leftarrow 1, (\text{TRD.size}()/n)$ **do**
11: $Q \leftarrow TRD[j : j + n][p]$
12: $EvM \leftarrow$ ApplyPCA(Q)
13: $LEM.add(EvM)$
14: $j \leftarrow j + n$
15: $TRD \leftarrow TRD[j : \text{TRD.size}()][p]$
16: SendPCA(LEM)
17: SendRawData(TRD)
18: **end procedure**

5.3.2.3 Root

Le nœud *Root* est le responsable des décisions de détection comme décrit dans l'algorithme 5. Comme les TN, il reçoit différents messages de la part de tout ses enfants. Ainsi, il construit également un TRD et une LEM. Ensuite, il applique l'algorithme 3 sur chacune des *(nombre des VMs / n) Cliques* ainsi que pour les *(nombre des VMs Modulo n)* données brutes restantes. Chaque résultat est ajouté à la LEM. Sa prochaine étape consiste à calculer l'espace factoriel qui décrit l'activité globale du *tenant*. Chaque élément de LEM représente un espace factoriel qui représente l'activité d'une *Clique*. Le centre de gravité de tout ces espaces factoriels représente un espace factoriel médian M_t^k qui décrit l'activité globale du *tenant*. Le calcul de ce dernier se fait via le calcul de la moyenne arithmétique de toutes les matrices de vecteurs propres M_t^{qk} éléments de LEM.

Une fois M_t^k calculé, la prochaine étape consiste en le calcul de sa dissimilarité (A_t^k) avec R, décrit par l'équation 4.1, comme pour le cas de l'approche centralisée détaillée dans la section 4.3.2.1. Ensuite, cette dissimilarité est traitée par le post-filtre EWMA comme décrit par l'équation 4.5. La dernière étape consiste en la comparaison de cette nouvelle dissimilarité appelée D_t^k avec le seuil H comme décrit par l'équation 4.6.

5.4 Evaluation de l'approche

Dans cette section nous présenterons les résultats que nous avons obtenus dans le cadre de l'évaluation de l'approche que proposons. Plus spécifiquement, cette évaluation porte sur une évaluation de l'efficacité du détecteur et la qualité de sa détection. Afin de permettre une comparaison directe des résultats obtenus dans le cas d'une mise en oeuvre centralisée, telle que présentée dans la section 4.4 du chapitre précédent, et l'approche distribuée que proposons maintenant, nous considérons ici les trois mêmes indicateurs qui sont (1) les courbes ROC et les AUC pour évaluer l'efficacité de la détection de l'occurrence de l'attaque en terme de vrais positifs versus faux positifs ; (2) l'exactitude ACC pour évaluer le taux des détections de type vrai ; (3) le coefficient de corrélation de Matthews qui décrit la qualité de la détection en se reposant équitablement sur les quatre métriques surveillées.

5.4.1 Scénarios d'évaluation

Le cadre d'évaluation de l'approche distribuée que nous proposons est similaire à celui décrit dans la section 4.4.1 : nous utilisons les jeux de données réels issus des campagnes de collecte présentés dans le chapitre 3 et nous utilisons l'outil R, comme environnement de simulation ; ce dernier implémente alors une version distribuée de l'algorithme de détection. Dans le but de comparer les performances de l'approche distribuée avec ceux de l'approche centralisée, nous utilisons les mêmes jeux de données que nous avons utilisé lors de l'évaluation de cette dernière et qui sont décrit en détail dans le tableau 4.4 du chapitre 4.

Etant donné que l'approche proposée repose sur les *Cliques*, notre évaluation se divise en deux parties : (1) l'évaluation de l'impact de la taille des *Cliques* sur la qualité de la détection et (2), pour une taille de *Clique* donnée, l'évaluation de la qualité de détection.

Algorithm 5: Détection au niveau global

Declaration :

Q : [n][p] Real ▷ Données d'une clique de VMs
EvM : [p][p] Real ▷ Matrice de vecteurs propres
M : [p][p] Real ▷ Centre de gravité des matrices de vecteurs propres
R : [p][p] Real ▷ Référence d'attaque
TRD : [x][p] Real ▷ Table de données brutes
LEM : List{[p][p] Real} ▷ Liste des matrices de vecteurs propres
A : Integer ▷ Valeur de dissimilarité brute
D : Integer ▷ Valeur de dissimilarité filtrée
H : Integer ▷ Seuil de détection

1: **function** RootFunction
2: TRD.concat(ReceiveRawData())
3: LEM.add(ReceivePCA())
4: $j \leftarrow 1$
5: **for** $i \leftarrow 1, (\text{TRD.Size}()/n)$ **do**
6: $Q \leftarrow TRD[j : j + n][p]$
7: $EvM \leftarrow \text{ApplyPCA}(Q)$
8: LEM.add(EvM)
9: $j \leftarrow j + n$
10: $TRD \leftarrow TRD[j : \text{TRD.size}()][p]$
11: $EvM \leftarrow \text{ApplyPCA}(TRD)$
12: LEM.add(EvM)
13: $M \leftarrow \text{LEM.mean}()$
14: $A \leftarrow \text{FrobeniusNorm}(M - R)$
15: $D \leftarrow \text{EWMA}(A)$
if $D < H$ **then**
16: $d \leftarrow True$
else
17: $d \leftarrow False$
18: **return** d
19: **end function**

Dans ce cadre, nous avons réalisé plusieurs simulations, en faisant varier n de 5 à 10, en considérant tous les débits d'attaque décrits dans le tableau 4.4 et en considérant 8 hyperviseurs pour chaque simulation. On note que les valeurs de n ont été choisies dans l'intervalle [5, 10] afin de toujours compter sur de petites *Cliques* pour la détection, tout en couvrant une plage de complexité de l'ordre d'une décade pour l'algorithme 3.

En l'absence de politique d'allocation des VMs sur les hyperviseurs des serveurs physiques, nous considérons une stratégie d'allocation aléatoire. Ainsi, pour chaque valeur de n, nous avons réalisé 50 simulations qui utilisent une réalisation d'allocation différente. Chaque résultat présenté représente ainsi la moyenne des 50 itérations réalisées. Les intervalles de confiance obtenus étant alors très étroits, pour des raisons de visibilité, nous les omettons dans les résultats présentés par la suite.

5.4.2 Evaluation de l'impact de la taille des *Cliques*

La figure 5.3.a représente l'évolution des AUC des courbes ROC, obtenues via les expérimentations impliquant Hybrid_V1.0, sur l'ensemble des valeurs de n. Nous pouvons noter qu'une augmentation de la valeur de n n'améliore que très légèrement la qualité de la détection de l'occurrence d'attaques, sauf pour le cas d'une très faible attaque comme celle à 8Mb/s où la valeur de n n'a pas d'influence sur la détection.

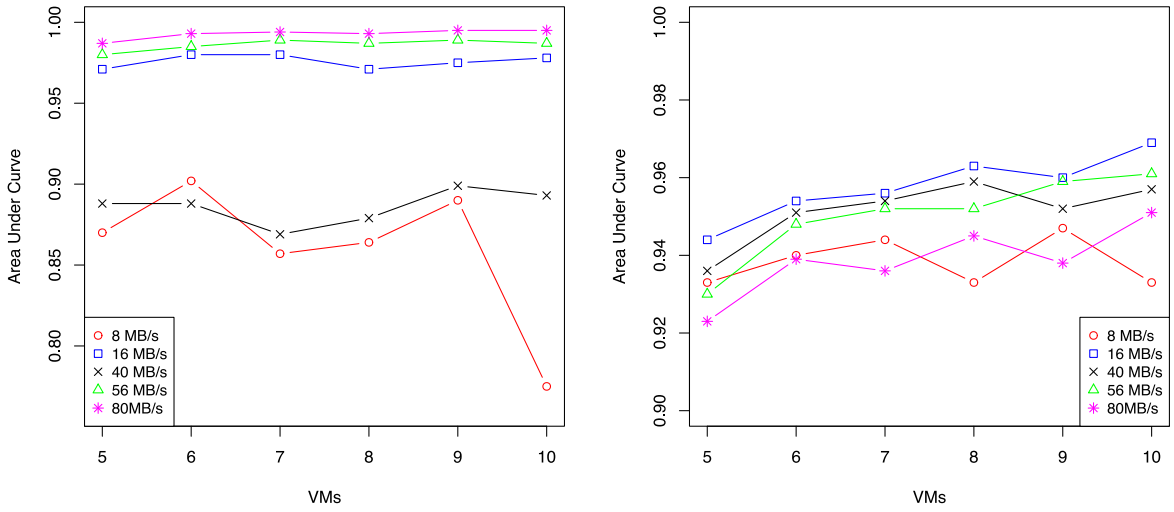
La figure 5.3.b représente l'évolution des AUC des courbes ROC, obtenues via les expérimentations impliquant Kaiten, sur l'ensemble des valeurs de n. Pour ce cas également, nous pouvons constater qu'une augmentation de la valeur de n améliore seulement légèrement la qualité de la détection de l'occurrence d'attaques. Nous constatons également le même scénario pour les attaques à faibles débits comme celle à 8Mb/s où la valeur de n n'a pas d'influence sur la détection.

La figure 5.4 représente l'évolution du MCC des expérimentations impliquant Hybrid_V1.0 er sur l'ensemble des valeurs de n pour trois expérimentations : l'attaque à 8 Mb/s, à 56 Mb/s et à 80 Mb/s. Ces résultats confirment le fait que l'augmentation de la valeur de n n'améliore que légèrement la qualité de la détection pour celles à haut débit. Nous constatons également que pour le cas de l'attaque à 8 Mb/s la taille de la *Clique* n'a pas d'influence considérable sur la détection. En outre, cette taille n'a également pas d'influence sur le seuil H optimal, sauf pour le cas de l'attaque a 8 Mb/s.

La figure 5.5 représente l'évolution du MCC des expérimentations impliquant Kaiten sur l'ensemble des valeurs de n pour les trois expérimentations ayant les mêmes débits d'attaque. Les résultats obtenus sont similaires à ceux décrits précédemment et nous permettent de conclure quant à l'indépendance relative de la taille des *Cliques* sur la performance de la détection.

5.4.3 Evaluation de la performance de détection

Dans cette section nous présentons l'évaluation de notre système de détection distribué pour une valeur du paramètre n fixée à 5. La valeur de n a été choisie en fonction des résultats exposés précédemment, car c'est celle qui permet d'obtenir des performances de détection acceptables tout en offrant le coût de calcul le plus faible. On note en outre, que c'est cette valeur qui place notre approche dans les conditions les plus défavorables étant



(a) (b)

FIGURE 5.3 – AUC des expérimentations de l'attaque UDP Flood ; (a) Hybrid_V1.0 ; (b) Kaiten

donné qu'elle conduit au plus grand nombre de *Cliques*, maximisant ainsi son aspect distribué, tout en considérant aussi le moins d'échantillons possibles pour une *Clique*, induisant ainsi les plus grandes erreurs de détection possibles.

De façon exhaustive et méthodique, on relève les résultats suivants pour chacun des indicateurs et scénario de détection :

— La figure 5.6.a représente les courbes ROC des expérimentations impliquant Hybrid_v1.0 réalisant des attaques UDP Flood. Pour toutes les expérimentations, les courbes montrent une très bonne qualité de détection de l'occurrence d'attaques. En effet, les AUC respectives aux expérimentations décrites sont : 0,870, 0.971, 0.888, 0.980 et 0.987. Nous constatons également que la détection de l'occurrence des attaques à faible débit reste moins performante que pour les autres.

— La figure 5.6.b décrit les courbes ROC des expérimentations impliquant Kaiten réalisant des attaques UDP Flood. Pour toutes les expérimentations, les courbes montrent ici aussi une très bonne qualité de détection de l'occurrence de l'attaque, pour les attaques à faible débit comme pour celle à haut débit. En effet, les AUC respectives aux expérimentations décrites sont : 0.93, 0.94, 0.93, 0.93 et 0.92.

— La figure 5.7.a et 5.7.b représentent l'évolution des valeurs de ACC en fonction des valeurs de H des expérimentations impliquant Hybrid_V1.0 et Kaiten respectivement. Pour les deux cas, nous constatons une baisse significative des valeurs de ACC avec l'évolution de H. En effet, ces dernières sont au dessus de 80% pour $H \leq 1.3$ et au dessus de 60% pour $1.3 \leq H \leq 1.43$. ACC continue de baisser jusqu'à atteindre des valeurs très faibles à partir de $H \geq 1.7$.

