UNIVERSITÉ DE MONTRÉAL

# DÉNI-DE-SERVICE : IMPLÉMENTATION D'ATTAQUES XML-DOS ET ÉVALUATION DES DÉFENSES DANS L'INFONUAGIQUE

ADRIEN BONGUET
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2015

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

DÉNI-DE-SERVICE : IMPLÉMENTATION D'ATTAQUES XML-DOS ET ÉVALUATION DES DÉFENSES DANS L'INFONUAGIQUE

présenté par : BONGUET Adrien
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

Mme BOUCHENEB Hanifa, Doctorat, présidente
Mme BELLAÏCHE Martine, Ph. D., membre et directrice de recherche
M. KHOMH Foutse, Ph. D., membre

## DÉDICACE

*A celle qui a cru en moi et m'a soutenu durant ces années,*
*dans les bons comme les moins bons moments...*
*Merci Sandie*

# REMERCIEMENTS

Mes premiers remerciements vont à Mme Martine Bellaïche, qui a rendu ces deux années possibles, qui a toujours su me guider quand je me perdais, m'a toujours prodigué de bons conseils, et avec qui j'ai finalement eu grand plaisir à travailler.

Merci à mes collègues de bureau avec qui il a toujours été facile d'échanger et de partager des idées, et qui m'ont parfois donné un recul qui me manquait sur certaines problématiques.

Merci aux chercheurs extérieurs à l'école que j'ai contactés et qui ont accepté de répondre à mes questions et mes interrogations. Notamment Thomas Vissers pour ses précisions sur son système de défense, et également Mateusz Guzek, un des principaux développeurs du simulateur qui m'a servi de base de travail, pour m'en avoir expliqué certaines subtilités.

Merci aux nombreux professeurs de génie informatique que j'ai eu la chance de côtoyer durant mes années à Polytechnique, j'ai appris énormément à votre contact. Un merci tout particulier à François-Raymond Boyer et à nouveau à Martine Bellaïche pour m'avoir fait confiance en m'accordant le temps de cinq sessions une charge de laboratoire. Cette expérience m'a appris énormément, tant sur le plan technique qu'humain. Merci donc à vous deux.

Merci aux membres du jury pour le temps accordé à la lecture de ce mémoire.

Merci enfin aux personnes qui m'ont apporté le soutien moral sans lequel tout aurait été bien plus difficile, et qui ont toujours su me motiver et m'aider à tirer le meilleur parti de mes capacités. Je ne remercierai jamais assez celle qui a été à mes côtés durant tout ce temps, merci Sandie, si ce travail a pu voir le jour c'est en grande partie grâce à toi.

Merci à mes parents et à ma famille. Merci pour vos conseils et pour votre soutien.

Merci à tous ceux encore que j'aurais oublié et qui ont contribué, directement ou indirectement, à ma réussite.

# RÉSUMÉ

L'infonuagique est un paradigme informatique dont la popularité et les usages n'ont fait que croître ces dernières années. Il doit son succès à sa grande adaptabilité et capacité de mise à l'échelle, ainsi qu'à sa facilité d'utilisation. Son but est de permettre à l'individu ou à l'entreprise d'utiliser des ressources informatiques qu'il ou elle ne possède pas physiquement, en utilisant des techniques déjà connues et éprouvées comme la virtualisation et les services web. Plusieurs modèles d'infonuagique existent, selon la fonctionnalité recherchée. Par exemple, déposer, partager et accéder depuis n'importe quel terminal un fichier sur Dropbox, et exécuter une application extrêmement gourmande en ressources sur une machine louée le temps de l'exécution sont deux exemples d'utilisation de l'infonuagique. Le modèle d'infonuagique influe sur la portion des ressources gérées par l'utilisateur, en comparaison des ressources gérées par le fournisseur. Dans le cas de Dropbox, l'utilisateur n'a pas besoin de se soucier des ressources allouées à sa requête ni de savoir quel système d'exploitation a servi ou encore comment est gérée la base de données, alors que dans l'autre cas tous ces paramètres rentreront probablement en compte et seront donc à la charge de l'utilisateur. Un réseau d'infonuagique peut tout autant être un réseau public accessible de tous moyennant finances, qu'un réseau privé bâti et utilisé seulement par une entreprise pour ses propres besoins. L'attrait considérable de l'infonuagique, pour les particuliers comme pour les entreprises, augmente par le fait même les risques liés à la sécurité de ces réseaux, ceux-ci devenant une cible de choix pour les attaquants. Ce risque accru allié à la confiance que l'utilisateur doit porter au fournisseur pour gérer et protéger ses données explique que nombreux sont ceux encore réticents à utiliser l'infonuagique, que ce soit pour des questions de confidentialité pour une entreprise ou de vie privée pour un particulier. La diversité des technologies utilisées implique une grande variété d'attaques possibles. Notamment, les réseaux d'infonuagique pâtissent des mêmes vulnérabilités que les réseaux conventionnels, mais également des failles de sécurité liées à l'utilisation de machines virtuelles. Cependant, ces menaces sont dans l'ensemble bien connues et la plupart du temps des mesures sont mises en place pour les contrer. Ce n'est pas le cas des vulnérabilités liées à l'utilisation des services web, utilisés abondamment dans le cas de l'infonuagique.

Les réseaux d'infonuagique se donnent pour but d'être accessibles depuis n'importe où et n'importe quel appareil, ce qui passe nécessairement par l'utilisation de services web. Or les serveurs web sont extrêmement vulnérables aux attaques de type XML-DoS, mettant à profit des requêtes SOAP (Simple Object Access Protocol) utilisant un message XML malveillant.

Ces attaques ont pour but d'utiliser une très grande partie si ce n'est pas toutes les ressources CPU et mémoire de la machine hébergeant le serveur web victime, la rendant indisponible pour des utilisateurs légitimes, ce qui est le but recherché lors d'une attaque de déni de service. Ces attaques sont extrêmement intéressantes d'un double point de vue. Elles sont tout d'abord très difficiles à détecter, car l'utilisateur qui en est à l'origine est perçu comme un utilisateur légitime (attaque au niveau de la couche application, donc impossible de la détecter au niveau de la couche TCP/IP). De plus, elles présentent une dissymétrie importante entre les ressources dont l'attaquant a besoin pour monter l'attaque, et les ressources nécessaires pour traiter la requête. En effet, une requête SOAP mal formée bien que très basique peut déjà demander des ressources considérables au serveur web victime. Ce type d'attaques ayant été assez peu étudié, malgré son efficacité et l'omniprésence des services web dans l'infonuagique, nous nous sommes proposé de démontrer et de quantifier l'impact que peuvent avoir ces attaques dans le cadre de l'infonuagique, pour ensuite proposer des solutions possibles pour s'en défendre. Nous avons jugé qu'il était plus approprié de recourir à des outils de simulation pour mener nos travaux, pour plusieurs raisons comme notamment la possibilité de suivre de façon précise l'évolution des ressources des différents serveurs du réseau, et la plus grande liberté qui nous était laissée de construire notre propre topologie.