— La figure 5.8.a représente l'évolution des valeurs du MCC en fonction des valeurs de

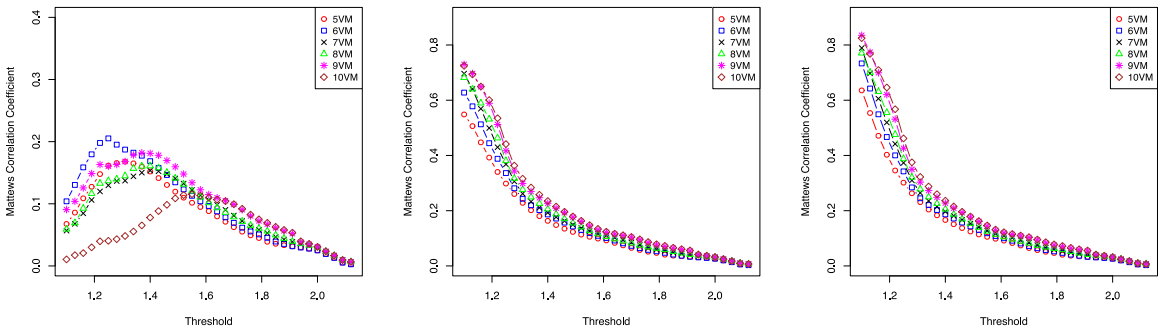


FIGURE 5.4 – Valeurs du MCC des expérimentations de Hybrid_V1.0 (a) 8Mb/s (b) 56 Mb/s (c) 80 Mb/s

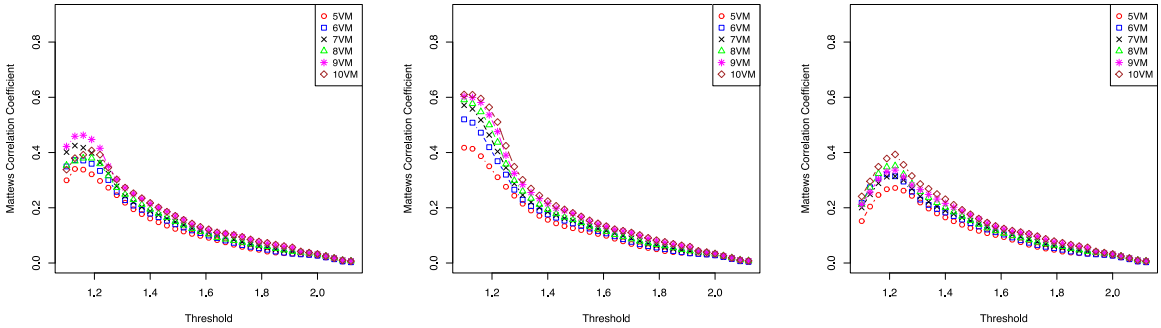


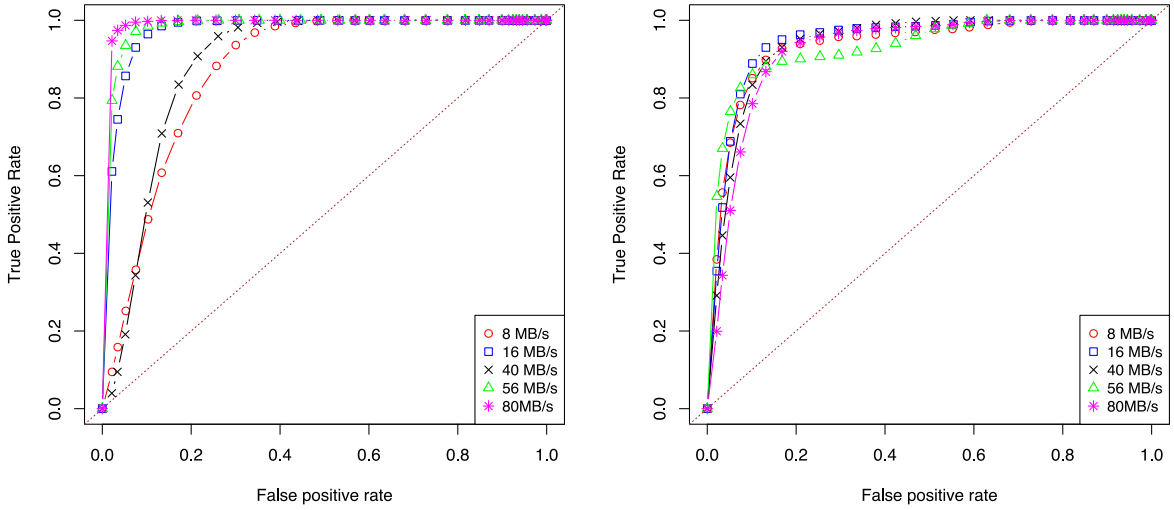
FIGURE 5.5 – Valeurs de MCC des expérimentations de Kaiten (a) 8Mb/s (b) 56 Mb/s (c) 80 Mb/s

H des expérimentations impliquant Hybrid_V1.0. Les meilleures valeurs obtenues pour MCC sont 0.17, 0.45, 0.21, 0.54 et 0.63. Ces valeurs reflètent une bonne détection pour les expérimentations des attaque à 16 Mb/s, 56 Mb/s et 80Mb/s. une détection moyenne pour le cas de 40 Mb/s et une mauvaise détection pour les attaques à faible débit comme celle à 8 Mb/s avec un taux élevé de faux négatifs. Ces résultats confirment que plus le débit d'attaque est élevé, meilleure est sa détection.

— La figure 5.8.b représente l'évolution des valeurs du MCC en fonction des valeurs de H des expérimentations impliquant Kaiten. Les meilleures valeurs obtenues pour MCC sont respectivement : 0.34, 0.34, 0.30, 0.42 et 0.27. Ces résultats reflètent une détection stable sur les débits sauf pour le cas de l'attaque à 80 Mb/s où la détection est moyenne. Cependant le problème de faux négatifs est toujours persistant.

Les valeur de ACC correspondant aux valeurs de MCC décrites ci-dessus sont respectivement : 0.78, 0.97, 0.82, 0.97 et 0.97 pour le cas de Hybrid_V1.0 et 0.95, 0.94, 0.92, 0.97 et 0.92 pour le cas de Kaiten, ce qui reflète une bonne exactitude des détections réalisées.

Comme décrit précedemment, l'étude du MCC permet également la détermination



(a) (b)

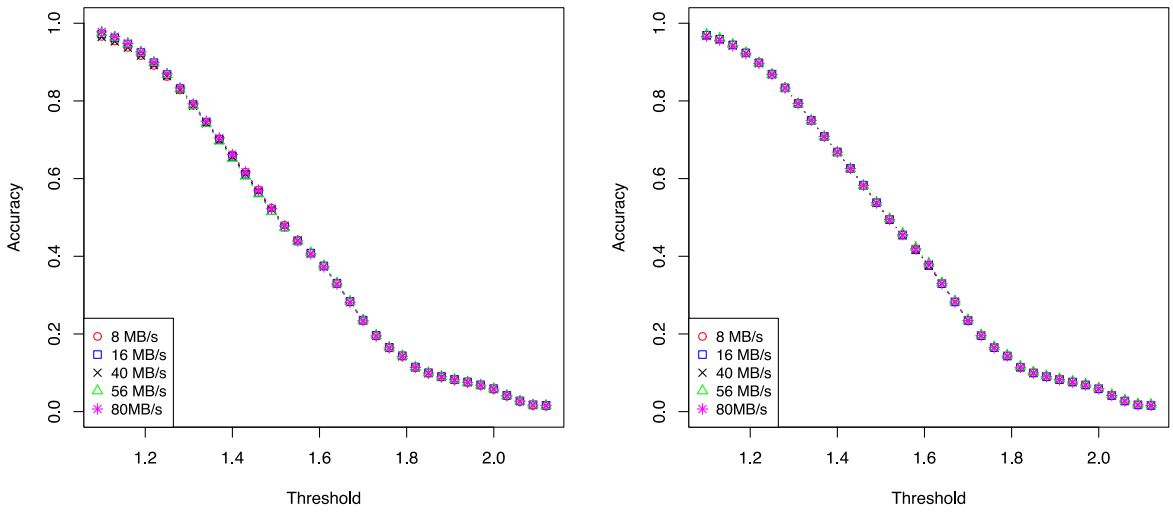
FIGURE 5.6 – ROC des expérimentations de l'attaque UDP Flood avec $n = 5$; (a) Hybrid_V1.0 ; (b) Kaiten

du seuil optimal de détection. Pour Hybrid_V1.0, les valeurs optimales de H pour les différents débits sont respectivement : 1.31, 1.10, 1.28, 1.10 et 1.10, conduisant ainsi à différentes valeurs de H pour différents débits. Ces dernières sont très proches pour les cas à 8 Mb/s et 45 Mb/s, mais éloignées des autres cas (16 Mb/s, 56 Mb/s et 80 Mb/s) qui ont tous un seuil optimal de $H = 1.10$. En outre, ces valeurs sont aussi très éloignées des valeurs obtenues pour ce même *botcloud* dans le cas de l'approche centralisée. Ceci s'explique par la différence entre l'espace factoriel calculé sur un petit ensemble d'individus et celui calculé sur la totalité des données, tel que réalisé dans l'approche centralisée.

Dans le cas de Kaiten, les valeurs optimales de H pour les différents débits d'attaque sont respectivement : 1.13, 1.16, 1.19, 1.10 et 1.19. Nous constatons par contre que la valeur optimale de H est peu variable, et se trouve dans un intervalle restreint $[1.10, 1.19]$. Cet intervalle est également très proche de celui où se situent les valeurs optimale de H pour la majorité des expérimentations impliquant Hybrid_V1.0 montrant ainsi la capacité du système de détection distribué à travailler indifféremment sur l'un ou l'autre *botcloud*.

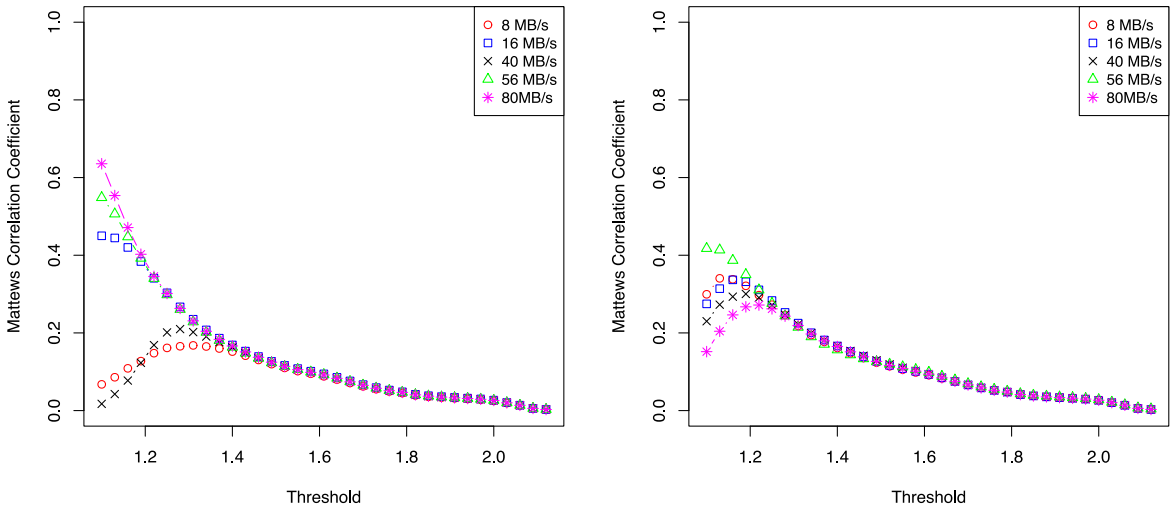
5.5 Conclusion

Dans ce chapitre, nous avons proposé une version distribuée pour la détection des *botclouds*. Cette dernière repose sur le même principe que l'approche centralisée décrite dans le chapitre 4, à savoir le calcul de la distance entre l'espace factoriel décrivant l'activité d'un *tenant* et des références des attaques. Nous avons produit une évaluation de cette approche qui repose sur des données réelles, et montre son efficacité et précision dans la détection d'attaques et cela pour plusieurs scénarios possibles impliquant une charge



(a) (b)

FIGURE 5.7 – ACC des expérimentations de l'attaque UDP Flood avec $n = 5$; (a) Hybrid_V1.0 ; (b) Kaiten



(a) (b)

FIGURE 5.8 – MCC des expérimentations de l'attaque UDP Flood avec $n = 5$; (a) Hybrid_V1.0 ; (b) Kaiten

totale de plus de 500 VMs.

Bien que l'approche présente une bonne qualité de détection, cette dernière souffre de quelques limites. Premièrement, un problème de faux négatifs qui pourrait dégrader globalement la qualité de détection, surtout dans le cas des attaques à faibles débits. Deuxièmement, le choix du seuil de dissimilarité qui se trouve dans la majorité des cas dans un intervalle très réduit, impacte fortement la qualité de la détection. Ainsi, certains débits pourront être bien détectés mais d'autres moins. Pour remédier à ce problème, nous envisageons de remplacer la signature statique par une signature modélisée sous la forme d'une distribution statistique.

Conclusion générale

Synthèse

Contexte et problématique

Depuis ces dernières années, le *cloud computing* fait partie intégrante du quotidien des individus et des organisations, et représente un acteur indispensable pour la production des services utilisés par ces usagers. Cette forte adoption du *cloud* tient son mérite à quelques caractéristiques fortes telles que le déploiement rapide des services, la réduction des coûts d'infrastructure et d'exploitation, la facturation à l'usage et l'évolutivité des infrastructures support.

Malgré ces avantages, la sécurité demeure une des principales préoccupations quant à la mise en oeuvre de services et données dans le *cloud* et de nombreuses études scientifiques et solutions techniques ont été proposées pour assurer la sécurité des utilisateurs et des infrastructures contre toute forme d'attaques et d'intrusions. Toutefois, au delà de ces efforts, il apparait que les avantages offerts par le *cloud* pour ses utilisateurs légitimes, permettent également à des utilisateurs malicieux d'en bénéficier afin de déployer facilement leurs attaques envers des tiers connectés à Internet. Cette utilisation malicieuse transforme ainsi le *cloud* en vecteur et support pour ces attaques.

Les plus grands bénéficiaires de cette exploitation sont les *botclouds*. En comparaison des *botnets* classiques, les *botclouds* présentent plus d'avantages. En effet, si la construction d'un *botnet*, qui consiste à prendre le contrôle de machines à l'insu de leurs usagers, nécessite un temps conséquent, exprimé en mois voire années, un *botcloud* peut être fonctionnel en quelques minutes. En outre, si un *botnet* n'est pas toujours fiable en raison du comportement de l'usager qui contrôle la machine hôte, un *botcloud* est toujours en ligne et prêt à l'usage. Enfin, si un *botnet* ne peut pas utiliser pleinement les ressources matérielles ou la bande passante de la machine hôte par souci de furtivité, un *botcloud* peut être pleinement utilisé sans crainte d'interruption. Pour un fournisseur de services *cloud*, être capable de le détecter et endiguer au plus proche de sa source est donc un enjeu essentiel pour ne pas en subir les dommages collatéraux et ne pas apparaitre comme la source d'activités malveillantes sur l'Internet.

Contributions

Le travail mené dans cette thèse à consisté à concevoir, mettre en oeuvre et valider une solution de détection distribuée et collaborative pour la détection des attaques DDoS perpétrées par des *botclouds* dans un contexte de *cloud* public. Pour ce faire, nous avons en

premier lieu réalisé une caractérisation des *botclouds* pour comprendre leur comportement. Afin d'acquérir des données qui permettent une telle étude, nous avons conduit une large campagne de mesures où nous avons réalisé un ensemble d'expérimentations visant la reproduction *in situ* des attaques DDoS de types UDP Flood et TCP SYN Flood via deux implémentations logicielles différentes de *botclouds*, à savoir Hybrid_V1.0 et Kaiten.

La caractérisation a été réalisée en deux temps en utilisant l'Analyse en Composantes Principales comme méthode d'analyse multidimensionnelles des données. Une première caractérisation du comportement système global d'un *botcloud* a mis en évidence le comportement et les corrélations de ses paramètres systèmes. Ensuite, la projection des individus résultats de ces ACP, a montré qu'il est possible de séparer les périodes d'attaques de celles de repos pour toutes les expérimentations menées. Dans la deuxième phase de cette caractérisation, nous nous sommes intéressés à la seule phase d'attaque. Nous avons montré que le comportement système d'un *botcloud* est très peu variable quelque soit le débit d'attaque et quelle que soit l'implémentation logicielle. Nous avons ainsi exploité ces invariants comportementaux d'un *botcloud* pour construire des références caractéristiques qui servent de signatures pour la détection de ces attaques. Une référence est représentée par un espace factoriel construit grâce aux matrices de vecteurs propres résultats des différentes ACP. Sur cette base, nous avons ensuite proposé un algorithme de détection qui repose sur l'ACP et qui propose d'évaluer la distance de l'activité d'un tenant face à cette signature.

Compte tenu des caractéristiques des infrastructures de *cloud computing*, de la taille de leur charge de travail ainsi que de la complexité de l'approche proposée, la considération d'une approche centralisée représente une véritable limite ainsi qu'un point de défaillance unique. Afin de palier à cette limite, nous avons proposé une approche distribuée collaborative fondée sur un réseau pair-à-pair qui repose sur une architecture composée de multiples structures hiérarchiques, maintenues par une table de hachage distribuée. Cette solution de détection repose sur le même principe que l'approche centralisée. Cependant, les calculs se font sur de petits groupes de VMs appelés *Cliques* qui sont combinés dans une phase d'agrégation des résultats afin de déterminer si à l'échelle d'un tenant, une attaque a lieu ou non. Nous avons évalué l'ensemble de ces propositions en se reposant sur différents indicateurs statistiques et en utilisant les données réelles collectées lors de la campagne expérimentale. Les résultats de cette évaluation ont montré la très bonne capacité de l'approche à détecter l'occurrence des attaques et cela pour l'ensemble des expérimentations.

En outre, la solution développée ici répond au différents besoins exprimés au début de ce manuscrit. Ainsi, (1) elle permet d'éviter les dommages collatéraux d'une attaque ayant un débit conséquent grâce à son implémentation à la source ; (2) elle considère des métriques système génériques qui peuvent être exploitées pour détecter d'autres d'attaques ; (3) elle est peu coûteuse grâce à la considération d'un nombre réduit de paramètres ; et enfin, (4) elle supporte le facteur d'échelle grâce à son architecture distribuée. On note toutefois que cette approche souffre de quelques limites et notamment celui d'un surcoût de faux négatifs qui nuisent à sa qualité globale de détection, surtout dans le cas des attaques à faibles débits. Une autre limite de cette approche réside dans le choix du seuil de dissimilarité à fixer comme paramètre de différenciation de l'activité légitime de la malicieuse.

Perspectives

Le travail que nous avons mené dans le cadre de cette thèse doit être considéré comme une première étape dans la mise en oeuvre de solutions de détection à la source dans un environnement de *cloud computing*. Ainsi, il ouvre de nombreuses perspectives de travaux futurs.

Dans la continuité de ce travail, nous envisageons : à court terme, l'extension de l'étude à d'autres formes d'attaques telles que l'attaque par force brute ou *VM leakeage* ; à moyen terme, de traiter spécifiquement la problématique du volume et de la vélocité des données à traiter pour exécuter une approche de détection similaire à la notre ; enfin, à long terme, d'implémenter notre solution dans le cadre d'un IDS qui puisse opérer dans un cadre réel.

Extension de l'étude

Le *cloud* représente un ensemble de technologies combinées. De ce fait, il est sensible aux vulnérabilités de chacune d'entre elles et dans la suite de ce travail, nous envisageons ainsi l'extension de notre étude afin de considérer deux autres cas qui y sont associés. Le premier cas porte sur l'attaque *VM leakeage* que nous avons décrite sans la section 1.4.2.3 et qui est liée spécifiquement aux technologies *cloud*. Cette attaque consiste à cartographier une infrastructure de *cloud* pour pouvoir, par l'utilisation de canaux latéraux, inférer l'activité d'une VM cible. Le second cas porte sur les attaques par force brute qui sont rendus possibles par l'exploitation des ressources colossales fournies par une infrastructure de *cloud*.

Pour chacun de ces cas, la démarche méthodologique que nous proposons est similaire au travail présenté ici. A savoir : (1) la reproduction de ce type d'attaque ; (2) la caractérisation de ses différentes phases (par exemple, la phase de cartographie et la phase d'accès à la cible en exploitant les canaux latéraux pour une attaque de type *VM leakeage*) ; (3) l'adaptation de la méthode de détection proposée dans ce manuscrit à ce nouveau cadre applicatif ; et enfin, (4) l'évaluation des performances et l'efficacité de détection de ces attaques.

La détection des attaques décrites ci-dessus s'avère plus complexe que celle des attaques de déni de services présentées ici. En effet, dans notre cadre de travail, nous nous sommes reposé sur la corrélation et la similarité de comportement d'un ensemble de VMs. Cependant, les attaques *VM leakage* et de force brute reposent généralement sur un nombre très réduit de VMs ce qui rend leur activité très diluée dans la charge légitime et donc leur détection plus difficile. Dans ce contexte, un usage comportemental (a contrario de signature) de l'analyse en composantes principales pourrait être considéré.

Porter l'étude dans un contexte *Big Data*

Afin de rendre notre approche de détection compatible avec les contraintes de facteur d'échelle inhérentes au contexte du *cloud*, nous avons proposé une approche distribuée de notre solution qui repose sur un réseau de recouvrement pair-à-pair. Une deuxième solution consisterait non pas à distribuer le plus fortement les traitements, mais à intégrer son algorithme dans un contexte *Big Data*. On rappelle que le volume de données générées dans notre campagne de tests dépasse la trentaine de giga-octets pour environ un millier

de VMs surveillées pendant 45 heures de surveillance au total sur l'ensemble des expérimentations. Dans la suite de notre étude, nous envisageons ainsi de porter notre solution dans un contexte *Big Data* en étudiant notamment son support de charges plus fortes et en utilisant des outils dédiés. Plus précisément, nous envisageons une implémentation de notre algorithme selon le modèle *Map-Reduce* [151] non pas pour une analyse de données en temps différé (traitement *batch*) mais en temps réel (traitement *streaming*).

Implémentation du travail réalisé

Pour évaluer l'approche proposée dans ce manuscrit, nous nous sommes reposés sur les données réelles collectées durant nos campagnes de mesures. Cependant, leur traitement a été effectué dans un simulateur au sein duquel nous avons implémenté nos algorithmes. Dans la suite de notre travail, nous envisageons de porter ce travail vers une implémentation réelle de notre approche. Cette contribution, au delà de la valorisation potentielle des résultats de ce travail de thèse permettrait aussi de pouvoir évaluer notre solution sur d'autres critères que la seule qualité de détection. On cite ainsi, le temps nécessaire à la détection d'une attaque, la taille des données nécessaires à sa mise en oeuvre (complexité spatiale) et la prise en compte des aléas associés à l'exécution dans un cadre réel.

Bibliographie

[1] Giannakouris Konstantinos and Smihily Maria. Cloud computing - statistics on the use by enterprises. Technical report, 2014.

[2] Md. Tanzim Khorshed, A.B.M. Shawkat Ali, and Saleh A. Wasimi. A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. *Future Generation Computer Systems*, 28(6) :833 – 851, 2012.

[3] M.G. Xavier, M.V. Neves, F.D. Rossi, T.C. Ferreto, T. Lange, and C.A.F. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 233–240. IEEE, 2013.

[4] OpenStack. Openstack manuals, openstack installation guide. Technical report, 2015.

[5] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2) :39–53, 2004.

[6] Akamai. Akamai's state of the internet / security. Technical report, December 2014.

[7] Yuh-Jye Lee, Yi-Ren Yeh, and Yu-Chiang Frank Wang. Anomaly detection via online oversampling principal component analysis. *Knowledge and Data Engineering, IEEE Transactions on*, 25 :1460–1470, 2013.

[8] Hayati Pedram, Jindou Jia, and Rvacheva Daria. botcloud an emerging platform for cyber-attacks. Technical report, 2012. http ://baesystemsdetica.blogspot.fr.

[9] Kassidy P. Clark, Martijn Warnier, and Frances M. T. Brazier. Botclouds - the future of cloud-based botnets ? In Frank Leymann, Ivan Ivanov, Marten J. van Sinderen, and Boris B. Shishkov, editors, *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER 2011)*, pages 597–603. Science and Technology Publications, 2011.

[10] Joep Ruiter and Martijn Warnier. Privacy regulations for cloud computing : Compliance and implementation in theory and practice. In Serge Gutwirth, Yves Poullet, Paul De Hert, and Ronald Leenes, editors, *Computers, Privacy and Data Protection : an Element of Choice*, pages 361–376. Springer Netherlands, 2011.

[11] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds : a berkeley view of cloud computing. *Commun. ACM*, 53(4) :50–58, 2010.

[12] A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1) :1 – 11, 2011.

[13] Peter mell and Timothy Grance. The NIST definition of cloud computing. volume 800, pages 1–7. Computer Security Devision, Information Laboratory, National Institute of Standards and Technology, 2011.

[14] Pascal Sauliere. Cloud computing et sécurité. Technical report, 2010.

[15] Minqi Zhou, Rong Zhang, Dadan Zeng, and Weining Qian. Services in the cloud computing era : A survey. In *Universal Communication Symposium (IUCS), 2010 4th International*, pages 40–46. IEEE, Oct 2010.

[16] B.P. Rimal, Eunmi Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, pages 44 –51. IEEE, 2009.

[17] Ronald L Krutz and Russell Dean Vines. *Cloud Security : A Comprehensive Guide to Secure Cloud Computing*. John Wiley & Sons, 2010.

[18] C. Baun, M. Kunze, J. Nimis, and S. Tai. *Cloud Computing : Web-Based Dynamic IT Services*. Springer, 2011.

[19] R. Buyya, J. Broberg, and A.M. Goscinski. *Cloud Computing : Principles and Paradigms*. Wiley Series on Parallel and Distributed Computing. Wiley, 2010.