Notre première contribution est de mettre en avant les équipements vulnérables dans un réseau d'infonuagique, et les différentes façons de les attaquer, ainsi que les différents types d'attaques XML-DoS. Cette analyse nous a permis d'apporter notre deuxième contribution, qui consiste à utiliser et à modifier en profondeur un simulateur d'infonuagique (le simulateur GreenCloud, basé sur le simulateur NS2) afin de le rendre apte à l'étude des attaques XML-DoS et plus réaliste. Une fois ces changements effectués, nous montrons l'efficacité des attaques XML-DoS et les répercussions sur les usagers légitimes. Par ailleurs, nous réalisons une comparaison critique des principales défenses contre les attaques XML-DoS et contre les services web en général, et sélectionnons celle qui nous semble la plus pertinente, afin de la mettre à l'épreuve de la simulation et de mesurer son efficacité. Cette efficacité doit être démontrée aussi bien en terme de capacité à déjouer l'attaque menée à l'étape précédente, que de précision en terme de "false positives" et "false negatives". Un des défis majeurs consiste en effet à concilier une défense qui se veut universelle pour toutes les machines du réseau, tout en étant capable de s'adapter à la grande hétérogénéité des services web qui cohabitent au sein d'un réseau d'infonuagique. Ces expérimentations sont alors l'objet de discussions et de conclusions sur la position à adopter quant aux attaques XML-DoS dans les réseaux d'infonuagique, notamment les défenses à adopter et les pratiques à observer, la défense choisie à l'étape précédente ayant montré quelques limitations. Nous sommes par-

tis de l'hypothèse que chacun des modèles d'infonuagique pouvait être touché par ce type d'attaques, bien que de façons différentes. Quel que soit le modèle d'infonuagique, il peut en effet avoir recours à des services web et se retrouve donc vulnérable d'une façon ou d'une autre, qu'il s'agisse d'un serveur web gérant toutes les requêtes entrantes des utilisateurs, ou un serveur web qu'un utilisateur a lui-même installé sur une machine virtuelle qu'il loue au fournisseur. Nous avons aussi jugé essentiel d'introduire les spécificités liées à l'utilisation de machines virtuelles, comme la compétition pour les ressources entre machines virtuelles situées sur une même machine physique.

# ABSTRACT

Cloud Computing is a computing paradigm that has emerged in the past few years as a very promising way of using highly scalable and adaptable computing resources, as well as accessing them from anywhere in the world and from any terminal (mobile phone, tablet, laptop...). It allows companies or individuals to use computing infrastructures without having to physically own them, and without the burden of maintenance, installations, or updates. To achieve that, Cloud Computing uses already known and tested technologies such as virtualization and web services. Several Cloud Computing models exist, divided by how much of the infrastructure the user is in charge of, ranking from nothing at all (the provider is in charge of everything, from the operating system to the applications installed) to managing a whole virtual machine without even an operating system preinstalled. For instance, sharing and accessing a document on Dropbox, or running a resource intensive application on a rented machine, are both exemples of what can be done with Cloud Computing. In the case of Dropbox, the user doesn't care what resources are allocated for his requests, no more than he needs to know on what operating system the request was run or how the database was accessed. But all those aspects will be part of what the user has to know and adjust in the second case. A Cloud Computing network can be public, as it is the case for Amazon, which allows you to access its resources if you pay for it, or private, if a company decides to build a cluster for its own needs. The strong appeal of Cloud Computing, for both businesses and individuals, dramatically increases the security risks, because it becomes a key target for attackers. This increased risk, added to the confidence the users must have in their services provider when it comes to managing and protecting their data, may explain why many are still reluctant to take the leap to Cloud Computing. For instance, a company may be reluctant for confidentiality reasons, while individuals may hesitate for privacy concerns. The broad range of technologies used in Cloud Computing makes it at risk for a wide variety of attacks, since it already comes with all the vulnerabilities associated with any conventional network, and all the security breaches that affect virtual machines. However, those threats are usually well documented and easily prevented. But this is not the case of the vulnerabilities that come from the use of web services, that are heavily used in Cloud Computing.

Cloud Computing networks aim at being accessible from all over the world and on almost any device, and that implies using web services. Yet, web services are extremely vulnerable to XML-DoS type of attack. Those attacks take advantage of Simple Object Access Protocol (SOAP) requests using a malicious XML content. Those requests can easily deplete a web

server from its resources, be it CPU or memory, making it unavailable for legitimate users; this is exactly the goal of a denial-of-service attack. The XML-DoS attacks are extremely interesting in two ways. First, they are very hard to detect, since the attack takes place on the application layer, so the user appears to be legitimate (it's impossible to detect it on the TCP/IP layer). Second, the resources needed to mount the attack are very low compared to what is needed on the web server side to process even a basic but malformed request. This type of attack has been surprisingly left quite aside, despite its efficiency and the omnipresence of web services in Cloud Computing networks. This is the reason why we decided to prove and quantify the impact such attacks can have on Cloud Computing networks, to later propose possible solutions and defenses. We estimated that using a simulated environment was the best option for various reasons, like the possibility to monitor the resources of all the servers in the network, and the greater freedom we had to build our own topology.

Our first contribution is to emphasize the vulnerable equipments in a Cloud Computing network, and the various ways to attack them, as well as the various forms an XML-DoS attack can take. Our second contribution is then to use, modify and improve a Cloud Computing simulator (the GreenCloud simulator, based on NS2), in order to make the study of XML-DoS attacks possible. Once the changes are made, we show the efficiency of XML-DoS attacks, and the impact they have on legitimate users. In addition, we compare the main existing defenses against XML-DoS attacks and web services attacks in general, and pick the one that seems to be best suited to protect Cloud Computing networks. We then put this defense to the test in our simulator, to evaluate its efficiency. This evaluation must take into consideration not only the ability to mitigate the attack we led in the previous step, but also the number of false positives and false negatives. One of the major challenges is to have a defense that will conciliate the ability to protect all the machines in the network, while still being able to adapt to the great heterogeneity of the various web services hosted at the same time in a Cloud Computing network. Those experimentations are then subjected to discussions and conclusions on the decisions to take when it comes to XML-DoS attacks. In particular, what defenses should be adopted and what practices should be followed, because the evaluation of the defense at the previous step will show that it may not be the optimal solution; this will be our final contribution. We made the assumption that all the Cloud Computing models could be the target of a XML-DoS attack in some way. No matter the model, it can actually use web services and is then vulnerable to those attacks, whether it is a web server handling incoming requests for all the users, or a web server a user installed on the virtual machine he rents. We thought it was essential to take into consideration the specificity of virtual machines, such as the contention for resources when they are located on

the same physical machine.