[20] G. Briscoe and A Marinos. Digital ecosystems in the clouds : Towards community cloud computing. In *Digital Ecosystems and Technologies, 2009. DEST '09. 3rd IEEE International Conference on*, pages 103–108. IEEE, 2009.

[21] Susanta Nanda Tzi-cker Chiueh and Stony Brook. A survey on virtualization technologies. *RPE Report*, pages 1–42, 2005.

[22] J. White and A. Pilbeam. A Survey of Virtualization Technologies With Performance Testing. *ArXiv e-prints*, 2010.

[23] Pascale Primet, Jean-Patrick Gelas Vicat-Blanc, Olivier Mornard, Dinil Mon Divakaran, Pierre Bozonnet, Mathieu Jan, Vincent Roca, and Lionel Giraud. Hipcal : State of the art of os and network virtualization solutions for grids. 14 :38, 2007.

[24] Daniel Nurmi, Richard Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 124–131. IEEE, 2009.

[25] Stefan Wind. Open source cloud computing management platforms : Introduction, comparison, and recommendations for implementation. In *Open Systems (ICOS), 2011 IEEE Conference on*, pages 175–179. IEEE, 2011.

[26] Rafael Moreno-Vozmediano, Rubén S. Montero, and Ignacio M. Llorente. Iaas cloud architecture : From virtualized datacenters to federated cloud infrastructures. *Computer*, 45(12) :65–72, 2012.

[27] Xiaolin Li and Judy Qiu. *Cloud Computing for Data-Intensive Applications*. Springer, 2014.

[28] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Computer Security – ESORICS 2009*, volume 5789 of *Lecture Notes in Computer Science*, pages 355–370. Springer Berlin Heidelberg, 2009.

[29] B.R. Kandukuri, V.R. Paturi, and A Rakshit. Cloud security issues. In *Services Computing, 2009. SCC '09. IEEE International Conference on*, pages 517–520. IEEE, 2009.

[30] NIST Cloud Computing Standards Roadmap Working Group, NIST Cloud Computing Standards Roadmap Working Group, et al. Nist cloud computing standards roadmap. Technical report, 2013.

[31] Greg Ness. 3 major barriers to cloud computing. Technical report, 2009.

[32] Neal Leavitt. Is cloud computing really ready for prime time ? *Computer*, 42(1) :15–20, 2009.

[33] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing v3.0. Technical report, 2011.

[34] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing v2.1. Technical report, 2009.

[35] V. Choudhary. Software as a service : Implications for investment in software development. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 1–10, 2007.

[36] D. Chappell. *Enterprise Service Bus*. O'Reilly Media, Incorporated, 2004.

[37] Jothy Rosenberg and David Remy. *Securing Web Services with WS-Security : Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Pearson Higher Education, 2004.

[38] Brian Wilkinson and Demed L'Her. Mastering SOA, part 2 : Wiring through an enterprise service bus, 2006.

[39] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and 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*, CCS '09, pages 199–212. ACM, 2009.

[40] Chi-Chun Lo, Chun-Chieh Huang, and J. Ku. A cooperative intrusion detection system framework for cloud computing networks. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 280 –284. IEEE, 2010.

[41] A.V. Dastjerdi, K.A. Bakar, and S.G.H. Tabatabaei. Distributed intrusion detection in clouds using mobile agents. In *Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP '09. Third International Conference on*, pages 175 –180. IEEE, 2009.

[42] Hatem Hamad and Mahmoud Al-Hoby. Managing intrusion detection as a service in cloud networks. *International Journal of Computer Applications*, 41(1) :35–40, 2012.

[43] K. Vieira, A. Schulter, C.B. Westphall, and C.M. Westphall. Intrusion detection for grid and cloud computing. *IT Professional*, 12(4) :38 –43, july-aug. 2010.

[44] S. Roschke, Feng Cheng, and C. Meinel. Intrusion detection in the cloud. In *Dependable, Autonomic and Secure Computing, 2009. DASC '09. Eighth IEEE International Conference on*, pages 729 –734. IEEE, 2009.

[45] Irfan Gul and M. Hussain. Distributed cloud intrusion detection model. *International Journal of Advanced Science and Technology*, 34 :1 –12, 2011.

[46] Breaking encryption in the cloud : Gpu accelerated supercomputing for everyone, 2012. http ://secdocs.lonerunners.net/documents/details/6482-breaking-encryption-in-the-cloud-gpu-accelerated-supercomputing-for-everyone.

[47] Sérgio S.C. Silva, Rodrigo M.P. Silva, Raquel C.G. Pinto, and Ronaldo M. Salles. Botnets : A survey. *Computer Networks*, 57(2) :378 – 403, 2013. Botnet Activity : Analysis, Detection and Shutdown.

[48] S. Khattak, N.R. Ramay, K.R. Khan, A.A. Syed, and S.A. Khayam. A taxonomy of botnet behavior, detection, and defense. *Communications Surveys Tutorials, IEEE*, 16(2) :898–924, 2014.

[49] M. Feily, A. Shahrestani, and S. Ramadass. A survey of botnet and botnet detection. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*, pages 268–273. IEEE, 2009.

[50] M. Eslahi, R. Salleh, and N.B. Anuar. Bots and botnets : An overview of characteristics, detection and challenges. In *Control System, Computing and Engineering (ICCSCE), 2012 IEEE International Conference on*, pages 349–354. IEEE, 2012.

[51] Gianluca Stringhini, Thorsten Holz, Brett Stone-Gross, Christopher Kruegel, and Giovanni Vigna. Botmagnifier : Locating spambots on the internet. In *USENIX Security Symposium*, pages 1–32, 2011.

[52] Johannes M Bauer, Michel JG Van Eeten, and T Chattopadhyay. Itu study on the financial aspects of network security : Malware and spam. *ICT Applications and Cybersecurity Division, International Telecommunication Union, Final Report, July*, 2008.

[53] Joshua Davis. Hackers take down the most wired country in europe. *Wired Magazine*, 15(9) :15–09, 2007.

[54] Stephen Herzog. Revisiting the estonian cyber attacks : Digital threats and multinational responses. *Journal of Strategic Security*, 4(2) :4, 2011.

[55] Ping Wang, Lei Wu, B. Aslam, and C.C. Zou. A systematic study on peer-to-peer botnets. In *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, pages 1–8. IEEE, 2009.

[56] Ping Wang, S. Sparks, and C.C. Zou. An advanced hybrid peer-to-peer botnet. *Dependable and Secure Computing, IEEE Transactions on*, 7 :113–127, 2010.

[57] Evan Cooke, Farnam Jahanian, and Danny McPherson. The zombie roundup : Understanding, detecting, and disrupting botnets. In *Proceedings of the USENIX SRUTI Workshop*, volume 39, page 44, 2005.

[58] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. Botminer : Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *USENIX Security Symposium*, volume 5, pages 139–154, 2008.

[59] Christophe Kalt. Rfc 2810-internet relay chat : Architecture. *Internet Engineering Task Force (IETF)*, 2000.

[60] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Rfc 2616, hypertext transfer protocol–http/1.1. *Internet Engineering Task Force (IETF)*, 2009.

[61] Márk Jelasity and Vilmos Bilicki. Towards automated detection of peer-to-peer botnets : On the limits of local approaches. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[62] T. Risse, P. Knezevic, and A. Wombacher. P2P evolution : from file-sharing to decentralized workflows. *IT Information Technology*, 46 :193–199, 2004.

[63] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160. ACM, 2001.

[64] Antony Rowstron and Peter Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer Berlin Heidelberg, 2001.

[65] Ben Yanbin Zhao, John Kubiatowicz, Anthony D Joseph, et al. Tapestry : An infrastructure for fault-tolerant wide-area location and routing. pages 1–28, 2001.

[66] Niels Provos et al. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, pages 1–13, 2004.

[67] Sergi Martinez-Bea, Sergio Castillo-Perez, and Joaquin Garcia-Alfaro. Real-time malicious fast-flux detection using dns and bot related features. In *PST*, pages 369–372, 2013.

[68] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou II, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots : Detecting the rise of dga-based malware. In *USENIX Security Symposium*, pages 491–506, 2012.

[69] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. Automatically generating models for botnet detection. In *Computer Security – ESORICS 2009*, volume 5789 of *Lecture Notes in Computer Science*, pages 232–249. Springer Berlin Heidelberg, 2009.

[70] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. Bothunter : Detecting malware infection through ids-driven dialog correlation. In *Usenix Security*, volume 7, pages 1–16, 2007.

[71] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale botnet detection and characterization. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, volume 7, 2007.

[72] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep : Finding P2P bots with structured graph analysis. In *USENIX Security Symposium*, pages 95–110, 2010.

[73] Junjie Zhang, R. Perdisci, Wenke Lee, Xiapu Luo, and U. Sarfraz. Building a scalable system for stealthy P2P-botnet detection. *Information Forensics and Security, IEEE Transactions on*, 9 :27–38, 2014.

[74] Jelena Mirkovic. *D-WARD : source-end defense against distributed denial-of-service attacks*. PhD thesis, University of California Los Angeles, 2003.

[75] Prolexic White Paper. DDoS boot camp : Basic training for an increasing cyber threat. Technical report, 2013.

[76] Chenfeng Vincent Zhou, Christopher Leckie, and Shanika Karunasekera. A survey of coordinated attacks and collaborative intrusion detection. *Computers and Security*, 29 :124 – 140, 2010.

[77] Jérôme François, Issam Aib, and Raouf Boutaba. Firecol : a collaborative protection network for the detection of flooding DDoS attacks. *IEEE/ACM Trans. Netw.*, 20(6) :1828–1841, 2012.

[78] Christos Douligeris and Aikaterini Mitrokotsa. DDoS attacks and defense mechanisms : classification and state-of-the-art. *Computer Networks*, 44(5) :643–666, 2004.

[79] Usman Tariq, Manpyo Hong, and Kyung-suk Lhee. A comprehensive categorization of DDoS attack and DDoS defense techniques. In *Advanced Data Mining and Applications, Second International Conference, ADMA 2006, Xi'an, China, August 14-16, 2006, Proceedings*, volume 4093, pages 1025–1036. Springer, 2006.

[80] Abbass Asosheh and Naghmeh Ramezani. A comprehensive taxonomy of DDoS attacks and defense mechanism applying in a smart classification. *WSEAS Transactions on Computers*, 7 :281–290, 2008.

[81] Arbor Networks. Worldwide infrastructure security report, volume ix. Technical report, December 2014.

[82] Angelos D Keromytis. Network bandwidth denial of service (dos). In *Encyclopedia of Cryptography and Security*, pages 836–838. Springer, 2011.

[83] Khatoun Rida. *Système multi-agents et architecture pair à pair pour la détection d'attaques de déni de service distribuées*. PhD thesis, Université de Technologie de Troyes, 2008.

[84] Ludovic Mé. *Audit de sécurité par algorithme génétique*. PhD thesis, Université de Renne 1, 1994.

[85] Teresa F Lunt, R Jagannathan, Rosanna Lee, Sherry Listgarten, David L Edwards, Peter G Neumann, Harold S Javitz, and Al Valdes. Ides : The enhanced prototype-a real-time intrusion-detection expert system. In *SRI International, 333 Ravenswood Avenue, Menlo Park*. Citeseer, 1988.

[86] M. Tavallaee, N. Stakhanova, and A.A. Ghorbani. Toward credible evaluation of anomaly-based intrusion-detection methods. *Systems, Man, and Cybernetics, Part C : Applications and Reviews, IEEE Transactions on*, 40(5) :516–524, 2010.

[87] P Ferguson and D Senie. Rfc 2827 (bcp38) : Network ingress filtering : Defeating denial of service attacks which employ ip source address spoofing. ietf. *Internet Engineering Task Force (IETF)*, 2000.

[88] C. Fung and R. Boutaba. *Intrusion Detection Networks : A Key to Distributed Security*. CRC Press, 2013.

[89] Daniel Joseph Dean, Hiep Nguyen, and Xiaohui Gu. Ubl : unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In *Proceedings of the 9th international conference on Autonomic computing*, ICAC '12, pages 191–200. ACM, 2012.

[90] P. Kannadiga and M. Zulkernine. Didma : a distributed intrusion detection system using mobile agents. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on*, pages 238–245. IEEE, 2005.

[91] Geetha Ramachandran and Delbert Hart. A P2P intrusion detection system based on mobile agents. In *Proceedings of the 42Nd Annual Southeast Regional Conference*, ACM-SE 42, pages 185–190. ACM, 2004.

[92] Phillip A Porras and Peter G Neumann. Emerald : Event monitoring enabling response to anomalous live disturbances. In *Proceedings of the 20th national information systems security conference*, pages 353–365, 1997.

[93] Ji Li, Dah-Yoh Lim, and Karen R Sollins. Dependency-based distributed intrusion detection. In *DETER*, 2007.

[94] L. Rabiner and B.H. Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3 :4–16, 1986.