# TABLE DES MATIÈRES

# LISTE DES TABLEAUX

# LISTE DES FIGURES

# LISTE DES SIGLES ET ABRÉVIATIONS

| | |
|---|---|
| VM | machine virtuelle |
| SaaS | Software as a Service |
| PaaS | Platform as a Service |
| IaaS | Infrastructure as a Service |
| DoS | Denial-of-Service |
| SOA | Service Oriented Architecture |
| SOAP | Simple Object Access Protocol |

# CHAPITRE 1 INTRODUCTION

Le "Cloud Computing" (ou infonuagique, mot qui sera utilisé par la suite) permet d'accéder à des ressources informatiques hautement configurables (réseaux, serveurs, stockage, applications et services) sans avoir à posséder ni à maintenir ces ressources. Un réseau d'infonuagique est caractérisé par sa grande échelle, puisqu'un tel réseau peut être constitué de plusieurs milliers de serveurs, la mise en commun des ressources, un accès depuis n'importe où et n'importe quel appareil, une grande élasticité (réponse rapide et dynamique aux demandes des utilisateurs), et un service à la demande (on paie pour ce qu'on utilise). Par exemple, une petite entreprise peut ne pas avoir le besoin ou les ressources financières d'acheter de nombreuses ressources informatiques dès le début, mais voudra sûrement s'agrandir avec le temps. Ce cas fournit un bon exemple du genre de situations dans lequel l'infonuagique est particulièrement approprié. Mais en dépit de nombreux avantages, et même si cette technologie commence à se faire une place de choix dans le monde numérique, beaucoup d'entreprises et de particuliers restent réticents à migrer vers des infrastructures d'infonuagique, principalement pour des raisons de sécurité. Dans ce domaine, les attaques de type Denial-of-Service (DoS) représentent une des plus grandes menaces.

## 1.1 Définitions et concepts de base

Il y a trois façons d'utiliser les ressources offertes par les réseaux d'infonuagique, correspondants aux trois couches suivantes : "Software as a Service (SaaS)", "Platform as a Service (PaaS)" et "Infrastructure as a Service (IaaS)". Chacune de ces couches (qu'on appelle également des modèles) vient avec ses propres failles de sécurité. La couche SaaS permet de délivrer des applications gérées par un tiers et dont l'interface est accédée côté client, un navigateur web permettant alors de faire tourner l'application, le tout sans avoir eu à se soucier de l'installation ni de la gestion de cette application. La couche PaaS met à disposition de l'utilisateur un système d'exploitation, un environnement de programmation ou encore un serveur web. Enfin, la couche IaaS représente le plus bas niveau d'abstraction. L'utilisateur dispose alors uniquement d'une infrastructure physique ou plus souvent virtuelle (au travers d'une machine virtuelle (VM)). Un résumé des ressources qui sont à la charge du fournisseur en fonction du modèle d'infonuagique est présenté figure 1.1.

La couche SaaS est celle qui semble rencontrer le plus de succès auprès des consommateurs de par sa facilité d'utilisation et sa grande flexibilité, grâce au recours à une architecture appelée "Service Oriented Architecture (SOA)" et aux services web. SOA est une architecture

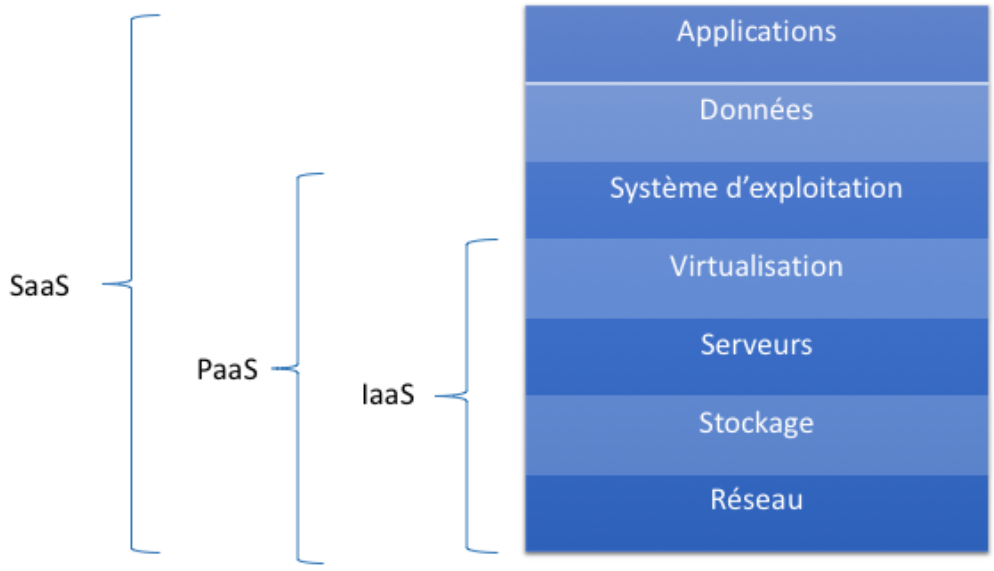


Figure 1.1 Ce qui est à la charge du fournisseur de services en fonction du modèle d'infonuagique

permettant de bâtir des applications au travers, par exemple, de services web, et grâce à laquelle des systèmes hétérogènes peuvent interagir entre eux (ce qui explique la propriété d'ubiquité des réseaux d'infonuagique). Cependant, l'utilisation de services web se heurte à de nombreux problèmes de sécurité, notamment les attaques de déni de service faisant un usage malveillant des langages HTTP et XML, utilisés abondamment dans le cadre des services web. Ces attaques prenant place sur la couche application du modèle OSI (couche 7), ces attaques ne sont généralement pas arrêtées par les pare-feux. Notre objectif dans ce projet est ainsi d'explorer les attaques de déni de services appliquées aux réseaux d'infonuagique et exploitant ces vulnérabilités. Nous savons déjà que ces attaques ont pour but d'empêcher l'utilisateur légitime d'utiliser les services offerts par un fournisseur. Il s'agira d'aller plus loin en précisant ce qui définit une telle attaque, comment en créer une, et surtout, comment faire pour s'en défendre. Par ailleurs, étudier l'impact d'un déni de service sur un réseau d'infonuagique ne revient pas au même que de l'étudier sur un réseau classique, ceci notamment à cause de l'utilisation de machines virtuelles.