[95] Abraham Wald. *Sequential analysis*. Courier Corporation, 1973.

[96] Abdoul Karim Ganame, Julien Bourgeois, Renaud Bidou, and Francois Spies. A global security architecture for intrusion detection on computer networks. *Computers and Security*, 27 :30 – 47, 2008.

[97] Arturo Servin and Daniel Kudenko. *Multi-agent reinforcement learning for intrusion detection*. Springer, 2008.

[98] Andrew G Barto. *Reinforcement learning : An introduction*. MIT press, 1998.

[99] M.E. Locasto, J.J. Parekh, A.D. Keromytis, and S.J. Stolfo. Towards collaborative security and P2P intrusion detection. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, pages 333–339. IEEE, 2005.

[100] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13 :422–426, 1970.

[101] Denver Dash, Branislav Kveton, John Mark Agosta, Eve Schooler, Jaideep Chandrashekar, Abraham Bachrach, and Alex Newman. When gossip is good : Distributed probabilistic inference for detection of slow network intrusions. In *Proceedings of the national conference on Artificial Intelligence*, volume 21, pages 1–8. MIT Press, 2006.

[102] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global intrusion detection in the domino overlay system. In *NDSS*, pages 1–16, 2004.

[103] Dayong Ye, Quan Bai, Minjie Zhang, and Zhen Ye. P2P distributed intrusion detections by using mobile agents. In *Computer and Information Science, 2008. ICIS 08. Seventh IEEE/ACIS International Conference on*, pages 259–265. IEEE, 2008.

[104] J. Mirkovic and P. Reiher. D-ward : a source-end defense against flooding denial-of-service attacks. *Dependable and Secure Computing, IEEE Transactions on*, 2 :216–232, 2005.

[105] Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection*, volume 2212 of *Lecture Notes in Computer Science*, pages 85–103. Springer Berlin Heidelberg, 2001.

[106] H Debar, D Curry, and B Feinstein. Rfc 4765 : The intrusion detection message exchange format (idmef), march 2007. *Internet Engineering Task Force (IETF)*, 2007.

[107] R Danyliw, J Meijer, and Y Demchenko. Rfc 5070 : The incident object description exchange format. *Internet Engineering Task Force (IETF)*, 2007.

[108] K. Moriarty. Rfc 6045 : Real-time inter-network defense (rid). *Internet Engineering Task Force (IETF)*, 2010.

[109] K. Moriarty. Rfc 6046 : Transport of real-time inter-network defense (rid) messages. *Internet Engineering Task Force (IETF)*, 2010.

[110] Etancelin, J.-M., Faure, S., Fevrier, T., and Huynh, C. Macroscopic modelization of the cloud elasticity. *ESAIM : Proc.*, 43 :136–146, 2013.

[111] A. Khan, X. Yan, Shu Tao, and N. Anerousis. Workload characterization and prediction in the cloud : A multiple time series approach. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 1287–1294. IEEE, 2012.

[112] A. Ganapathi, Yanpei Chen, A. Fox, R. Katz, and D. Patterson. Statistics-driven workload modeling for the cloud. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 87–92. IEEE, 2010.

[113] Arshdeep Bahga, Vijay Krishna Madisetti, et al. Synthetic workload generation for cloud computing applications. *Journal of Software Engineering and Applications*, 4 :396 – 410, 2011.

[114] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software : Practice and Experience*, 41 :23–50, 2011.

[115] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab : an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33 :3–12, 2003.

[116] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2 :37 – 52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

[117] Bouroche Jean-Marie and Saporta Gilbert. *Que sais je ? L'Analyse Des Données*. Presses Universitaires De France, 1979.

[118] Ling Huang, Xuanlong Nguyen, Minos Garofalakis, Michael Jordan, A Joseph, and Nina Taft. Distributed pca and network anomaly detection. *Submitted to NIPS*, pages 1–15, 2006.

[119] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *SIGCOMM Comput. Commun. Rev.*, 34(4) :219–230, 2004.

[120] CW Gardiner. *Stochastic methods*. Springer, 1985.

[121] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. Technical report, DTIC Document, 2003.

[122] Prasanta Chandra Mahalanobis. on the generalized distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2 :49–55, 1936.

[123] Huizhong Sun, Yan Zhaung, and H.J. Chao. A principal components analysis-based robust DDoS defense system. In *Communications, 2008. ICC '08. IEEE International Conference on*, pages 1663–1669. IEEE, 2008.

[124] D. Brauckhoff, K. Salamatian, and M. May. Applying pca for traffic anomaly detection : Problems and solutions. In *INFOCOM, IEEE*, pages 2866–2870. IEEE, 2009.

[125] Yacine Bouzida, Frederic Cuppens, Nora Cuppens-Boulahia, and Sylvain Gombault. Efficient intrusion detection using principal component analysis. In *3éme Conférence sur la Sécurité et Architectures Réseaux (SAR)*, page 12, 2004.

[126] J.R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3) :221 – 234, 1987.

[127] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3) :175–185, 1992.

[128] Guisong Liu, Zhang Yi, and Shangming Yang. A hierarchical intrusion detection model based on the pca neural networks. *Neurocomputing*, 70(7–9) :1561 – 1568, 2007. Advances in Computational Intelligence and Learning 14th European Symposium on Artificial Neural Networks 2006 14th European Symposium on Artificial Neural Networks 2006.

[129] David E Rumelhart, James L McClelland, PDP Research Group, et al. Parallel distributed processing : Explorations in the microstructures of cognition. volume 1 : Foundations. *MIT Press, Cambridge, MA*, 2 :560–567, 1986.

[130] Xin Xu and Xuening Wang. An adaptive network intrusion detection method based on pca and support vector machines. In *Advanced Data Mining and Applications*, volume 3584 of *Lecture Notes in Computer Science*, pages 696–703. Springer Berlin Heidelberg, 2005.

[131] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[132] Huan Liu. Amazon data center size, 2012.

[133] Guy Rosen. Anatomy of an amazon ec2 resource id. Technical report, 2009.

[134] Carl D Meyer. *Matrix analysis and applied linear algebra*, volume 2. Siam, 2000.

[135] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling : Theory and applications*. Springer, 2005.

[136] James M Lucas and Michael S Saccucci. Exponentially weighted moving average control schemes : properties and enhancements. *Technometrics*, 32(1) :1–12, 1990.

[137] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus AF Andersen, and Henrik Nielsen. Assessing the accuracy of prediction algorithms for classification : an overview. *Bioinformatics*, 16(5) :412–424, 2000.

[138] Data Artisans. Computing recommendations at extreme scale with apache flink and google compute engine. Technical report, 2013.

[139] Ramaprabhu Janakiraman, Marcel Waldvogel, and Qi Zhang. Indra : A peer-to-peer approach to network intrusion detection and prevention. In *Enabling Technologies : Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, pages 226–231. IEEE, 2003.

[140] H.A. Kholidy and F. Baiardi. Cids : A framework for intrusion detection in cloud systems. In *Information Technology : New Generations (ITNG), 2012 Ninth International Conference on*, pages 379–385. IEEE, 2012.

[141] Min Cai, Kai Hwang, Yu-Kwong Kwok, Shanshan Song, and Yu Chen. Collaborative internet worm containment. *IEEE Security and Privacy*, 3 :25–33, 2005.

[142] J. Newsome, B. Karp, and D. Song. Polygraph : automatically generating signatures for polymorphic worms. In *Security and Privacy, 2005 IEEE Symposium on*, pages 226–241. IEEE, 2005.

[143] Catherine Meadows, Radha Poovendran, Dusko Pavlovic, LiWu Chang, and Paul Syverson. Distance bounding protocols : Authentication logic analysis and collusion attacks. In *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks*, volume 30 of *Advances in Information Security*, pages 279–298. Springer US, 2007.

[144] C.V. Zhou, S. Karunasekera, and C. Leckie. A peer-to-peer collaborative intrusion detection system. In *Networks, 2005. Jointly held with the 2005 IEEE 7th Malaysia International Conference on Communication., 2005 13th IEEE International Conference on*, volume 1, pages 1–6. IEEE, 2005.

[145] Zhichun Li, Yan Chen, and Aaron Beach. Towards scalable and robust distributed intrusion alert fusion with good load balancing. In *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense*, LSAD '06, pages 115–122. ACM, 2006.

[146] Henrique Rodrigues, Jose Renato Santos, Yoshio Turner, Paolo Soares, and Dorgival Guedes. Gatekeeper : Supporting bandwidth guarantees for multi-tenant datacenter networks. In *WIOV*, pages 1–8, 2011.

[147] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32(3) :241–280, 1999.

[148] Rajiv Ranjan, Liang Zhao, Xiaomin Wu, Anna Liu, Andres Quiroz, and Manish Parashar. Peer-to-peer cloud provisioning : Service discovery and load-balancing.

In *Cloud Computing*, Computer Communications and Networks, pages 195–217. Springer London, 2010.

[149] Rajkumar Buyya, Rajiv Ranjan, and RodrigoN. Calheiros. Intercloud : Utility-oriented federation of cloud computing environments for scaling of application services. In *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Science*, pages 13–31. Springer Berlin Heidelberg, 2010.

[150] Wenwu Zhu, Chong Luo, Jianfeng Wang, and Shipeng Li. Multimedia cloud computing. *Signal Processing Magazine, IEEE*, 28 :59–69, 2011.

[151] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : Simplified data processing on large clusters. *Commun. ACM*, 51 :107–113, 2008.

Publications de thèse

Conférences internationales avec comité de lecture et actes

- Badis Hammi, Guillaume Doyen, Rida Khatoun. "Understanding Botclouds from a System Perspective : a Principal Component Analysis". In proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium (NOMS 2014), 8 pages, IEEE 2014. 5 - 9 May 2014, Krakow, Poland.
- Badis Hammi, Rida Khatoun, Guillaume Doyen. "A factorial space for a system-based detection of botcloud activity". In proceedings of the 6th IEEE/IFIP International Conference on New Technologies, Mobility and Security (NTMS 2014), 5 pages, IEEE 2014. 30 March – 2 April 2014, Dubai, UAE.
- Badis Hammi, Guillaume Doyen, Rida Khatoun. "Toward a source detection of botclouds : a PCA-based approach". In proceedings of the 8th IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS 2014), 12 pages LNCS, Springer Berlin Heidelberg 2014. June 30 - July 3, 2014, Brno, Czech Republic.
- Badis Hammi, Guillaume Doyen, Rida Khatoun. "A Collaborative Approach for a Source based Detection of Botclouds". In proceedings of the 14th International IEEE/IFIP Symposium on Integrated Network and Service Management (IM 2015), 4 pages (papier court), IEEE 2015. 11 - 15 May 2015, Ottawa, Canada.

Conférences francophones avec comité de lecture et actes

- Badis Hammi, Rida Khatoun, Guillaume Doyen. "Caractérisation des attaques DDoS générées par un Botcloud". in proceedings of the 8ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI), 2 pages (Poster). Mont de Marsan - Landes, France, 16 - 18 septembre 2013.