### 1.1.1 Les machines virtuelles dans les réseaux d'infonuagique : bénéfices et limitations

La virtualisation est devenue une technologie essentielle des centres de données. En faisant tourner plusieurs machines virtuelles dans une même machine physique partagée, elle permet une meilleure utilisation des ressources matérielles. Mais malgré des avantages comme l'isolation entre les machines virtuelles en terme de sécurité (pas d'attaques d'un domaine à partir d'un autre domaine), de défauts et fautes (une application qui se comporte mal ne va pas affecter tout le système), et en terme d'environnement (plusieurs systèmes d'exploitation (OS) peuvent tourner sur la même machine, avec des configurations et des paramètres différents), il n'y a par exemple pas encore d'isolation au niveau des performances entre les machines virtuelles. Notamment, la compétition pour les ressources physiques peut avoir des impacts différents suivant la charge de travail, impliquant une variation significative du flux de production. Dans la plupart des cas, un environnement virtualisé utilise un "hypervisor", qui est en fait une couche logicielle faisant le lien entre les machines virtuelles et le matériel physique. L'"hypervisor" alloue les ressources aux machines virtuelles, et leur donne l'impression qu'elles possèdent leurs propres mémoire, CPU, stockage, etc. Mais même avec cet « hypervisor », qui se charge du découpage et de la répartition des ressources aux différentes machines virtuelles, le comportement d'une machine virtuelle peut toujours affecter la performance d'une autre à cause du partage des ressources du système. Par ailleurs, l'"hypervisor" a très peu de visibilité quant à l'OS invité et à ses applications, et mesurer l'utilisation des ressources sur des serveurs virtuels est difficile, car les "performance counters" pour comptabiliser les

ressources ne sont pas virtualisés.

La virtualisation implique aussi que l'OS invité et les applications qui tournent dans une machine virtuelle ne peuvent pas être pleinement informés de ce qu'il se passe dans les autres machines virtuelles, et ne peuvent pas déterminer les effets des interférences au niveau des performances. Des outils existent néanmoins, mais ils ne permettent pour la plupart que la connaissance de l'hôte qui sature ou a des ressources inactives, mais pas de quelle façon les machines virtuelles utilisent ces ressources [30].

### 1.1.2 Les attaques de déni de service dans le cadre de l'infonuagique : l'exemple du XML-DoS et du HTTP-DoS

Les attaques DoS ont pour particularité d'être très répandues, relativement faciles à mettre en œuvre, mais extrêmement difficiles à éviter et contrôler. Si la sécurité des réseaux d'infonuagique est de plus en plus étudiée, peu de recherches portent à ce jour sur l'étude des attaques de déni de service appliquées aux réseaux d'infonuagique. De plus, le problème des attaques de déni de service est loin d'avoir été résolu malgré leur existence dans le paysage informatique depuis de nombreuses années pour ne pas dire dizaines d'années, et une part de plus en plus grande dans l'ensemble des attaques menées de nos jours. Par ailleurs, certaines attaques de type DoS ont déjà été étudiées de façon approfondie par les chercheurs du monde entier (c'est le cas par exemple des attaques par "flooding" qui cherchent à inonder un réseau de requêtes), mais les attaques XML-DoS n'ont quant à elles pas été réellement considérées. Cependant, le langage XML, qui est le support de cette attaque, est devenu la brique fondamentale d'internet de nos jours, tous les services web se basant sur lui. Ainsi, détecter, filtrer et analyser tous les messages XML est difficilement envisageable, car cela complexifierait énormément les communications, et rien ne garantit que toutes les attaques seraient détectées ni même que l'expérience utilisateur n'en serait pas affectée. Parce qu'ils veulent se rendre disponibles partout et depuis n'importe quel appareil, les réseaux d'infonuagique utilisent massivement les services web qui permettent cette interopérabilité entre systèmes hétérogènes, et sont donc de potentielles victimes de ces attaques XML-DoS. Certaines approches et architectures ont été proposées pour combler le trou de sécurité existant sur ces attaques, et nous ont inspiré dans notre travail.

## 1.2 Éléments de la problématique

## 1.3 Objectifs de recherche

Notre objectif principal dans ce projet est de simuler des attaques XML-DoS, ainsi que les défenses correspondantes, et d'évaluer la performance de ces défenses dans le cadre de l'infonuagique. Les 4 sous-objectifs présentés ci-dessous permettent de répondre à cet objectif principal.

1. Analyser la vulnérabilité des réseaux d'infonuagique aux attaques XML-DoS

   — Étude des équipements vulnérables
   — Possibilités offertes pour attaquer ces équipements. Manière d'accéder aux services offerts par un réseau d'infonuagique et vulnérabilités qui en découlent
   — Efficacité des attaques (relativement aux objectifs d'un DoS : rendre le service inutilisable pour les autres utilisateurs)
   — Différence entre les réseaux d'infonuagique et les autres réseaux face aux attaques XML-DoS

2. Utiliser et améliorer un environnement simulé pour décrire les attaques XML-DoS dans l'infonuagique

   — Choisir le simulateur le plus adapté
   — En comprendre le fonctionnement et identifier les modifications à apporter pour le rendre apte à l'étude des attaques XML-DoS
   — Effectuer les modifications en respectant l'architecture du simulateur

3. Comparer les défenses existantes contre les attaques XML-DoS

   — Étude des défenses classiques contre les vulnérabilités des services web. Lister et analyser les défenses existantes, et discuter de leur utilisation dans la pratique
   — Implémenter les défenses pertinentes dans le simulateur

4. Analyser les défenses grâce à leur implémentation dans le simulateur

   — Étude de l'efficacité des défenses

— Analyse de la nécessité ou non de défenses plus complexes que celles présentées ici

Voici la méthodologie retenue pour atteindre ces objectifs. Tout d'abord, effectuer une revue de littérature sur l'état de l'art des attaques de déni de service et en particulier les attaques XML-DoS et des défenses dans l'infonuagique. Ensuite, chercher à déterminer dans quelle mesure on peut étendre les possibilités offertes par le simulateur GreenCloud [15], afin de mettre en place une attaque XML-DoS et d'en mesurer l'impact, pour ensuite implémenter une ou plusieurs contremesure(s). Cette analyse du simulateur doit aussi conduire à la façon dont le simulateur permet le suivi du CPU et de la mémoire (les principaux paramètres d'intérêt dans le cadre de ce projet) des équipements du réseau, et à la création de nouveaux outils de suivi si ceux existants ne suffisent pas ou sont incomplets. Ensuite, déterminer comment rendre compte, de façon simple, mais en restant suffisamment proche de la réalité, du traitement effectué par un service web dans le simulateur, cette façon de traiter le message "Simple Object Access Protocol (SOAP)" qui, rappelons-le, est à la base de l'attaque. Enfin, trouver une façon de simuler plusieurs des attaques XML-DoS qui représentent toujours une menace contre lesquelles aucune contremesure efficace n'est actuellement et systématiquement mise en oeuvre, ainsi que les défenses pertinentes. Il faut aussi rendre compte de l'impact d'une attaque XML-DoS au travers de nouveaux indicateurs, notamment l'interférence de performance qu'il peut y avoir entre machines virtuelles situées sur une même machine physique. Nous finirons par dresser un bilan des attaques les plus critiques ainsi que des défenses les plus pertinentes pour ce type d'attaque.