A

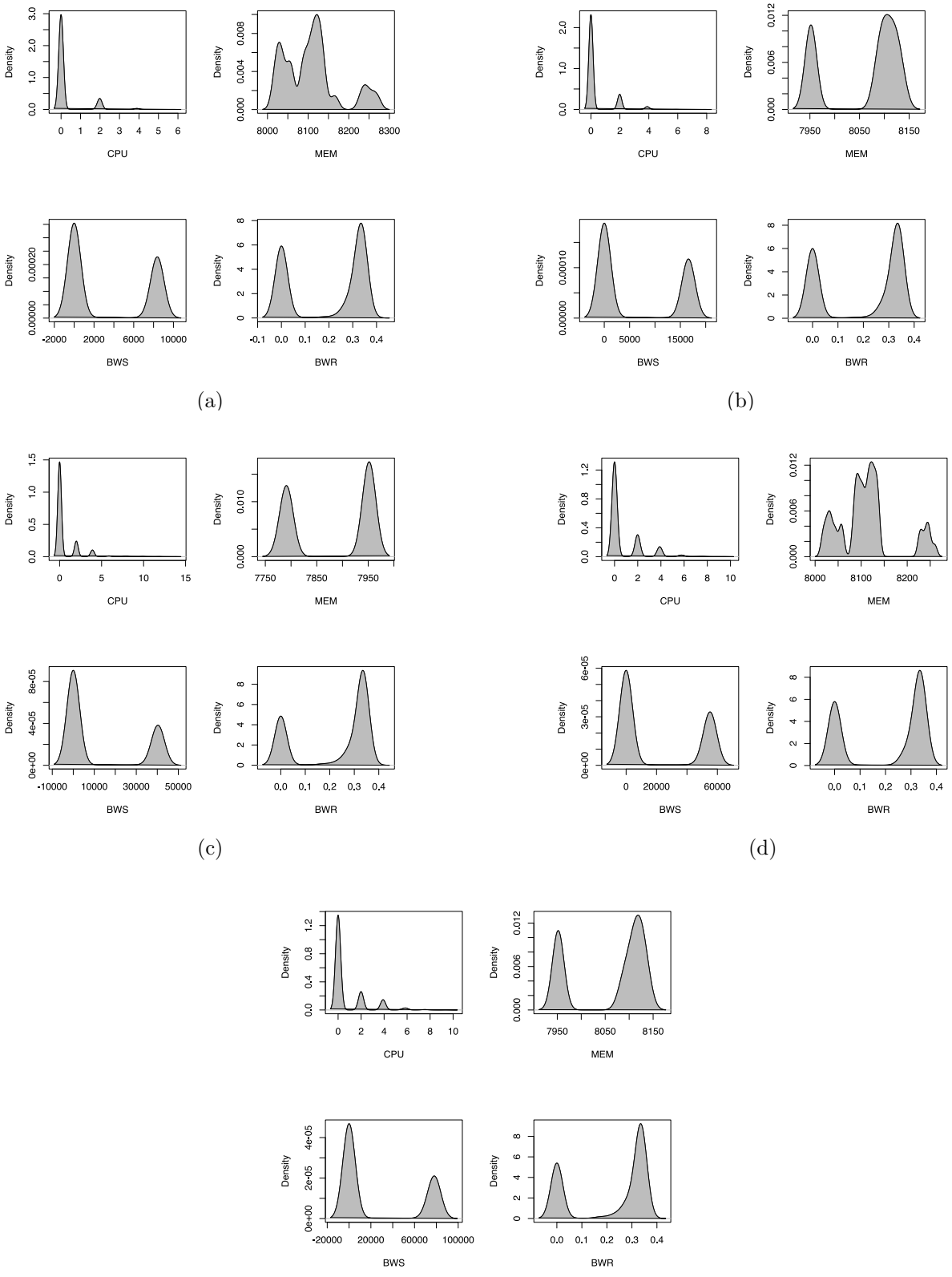
Résultats complets de la campagne de caractérisation du comportement système d'un *botcloud*

Dans le chapitre 3, nous avons produit une caractérisation du comportement global d'un *botcloud* réalisant des attaques DDoS. Cette caractérisation a été présentée en trois phases : (1) analyse de la distribution statistique des paramètres ; (2) analyse des corrélations des paramètres ; (3) analyse du comportement des individus résultats des ACP. Afin de produire une illustration des résultats obtenus, nous avions choisi, dans ce chapitre, les deux cas : (1) le *botcloud* Hybrid_V1.0 réalisant une attaque TCP SYN Flood à 290 paquets/s et (2) le *botcloud* Kaiten réalisant une attaque UDP Flood à 56 Mb/s. Dans cette annexe nous présenterons les résultats obtenus pour l'ensemble des expérimentations réalisées et pour les trois phases de caractérisation.

Le tableau A.1 décrit les différentes figures présentées ci-après.

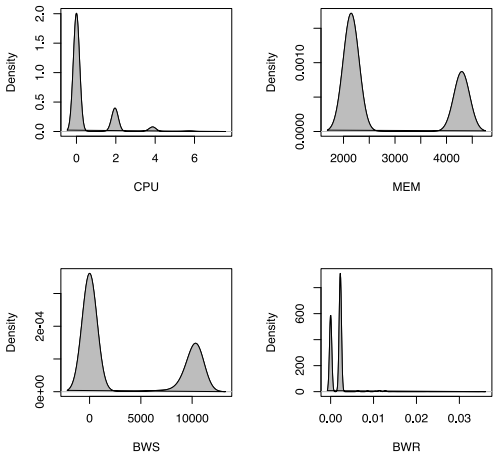
Attaque	**Botnet**	**Type de résultat**	**Figure**
UDP Flood	Hybrid_V1.0	Distribution statistique	Figure A.1
		Cercle de corrélations	Figure A.4
		Projection d'individus	Figure A.7
	Kaiten	Distribution statistique	Figure A.2
		Cercle de corrélations	Figure A.5
		Projection d'individus	Figure A.8
TCP SYN Flood	Hybrid_V1.0	Distribution statistique	Figure A.3
		Cercle de corrélations	Figure A.6
		Projection d'individus	Figure A.9

Table A.1 – Description des figures

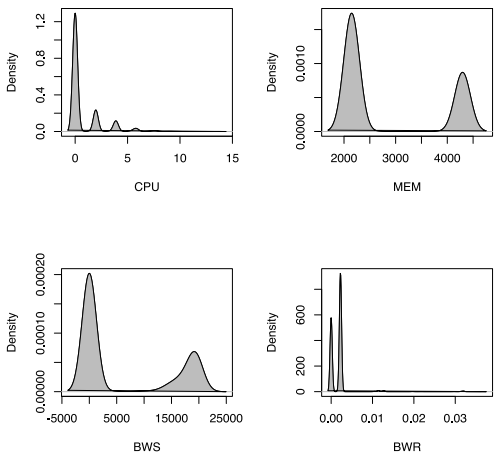


(a) (b) (c) (d) (e)

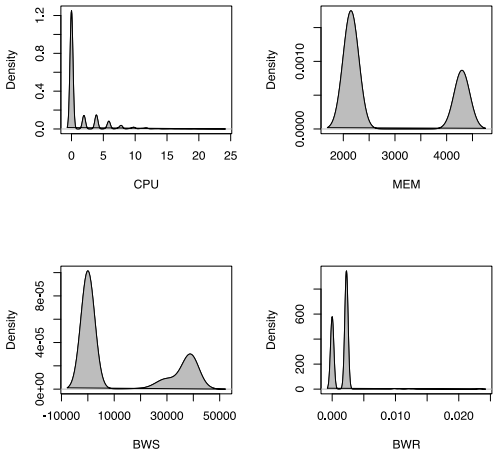
FIGURE A.1 – Distributions statistiques des métriques du *botnet*Hybrid_V1.0 réalisant des attaques UDP Flood : (a) 8 Mb/s ; (b) 16 Mb/s ; (c) 40 Mb/s ; (d) 56 Mb/s ; (e) 80 Mb/s



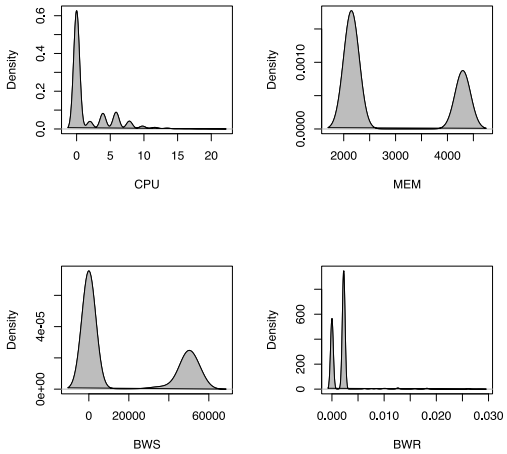
(a)



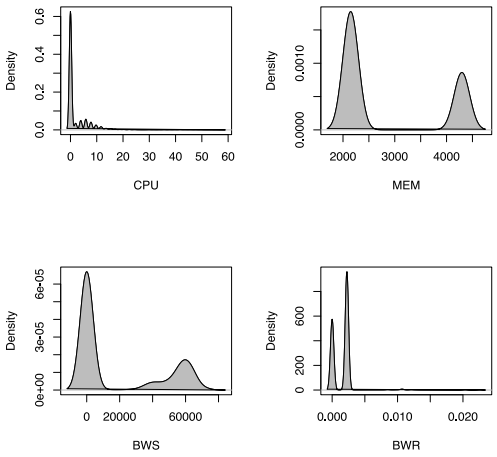
(b)



(c)



(d)



(e)

FIGURE A.2 – Distributions statistiques des métriques du *botnet*Kaiten réalisant des attaques UDP Flood : (a) 8 Mb/s ; (b) 16 Mb/s ; (c) 40 Mb/s ; (d) 56 Mb/s ; (e) 80 Mb/s

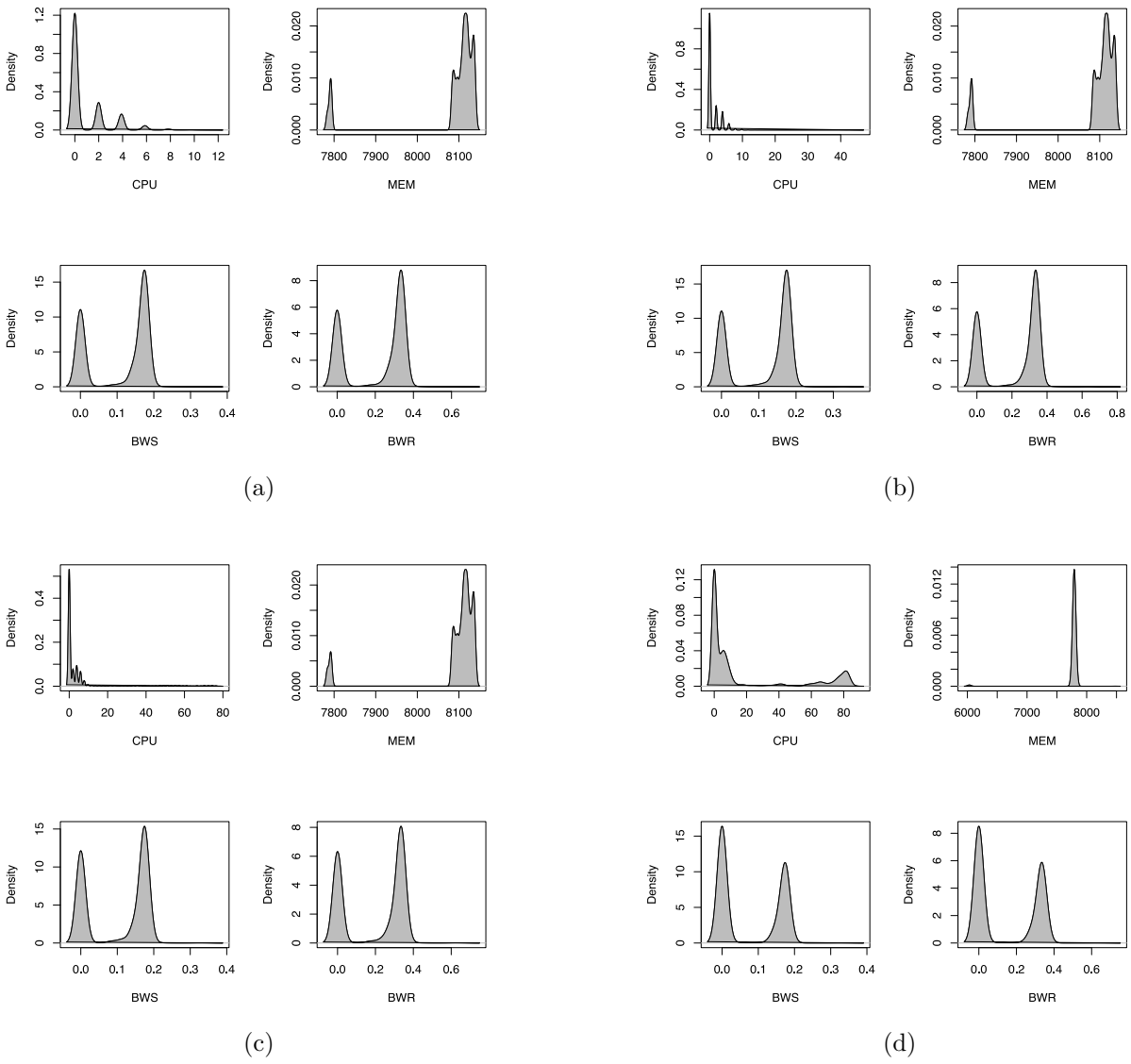
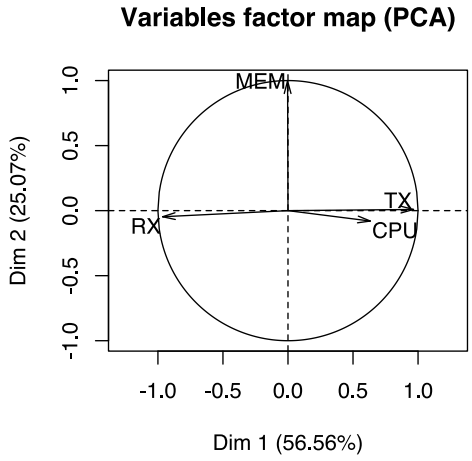
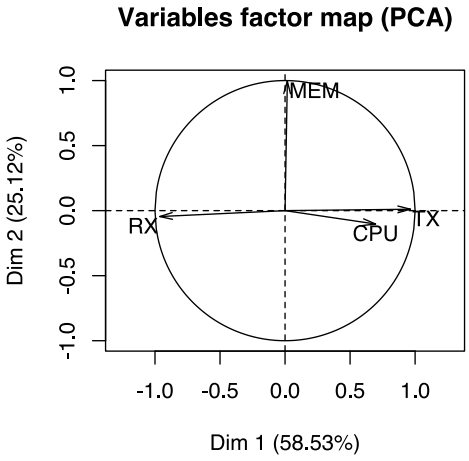