### 1.4 Plan du mémoire

Ce mémoire est constitué de six chapitres. Le chapitre 1 introduit le concept d'infonuagique, en expliquant les particularités, les possibilités que ce paradigme offre, mais également ce qui le rend vulnérable. Dans ce chapitre seront également discutées les attaques de déni de service sur la couche applicative, comme les HTTP-DoS et les XML-DoS, et en quoi les réseaux d'infonuagique y sont particulièrement vulnérables. Le chapitre 2 est un article sur l'état de l'art des attaques de déni de services visant les réseaux d'infonuagique. Nous y décrivons les différents scénarios d'attaques et d'attaquant, ainsi que les différentes façons de parvenir à un déni de service. Plusieurs systèmes de défense sont ensuite présentés, accompagnés d'une discussion sur leur évaluation. Le chapitre 3 se concentre sur les attaques XML-DoS appliquées aux réseaux d'infonuagique. Après une mise en évidence des vulnérabilités de tels réseaux face aux attaques XML-DoS, nous justifions ce qui fait l'intérêt d'utiliser un simulateur pour conduire nos expériences, et nous offrons un aperçu détaillé du simulateur

choisi, GreenCloud. Après de brefs rappels concernant le langage XML, nous proposons plusieurs attaques possibles. Enfin, nous expliquons les changements apportés au simulateur pour le rendre apte à l'étude des attaques XML-DoS. Le chapitre 4 permet d'expliquer et de justifier le choix de défenses que nous avons fait, ainsi que d'expliquer quelles métriques ont été évaluées. Le chapitre 5 présente les résultats obtenus grâce au simulateur. Notamment, nous présentons l'impact d'une attaque XML-DoS pour un réseau d'infonuagique, et comment certaines défenses permettent de prévenir cette menace. Ces défenses sont ensuite évaluées pour mesurer leur efficacité. Enfin, le chapitre 7 apporte des conclusions, et permet de dresser un bilan des difficultés rencontrées et des améliorations qui pourraient être apportées.

# CHAPITRE 2 ARTICLE 1 : A SURVEY OF DENIAL-OF-SERVICE ATTACKS AND DEFENSES TO CLOUD COMPUTING AVAILABILITY

## Authors

Adrien Bonguet
Génie Informatique et Génie Logiciel
École Polytechnique de Montréal
Montréal, QC, CANADA

Martine Bellaïche
Génie Informatique et Génie Logiciel
École Polytechnique de Montréal
Montréal, QC, CANADA

Paper submitted to IEEE Cloud Computing.

### 2.1 Abstract

Cloud Computing refers to a model that allows ubiquitous, convenient and on-demand access to a shared pool of highly configurable computing resources (networks, servers, storage, applications and services). This paper focuses specifically on new Denial-of-Service (DoS) attacks on Cloud Computing, which harm availability. Our goal is to explore exactly what a DoS attack is, and to examine all the types of DoS attacks, with the practical examples of XML-DoS and HTTP-DoS, to subsequently study the possible detections and mitigations. Our contribution to this survey focuses on DoS attacks and defenses applied to Cloud Computing availability. Our survey will demonstrate that DoS attacks (specifically XML-DoS and HTTP-DoS) present a serious threat to Cloud Computing, with many vulnerabilities, originating from various types of attacks and attackers, that have yet to be solved. However, there also are new and original solutions emerging to embrace the security issues of this very powerful and promising new paradigm of Cloud Computing. Furthermore, this paper presents the design experiment and the metrics used to evaluate DoS defenses.

***Index terms***— Cloud Computing, Denial-of-Service, Security, Countermeasures.

## 2.2 Introduction

Cloud Computing includes both what is delivered as a service over the internet, and the hardware behind those services. Resources can be provisioned and released very easily, requiring little if any intervention from the provider. From the user's point of view, Cloud infrastructures seem to provide infinite resources, which can be adjusted to one's needs. For example, a small start-up company may lack the need or the financial resources required to buy many computing resources, but may want to leave its options open for a future expansion, if successful, making Cloud Computing particularly appropriate in such case. The company simply pays for what is actually used given that resources can be released when they are no longer required.

In a Cloud network users do not own the computing servers. They can access numerous services without the burden of Cloud management and their data can be accessed by way of many devices (such as smart phones, sensors, tablets, etc.).

More generally, the main features of Cloud Computing are [57] :

1. *Large-Scale* : to satisfy the customers' demands, companies like Amazon, IBM, Microsoft, Yahoo, Google own hundreds of thousands of distributed servers.
2. *Resource pooling* : providers serve multiple customers with provisional and scalable services. These services can be transparently adjusted to the clients' needs [22].
3. *Ubiquitous network access* : users can access services anywhere, through any kind of terminal.
4. *Rapid elasticity* : users can increase and release their demand quickly and dynamically.
5. *On-demand self-service* : since a Cloud infrastructure is a large pool of resources, users can buy according to their needs. The provider automatically provides services and associates resources to the Cloud user, as requested.
6. *High extensibility* : the scale of a Cloud infrastructure can extend to meet the increasing requirements of customers.

Security in Cloud Computing is critical when developing services. Updating the operating systems of virtual machines, ensuring availability, data isolation between users, implementing authentication mechanisms, encryption or configuring VPN and VLAN, are but a few examples of what needs to be considered [47]. Below is a list of the security aspects that challenge Cloud Computing.

1. *Identity, Authentication, Authorization, Auditing*
   Identity enables to characterize a user through the use of a login. Authentication is

used to verify the user's credentials. This is done in a secure, trustworthy, and manageable manner [3]. When authentication is complete, the Cloud authorization verifies what the user is allowed to do. Guidance includes a centralized directory, identity management, privileged user and access management, role-based access control and separation of duties among main features.

Any credit card holder can quickly use Cloud Computing services. Also, the service provider can frequently offer a free trial period. For example, in the summer of 2012, attackers accessed Mat Hona's data (writer for Wired Magazine) Apple, Gmail and Twitter accounts [35]. They erased all his personal data in those accounts.

2. *Confidentiality*
   A malicious attacker in a virtual machine can listen to another virtual machine [42]. An attacker can very easily identify the data center of the Virtual Machine (VM), and can also obtain information about the IP address and the domain name of the data center. In addition, a VM can extract private cryptographic keys being used in other VMs on the same physical server, which subsequently implies the risk of data leakage [35]. Thus it is important to protect the confidentiality of VM data.
3. *Integrity*
   Phishing, fraud and exploitation of software vulnerabilities, traffic hijacking can eavesdrop activities and transactions, manipulate data, return falsified information and redirect clients to illegitimate sites. Similarly, weak interfaces and APIs cannot protect users from accidental or malicious attempts [35].
4. *Isolation*
   Cloud Computing must have a level of isolation among all the VM data and the hypervisor [19] [13]. In IaaS, it means isolating VMs' storage, processing memory and access path networks. In PaaS : running services and API calls must be isolated. And in SaaS : isolation amongst transactions.
5. *Availability*
   Illegitimate users consume much of the victim's processing power, memory, disk space or network bandwidth. It also causes system slowdowns, which prevents legitimate users from using the service. Consequently, the VM becomes unavailable, causing a DoS.

Our contributions to this survey are to present :

— DoS attacks targeting Cloud availability,
— a description of the XML-DoS and HTTP-DoS attacks,
— the defenses applied to DoS attacks in Cloud Computing,

— the specific defenses against XML-DoS and HTTP-DoS attacks,
— a summary of the means to evaluate such defenses.

So far, we have presented features of Cloud Computing and certain security aspects. The remainder of this paper is organized as follows : Section 2.3 describes the work related to the classification of security issues, Section 2.4 identifies the possible attacks and attackers, Section 2.5 focuses more specifically on DoS applied to Cloud Computing and Section 2.6 examines possible detections and mitigations of DoS attacks. Furthermore, Section 2.7 identifies the experimentation and the metrics to evaluate the defenses and Section 2.8 offers a conclusion.

## 2.3 Related Work

The new paradigm introduced by Cloud Computing creates new security challenges, and therefore, a number of scientific contributions were made in this field during the past few years. Much work has been done to identify threats and vulnerabilities and new frameworks and strategies were created to address such problems. Furthermore, these security concern are likely to increase in the coming years due to the progressive migration of companies and individuals to Cloud infrastructures. Following is a review of some of the Cloud security surveys that were recently published.

Grobauer et al. [16] expose vulnerabilities associated to Cloud Computing. For example, the vulnerabilities are (1) VM escape, (2) session riding and hijacking, (3) insecure or obsolete cryptography, (4) unauthorized access to management interface, (5) Internet protocol vulnerabilities and (6) data recovery vulnerability. The authors specify that the current security metrics are not adapted to Cloud infrastructures, so that new metrics standards must be developed for greater security. Although they clarify indicators of Cloud-specific vulnerabilities, no solutions are presented to solve them.

Gonzalez et al. [13] identify, classify, organize and quantify the security taxonomy-architecture : network configuration, hosts and virtualization issues, applications and services, data security and storage, security management as well as identities and access to Cloud Computing. In addition, the authors present security concerns and solutions using pie charts in order to show the representativeness of each group with identified references. They identify that the security problems on virtualization are the most seriously evaluated at 12%, but the research on solutions for this aspect is only 3%. They propose to develop new mechanisms to isolate VMs, since proper isolation between VM must be implemented to avoid cross-VM attacks due to the sharing of hardware (CPU, storage, memory...). Firewalls

protect the provider's internal Cloud infrastructure against insiders and outsiders, while enabling VM isolation and fine-grained filtering of addresses and ports, thus preventing DoS attacks.

Khorshed et al. [29] organized Cloud Computing security into three sections : security categories (Cloud providers or Cloud customers), security in service delivery models (SaaS, PaaS, IaaS) and security dimensions. They present a survey on the top threats to Cloud Computing and an attack detection for Cloud Computing using machine learning techniques.

Hashizume et al. [20] identify, classify, analyze and list a number of vulnerabilities, threats, mechanisms, security standards, data security, trust, security requirements for the SaaS, PaaS and IaaS delivery models of Cloud Computing. The paper enumerates the threats in detail : service hijacking, stolen data, DoS and VM related issues.

Khalil et al. [28] classify Cloud security threats into five categories : Security Standards, Network, Access Control, Cloud Infrastructure and Data. They compare and analyse only countermeasures such as IDS and Identity Management Systems.

The features of Cloud Computing (large scale, direct access to Cloud infrastructures, resource sharing, etc.) need new and innovative solutions to protect both the users and the provider. Depending on the Cloud model, security relies on the provider or on the user.

As mentioned above, Cloud Computing security is now well documented. However, those studies do not investigate DoS attacks specifically targeting the availability of Cloud Computing. This will be addressed in the following sections.

## 2.4 Attack and Attacker

Before dealing with possible detections and mitigations of attacks on Cloud Computing, the kinds of attacks and the types of attackers that are actually a threat to Cloud Computing shall be addressed. Our first focus will be on the various forms an attack can take. There are multiple scenarios involved in the Cloud infrastructure itself and its environment.

In a DDoS attack, some hosts (VM), also called "bots" or "zombies", can then be controlled remotely. A collection of such bots controlled by a master entity (attacker) is known as a "botnet".

The typical attackers will be classified into three categories, according to their location, their motivation or their level of activity in the attack.

### 2.4.1 Attack

Cloud Computing infrastructures can be compromised in three ways : the attack can come from the outside and target be inside (external to internal), it can even originate from within the system (internal to internal) and it can even occur from within to target the outside of the infrastructure [31].

— *External to internal.* In such case, the botnet used to perform the attack comes from outside the target system. The attack can target the internet gateway of the Cloud infrastructure, or the servers. If a particular client is the VM is victim of an attack, it will also affect the other VMs present on the same physical server of the Cloud (performance interference between VMs).

— *Internal to external.* In such a case, the attack begins by taking ownership of a VM running in the Cloud. This can be done with a Trojan horse. The choice of which customer's VM to infect is important because if this customer owns a large number of VMs, the Trojan horse can potentially spread over all those VMs, therefore forming a botnet. The great computing power and resource availability of the Cloud becomes a real threat for an external target.

— *Internal to internal.* In the Cloud infrastructure, an internal botnet is formed and can attack another target inside the system (such as a VM or a group of VM). All Cloud infrastructures may break down under these kinds of attacks.

With the different kinds of attacks come different types of attackers. Indeed, each attack scenario corresponds to a particular attacker with a specific location and goals.

### 2.4.2 Attacker

The scope of an attack may greatly vary, depending on who perpetrates the attack. System administrators must determine the identity of the attacker to handle an attack adequately and take appropriate actions (to ensure a quick recovery and allow subsequent investigations). We can identify four categories of attackers [40].