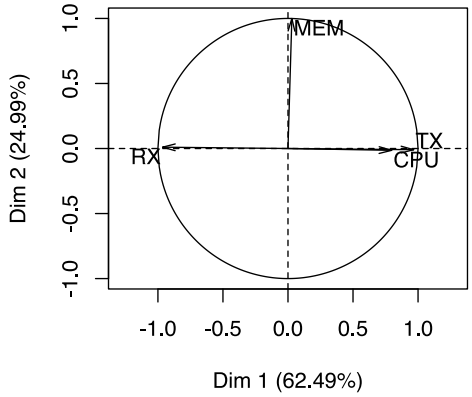
FIGURE A.3 – Distributions statistiques des métriques du *botnet*Hybrid_V1.0 réalisant des attaques TCP SYN Flood : (a) 159 p/s ; (b) 171 p/s ; (c) 290 p/s ; (d) 420 p/s



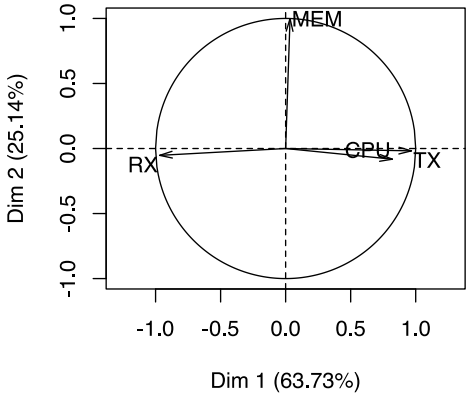
(a)



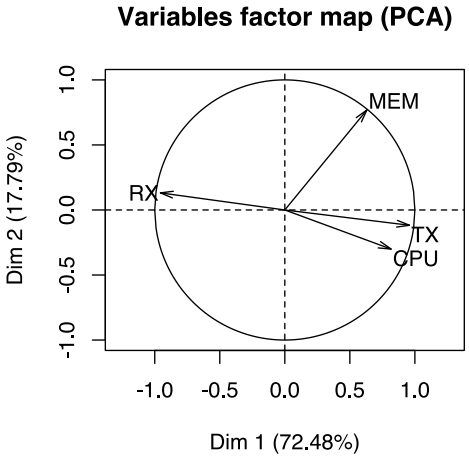
(b)



(c)



(d)



(e)

FIGURE A.4 – Cercles de corrélations des ACP réalisées sur le *botnet*Hybrid_V1.0 réalisant des attaques UDP Flood : (a) 8 Mb/s ; (b) 16 Mb/s ; (c) 40 Mb/s ; (d) 56 Mb/s ; (e) 80 Mb/s

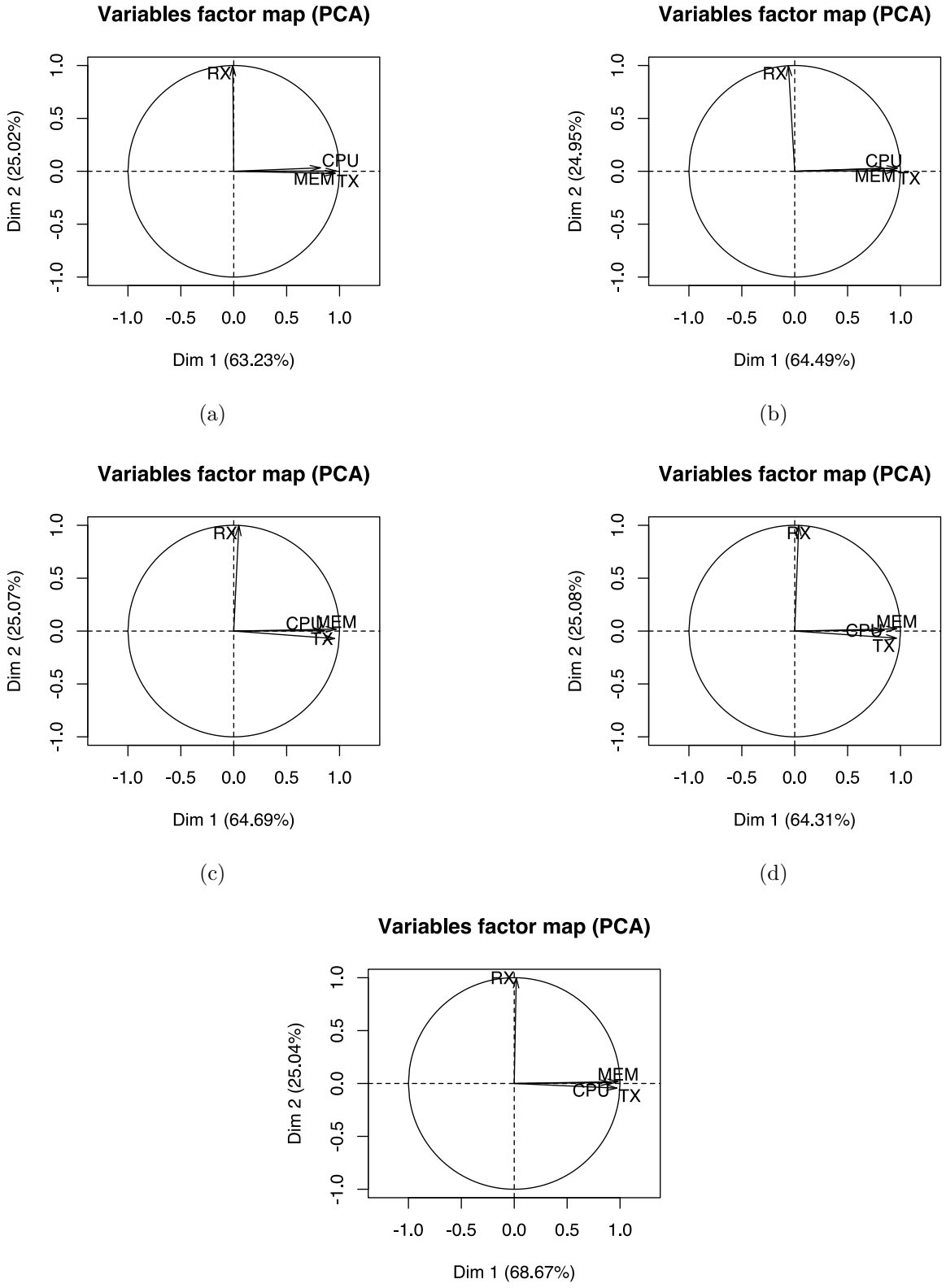


FIGURE A.5 – Cercles de corrélations des ACP réalisées sur le *botnet*Kaiten réalisant des attaques UDP Flood : (a) 8 Mb/s ; (b) 16 Mb/s ; (c) 40 Mb/s ; (d) 56 Mb/s ; (e) 80 Mb/s

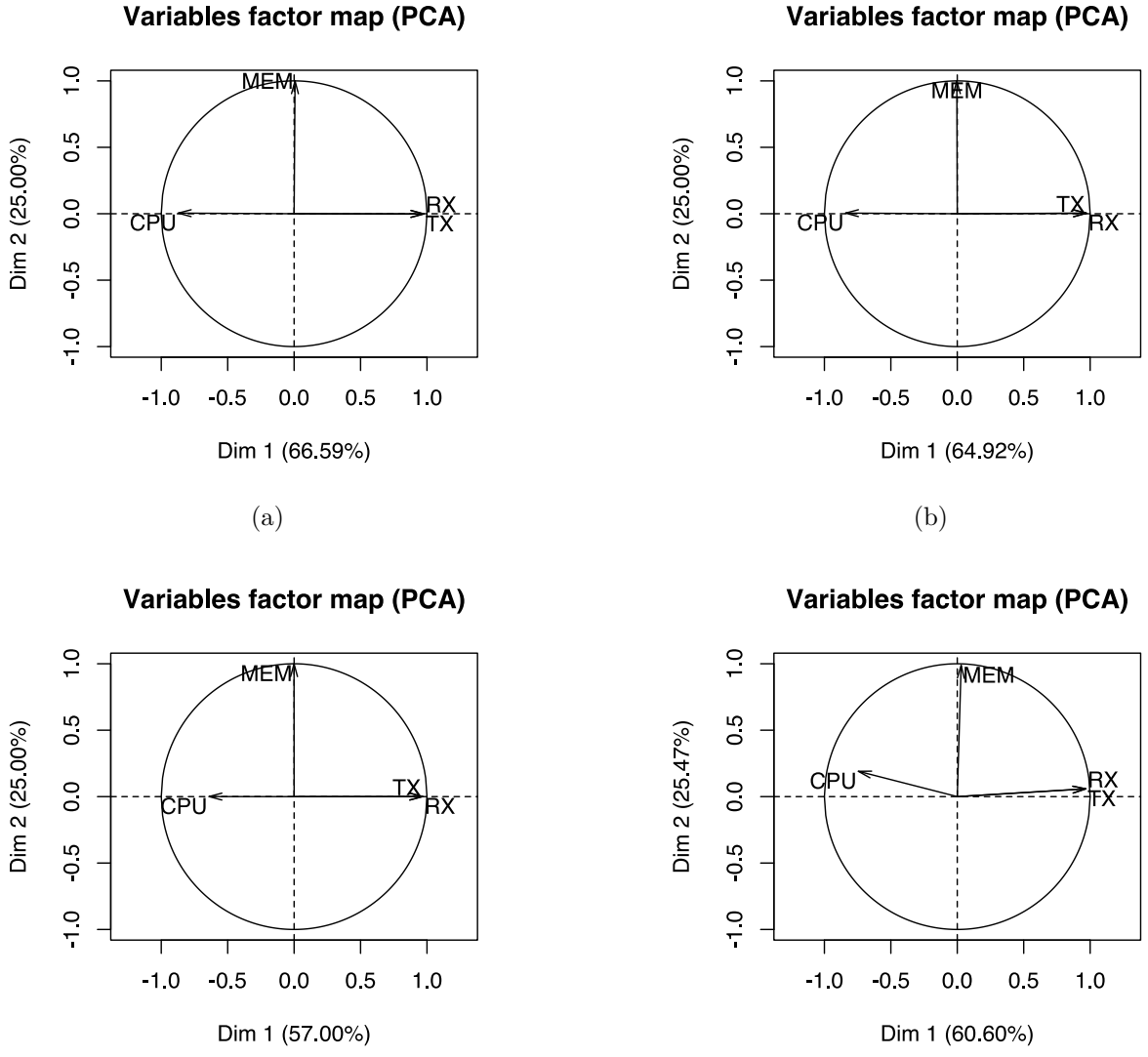
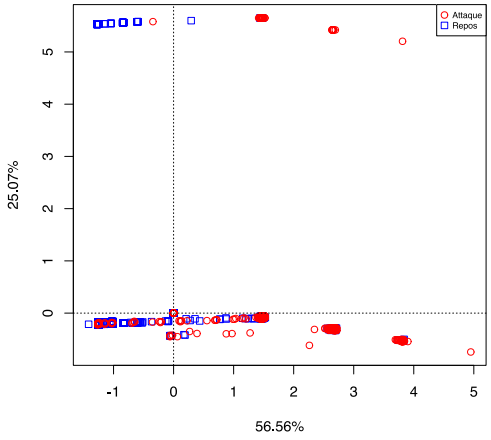
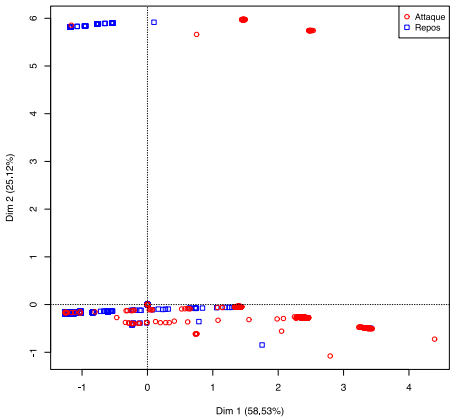


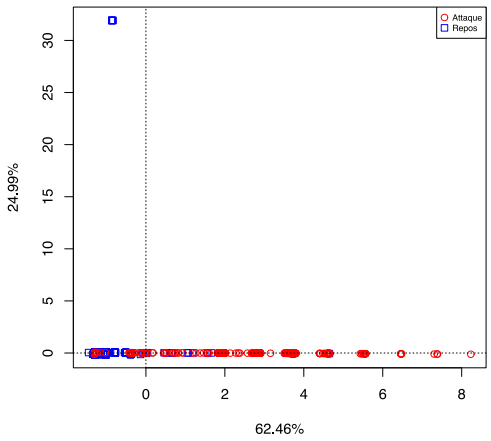
FIGURE A.6 – Cercles de corrélations des ACP réalisées sur le *botnet*Hybrid_V1.0 réalisant des attaques TCP SYN Flood : (a) 159 p/s ; (b) 171 p/s ; (c) 290 p/s ; (d) 420 p/s