— *Insider vs Outsider*. In such a case, the insider belongs to the network that is under attack : he is an authenticated user with privileged access to critical data. Of course, the insider can do more harm than the outsider since the latter would be considered an intruder from the network perspective. Moreover, he would have fewer resources to mount an attack. In the case of Cloud Computing, an insider could be an employee of the Cloud infrastructure, or someone controlling one or several VMs inside the Cloud network, whereas an outsider would not be part of the network at all. For example,

an insider attacker is able to execute arbitrary commands on behalf of a legitimate Cloud user, thus performing a DoS on the user's services, or to create a botnet for charging the Amazon Elastic Cloud Computing costs on the user's invoice [17].

— *Malicious vs. Rational*. Malicious attackers have a general goal of harming the network or the network users (employees or customers of the network). Whatever the costs or the consequences, all means are good to achieve his goal, and such attackers are usually harder to stop or to track since no logic is involved. On the contrary, rational attackers can be more predictable in the way the attacks are lead and which specific targets are reached. Consider the example of a DoS attack in Cloud Computing : a malicious attacker may want to destabilize a government or an organization without any claim or consistent reasons to motivate his actions. However, a rational attacker could be a competitor desiring to create a commercial threat or an organisation leading a DoS against a company or a government for ideological reasons.

— *Active vs. Passive*. Active attackers lead attacks by consciously sending packets or signals while passive attackers may simply eavesdrop. Victims may not even be aware that their machine is under the control of a master machine that forces it to contribute to the attack (a botnet is such an example). In a DoS attack, this defines the difference between the zombies and the master entity (active attacker) : both participating in the attack, but zombies (passive attacker) are never aware that they are vehiculing an attack. In the context of Cloud Computing, an active attacker would have taken control of one or several VMs inside the Cloud network, for instance, and would send huge amounts of traffic or malformed packets to a specific host or subnet in the network. So a legitimate user (passive attacker) whose VM has been taken over by an active attacker, also performs the attack. Sniffing traffic to discover vulnerable links for future exploitations is another form of passive attack.

— *Local vs. Extended*. The scope of the attacker depends on the number of machines he can control. More than just a number, it really is about how those machines are linked together and scattered across the network. An attacker controlling thousands of machines outside the cloud to perpetrate a DoS, would be considered an extended attacker. On the other hand, an attacker in the Cloud, with one or several entities would be described as local. In a Cloud environment, an effective attack may depend on the number of VMs controlled and how the general topology of the Cloud network is organized.

## 2.5 Denial of Service

A distributed-denial-of-service, or DDoS, is a DoS that potentially uses a really high number of hosts to make the attack even more disruptive. The number of hosts can reach thousands if not millions. Most of the time, the machine's owners are unaware that their machines were previously infested and corrupted through a Trojan or a backdoor program.

The actions leading to a DoS, the ultimate goal of which is to compromise the availability of the Cloud, can take place remotely or locally from the victim's or user's service. It generally targets the victim's communication bandwidth, computational resources, memory buffers, network protocols or the victim's application processing logic.

This section specifically addresses DoS and DDoS applied to Cloud Computing networks. DoS and DDoS are not specific to Cloud networks, but they entirely apply to them.

Riquet et al. [41] study the impact of DDoS attacks on Cloud Computing with a defense such as an IDS (snort [43]) and a commercial firewall. Their experiments show that distributed attacks aren't detected, even with security solutions.

As mentioned in [11], DoS attacks on Cloud Computing can be direct or indirect. In direct attacks, the target service or host machine is predetermined although collateral damages may result in indirect DoSs by denying access to other services hosted on the same machine or network. There is even a scenario called *race in power*, induced by a Cloud mechanism that relocates flooded services to other machines. This allows to use Cloud elasticity to mitigate the effects of the attack, but it is entirely possible that it will just spread the workload, in other words, direct the attack to many other servers.

According to [2], a DoS attack can have two objectives. The first consists in overwhelming the target system resources or the network connections, by taking advantage of the superior capacity of the attacker, compared to what the system is capable to cope with in terms of CPU or bandwidth for instance. The second consists of exploiting vulnerabilities in the system by sending specific malicious packets (not necessarily at a huge rate).

### 2.5.1 Overwhelm the Resources

**Exhausting memory** Attacks of this category take advantage of vulnerabilities in Internet protocols, routing and networking devices. They include, for instance, SYN flood attacks that consist of sending many SYN packets, while ignoring the SYN ACK packets. Since the number of simultaneous TCP connections is limited and the server is waiting for the ACK packets, new users cannot get connected. Such attacks could be avoided with proxy-based

applications for instance. The number of simultaneous TCP connections is then much higher, and it decreases the server's memory load, since only the connections that have successfully completed the "3-way handshake" are forwarded to the server.

**Exhausting bandwidth** One way to overwhelm the target system is to exhaust the bandwidth. The aim is to flood the network to prevent legitimate users from accessing the Cloud infrastructures, by imposing greater traffic than the available bandwidth. In this case, more and more packets are dropped, including the legitimate ones. An example of such an attack is given in [32]. The first step is to gain access to the topology (or at least enough to reveal useful information such as a bottleneck uplink). According to the author, an attack has really little chance to succeed if it does not take the topology of the Cloud into account, and more specifically all the vulnerable links. The second step is to take possession of enough hosts in the target subnet and to produce as much UDP traffic as possible through the vulnerable uplink (by targeting hosts in a specific subnet for instance). The choice of UDP is motivated by the expected starvation of the legitimate TCP sessions due to the TCP congestion handling mechanisms. In the case of CPU intensive requests, the system will predominantly process the malicious packets rather than the legitimate ones.

**Exhausting computing time/bandwidth** This attack steals computing time/bandwidth from other users. With Amazon's Cloud platform and Elastic Compute Cloud (EC2) services, an attacker boots up a massive number of machines. With a script Twill, multiple accounts are created and run the machines. This recursive registering of accounts and booting of machines means that the number of running machines grows exponentially. This may continue until the system can no longer handle the machine load [5].

**Exhausting computing time** In oversized payload attacks, an attacker sends an excessively large payload to deplete the victim's system resources. SOAP messages from an attacker contain a large amount of references to external entities to force the server to open a large number of TCP connections to download the actual contents of the entities, and consequently uses a large amount of CPU cycles to process the downloaded contents.

**XML-DoS and HTTP-DoS** Those attacks belong to the resource exhaustion attack category. XML and HTTP are heavily used in Cloud Computing web services and very little work has been done to ensure security related to these protocols, because most of the time, for example with XML (XML encryption, digital signatures, user tokens, etc.), the request is

implicitly assumed to be necessarily legitimate. This puts XML-DoS and HTTP-DoS among the most destructive DoS attacks in Cloud Computing.