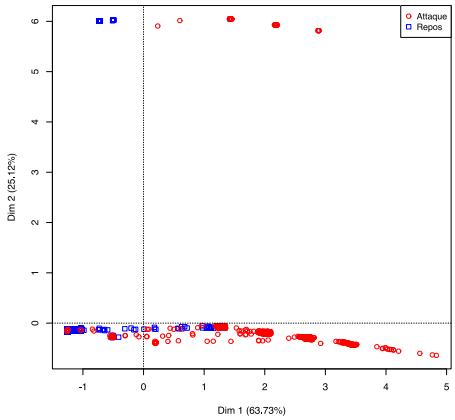
(a)



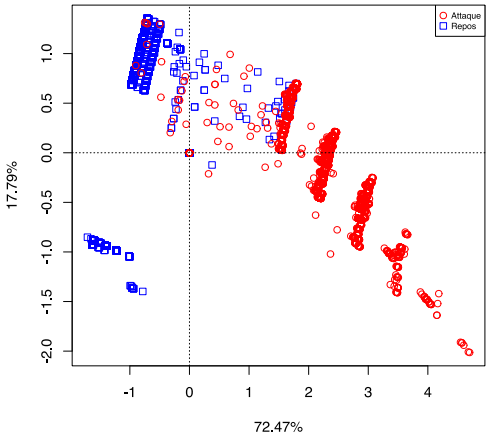
(b)



(c)

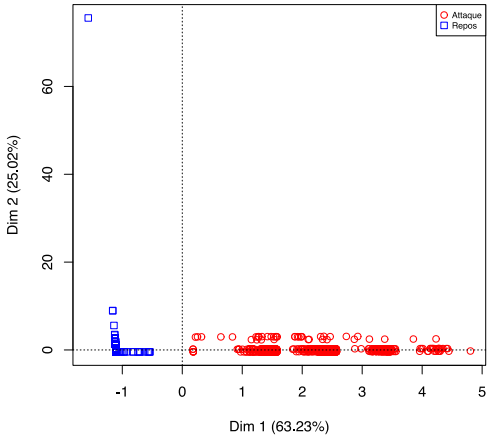


(d)

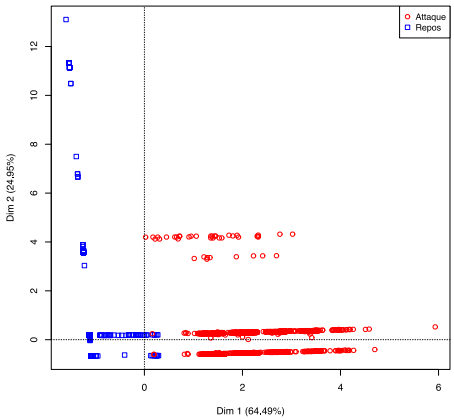


(e)

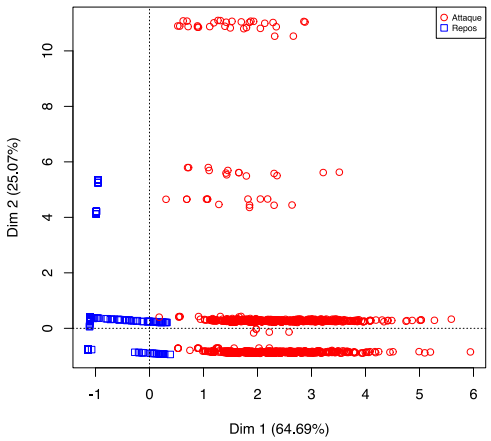
FIGURE A.7 – Projections des individus résultats des ACP réalisées sur le *botnet*Hybrid_V1.0 réalisant des attaques UDP Flood : (a) 8 Mb/s ; (b) 16 Mb/s ; (c) 40 Mb/s ; (d) 56 Mb/s ; (e) 80 Mb/s



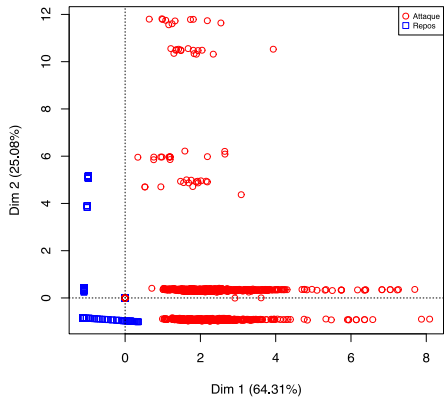
(a)



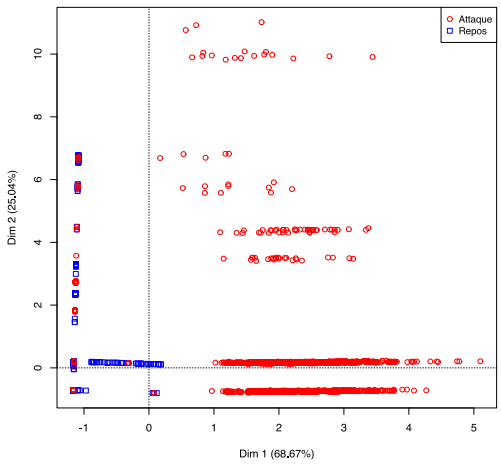
(b)



(c)

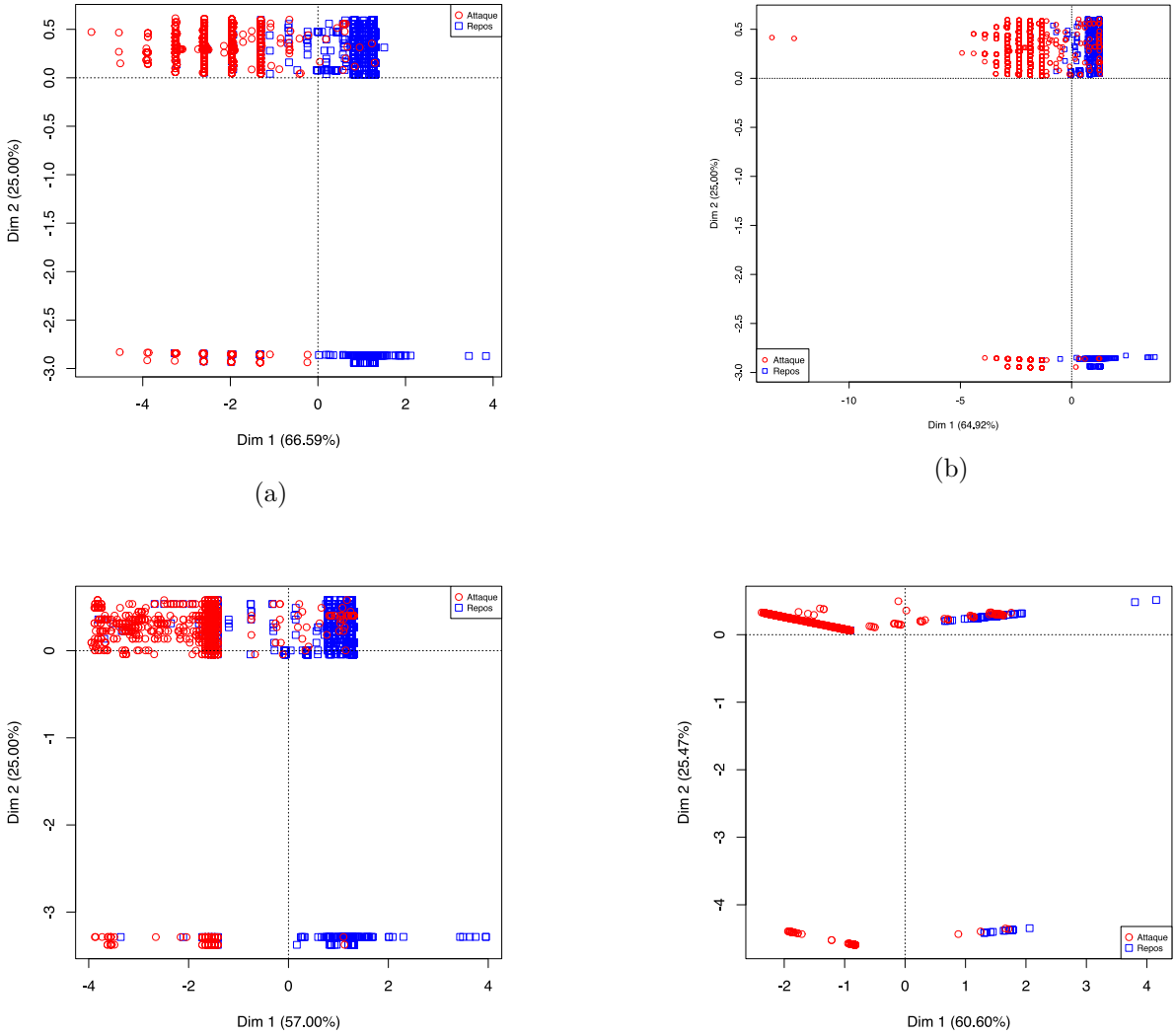


(d)



(e)

FIGURE A.8 – Projections des individus résultats des ACP réalisées sur le *bot-net*Kaiten_V1.0 réalisant des attaques UDP Flood : (a) 8 Mb/s ; (b) 16 Mb/s ; (c) 40 Mb/s ; (d) 56 Mb/s ; (e) 80 Mb/s



(a) (b) (c) (d)

FIGURE A.9 – Projections des individus résultats des ACP réalisées sur le *botnet*Hybrid_V1.0 réalisant des attaques TCP SYN Flood : (a) 159 p/s ; (b) 171 p/s ; (c) 290 p/s ; (d) 420 p/s

Badis HAMMI

Doctorat : Réseaux, Connaissances et Organisations

Année 2015

Vers une détection à la source des activités malveillantes dans les *clouds* publics : application aux attaques de déni de service

Le cloud computing, solution souple et peu couteuse, est aujourd'hui largement adopté pour la production à grande échelle de services IT. Toutefois, des utilisateurs malveillants tirent parti de ces caractéristiques pour bénéficier d'une plate-forme d'attaque prête à l'emploi dotée d'une puissance colossale. Parmi les plus grands bénéficiaires de cette conversion en vecteur d'attaque, les botclouds sont utilisés pour perpétrer des attaques de déni de service distribuées (DDoS) envers tout tiers connecté à Internet.

Si les attaques de ce type, perpétrées par des botnets ont été largement étudiées par le passé, leur mode opératoire et leur contexte de mise en œuvre sont ici différents et nécessitent de nouvelles solutions. Pour ce faire, nous proposons dans le travail de thèse exposé dans ce manuscrit, une approche distribuée pour la détection à la source d'attaques DDoS perpétrées par des machines virtuelles hébergées dans un cloud public.

Nous présentons tout d'abord une étude expérimentale qui a consisté à mettre en œuvre deux botclouds dans un environnement de déploiement quasi-réel hébergeant une charge légitime. L'analyse des données collectées permet de déduire des invariants comportementaux qui forment le socle d'un système de détection à base de signature, fondé sur une analyse en composantes principales. Enfin, pour satisfaire au support du facteur d'échelle, nous proposons une solution de distribution de notre détecteur sur la base d'un réseau de recouvrement pair à pair structuré qui forme une architecture hiérarchique d'agrégation décentralisée.

Mots clés : informatique dans les nuages - réseaux d'ordinateurs, mesures de sûreté - attaques par déni de service - expériences.

Toward a Source Based Detection of Malicious Activities in Public Clouds: Application to Denial of Service Attacks

Cloud computing, flexible and cheap solution, is actually widely adopted for large-scale production of IT services. However, beyond their legitimate usage, malicious users take advantage of these features in order to get a ready-to-use attack platform, equipped with a huge power. Among the greatest beneficiaries of this cloud conversion into an attack support we find botclouds, which are used to perpetrate Distributed Denial of Service (DDoS) attacks toward any third party connected to the Internet.

Even if such attacks when perpetrated by botnets have been extensively studied in the past, their modus operandi and their implementation context are different herein and require new solutions. In order to achieve such a goal, we propose in this thesis a distributed approach for a source- based detection of DDoS attacks perpetrated by virtual machines hosted in a public cloud.

Firstly, we present an experimental study that consists in the implementation of two botclouds in a real deployment environment that hosts a legitimate workload. The analysis of the collected data allows the deduction of behavioural invariants that form the basis of a signature based detection system. Thus, we present in the following a detection system based on the identification of principal components of the deployed botclouds. Finally, in order to address the scaling factor, we propose a distributed solution of our detection system and which relies on a mesh peer-to-peer architecture resulting from the overlap of several trees of overlay networks.

Keywords: cloud computing - computer networks, security measures - denial of service attacks - experimental approach.

Thèse réalisée en partenariat entre :

RÉGION CHAMPAGNE ARDENNE

utt université de technologie Troyes

Ecole Doctorale "Sciences et Technologies"