As Ye et al. in [54] explain, web services rely on SOAP (Simple Object Access Protocol) to send and receive messages. Yet SOAP uses XML, which can be used to perpetrate XML-DoS attacks, based mainly on three strategies. The first uses an oversized payload to deplete the target system resources. The second is the External Entity DoS Attack. In this attack the server is forced to resolve many large external entities (remote XML files) defined within the Document Type Definition (DTD). This means opening many TCP connections on the one hand, and making extensive use of the CPU to process the entities on the other hand. Eventually, the third strategy, the XML Entity Expansion Attack, forces the server to recursively resolve entities defined within the DTD, which makes intensive usage of the CPU and the memory.

The Coercive Parsing [52] attack is one such example of XML-DoS : it uses a continuous sequence of opened tags that primarily exhausts the CPU, but also the memory. Other forms of coercive parsing include many namespace declarations, a large prefix, namespace URIs, or very deeply nested XML structures [50]. However, this attack can only be successful if the web service uses a DOM parser that creates a tree representation of the XML document.

Padmanabhun et al. [39] give an overview of the underlying issues behind XML and how this can lead to a DoS. They explain how SOAP allows to send and receive XML messages regardless of the underlying implementation of the application or the transport protocol (HTTP, SMTP, etc.).

A HTTP-DoS consists of sending a lot of arbitrary HTTP requests to a particular web service [50]. The processing of all the requests and the cost associated with each request (which may be quite heavy for certain web services) eventually triggers the DoS. Here again, this kind of attacks is very difficult to detect.

### 2.5.2 Exploit the Target System's Vulnerabilities

Antunes et al. [2] give an overview of those attacks and propose a method to automatically and systematically detect the vulnerabilities that can lead to a DoS. Those attacks are perpetrated by way of a malicious interaction with the target system. This results in either a crash or a service degradation. This can be caused by a design flaw or a software bug, for instance. As the authors point out, for system administrators, the factors leading to those kinds of attacks are very difficult to detect and are therefore to be avoided, since the vulnerability may only be leveraged under very specific conditions or after many activations. They define

resource-exhaustion vulnerabilities as "a specific type of fault that causes the consumption or allocation of some resource in an undefined or unnecessary way, or the failure to release it when no longer needed, eventually causing its depletion".

### 2.5.3 Section Summary

Because of its very own nature, Cloud Computing is vulnerable to DoS and DDoS attacks, but it also offers great opportunities to recover quickly from these attacks since resources can be provisioned very easily and quickly [12]. Hence, at first sight, a DoS attack appears to be harder to implement and its success not granted given that the attackers need a lot more resources to achieve their goal in the case where Cloud infrastructures are well designed). However, service providers should still take those attacks into account, otherwise Cloud elasticity would be used to serve huge amounts of illegitimate traffic, which is costly. Moreover, due to the growing botnet market, e.g. people selling access to infested machines), one cannot presume the extent of an attacker's strike force.

## 2.6 DoS Defense

The available Cloud Computing DDoS defenses cover various aspects, such as prevention, mitigation strategies and security architectures (see Table I). During a DoS attack, the most important thing is to maintain the availability for the service providers, the end users and the Cloud infrastructure managers.

Defending against DoS attacks is difficult. A DoS could theoretically be stopped by identifying and then blocking the unique source of the attack. Not only is this not always easy, but most of the time, the attack leverages a huge amount of bots through a DDoS attack.

### 2.6.1 Prevention

**Service Level Agreements.** Kandukuri et al. [25] demonstrate the necessity of an exhaustive and standardized Service Level Agreement (SLA), which is the only legal agreement between a client and the service provider for availability, confidentiality and trust. An SLA can take care of the following : (1) privileged user access, that assures the customer outsourced sensitive data do not fall into malicious hands, (2) regulatory compliance, that holds the customer ultimately responsible for his own data, and subjects the service provider to external audits and security certifications, (3) data location, that is a commitment to comply to local jurisdiction and to store and process data in specific jurisdictions only and (4) data

| | | Techniques |
|---|---|---|
| **Defense Strategy** | **Prevention** | Service Level Agreement |
| | **Attack Mitigation** | Virtual Machine Monitor<br>IDS<br>Firewall<br>Detection<br>Filter<br>Limitation<br>TraceBack |
| | **Architecture** | |

Table 2.1 Cloud Computing DDoS Defenses

segregation : data must be properly encrypted to avoid leakages between users sharing the same environment. The authors also provide a list of questions that SLA must answer.

### 2.6.2 Attack Mitigation

To eradicate an attack, there are five general requirements [31]. First, detect the attack as quickly as possible and determine its magnitude (determine the impact and their level of significance). Second, try to mitigate the effects of the attack as much as possible on the various attackers (section 2.4.2). Third, if step two is not sufficient enough or impossible, migrate the VM under attack to safe physical servers. In order to do so, there is a fourth requirement guaranteeing network bandwidth. Eventually, put an end to the attack with countermeasures that rank from very basic to highly complex. Whatever the measure, it will not be perfect for every situation and often a compromise must be made when choosing one or another.

A general rule, to prevent such attacks, the resources allocated to users must be limited to a bare minimum. For authenticated users, it is possible to establish quotas to limit the load a particular user can put on the system. In particular, one might consider handling only a single request per user at one given time, by synchronizing the user'sessions. However, this solution remains problematic due to the choice of quotas and the resulting quality of service for the end user which may deteriorate. A more efficient solution would be to dynamically use the scalability of Cloud infrastructures to maintain availability while the attack is being eradicated.

**Virtual Machine Monitor (VMM).** Zhao et al. [56] propose a virtual machine monitor

(VMM) composed of a tagger, a duplicator and a detector. The goal is to monitor and compute the amount of available resources and to compare it to a threshold to decipher the presence of an attack. Since the VMM has greater privileges than the guest operating system, it can monitor and evaluate the guest's performance. Under attack, the OS and all the applications are moved to a new isolated entity. During the migration process, there is no service interruption for the user under attack since the applications are still running in both the original VM and in the new isolated VM. Basically, the only difference with duplication is that the original VM is destroyed when the migration is complete. This way, the attack has no more influence on the user's applications. The difficulty is to correctly set the threshold value that indicates an attack. Another difficulty lies in the fact that the Virtual Machine Monitor (VMM) exists whether or not there is an attack. Thus, most of the time, it is likely to be idle. However, the advantage of this system is the possibility to migrate the VM without interrupting the service which is a significant advantage of this system. There is no need to migrate the entire VM, only the selected applications and OS.

**Intrusion Detection Systems (IDS).** An Intrusion Detection Systems (IDS) can be used in VMs. IDS can be classified in two categories [36] : Host-based Intrusion Detection Systems (HIDS), and Network-based Intrusion Detection Systems (NIDS). For HIDS, the detection applies for a specific host, whereas a NIDS is used for all the traffic inside a particular network.

**IDS and Behavior, Knowledge Analysis.** Vieira et al. [49] propose an architecture (node, service, event auditor, storage service) for IDS to examine network traffic, log files and user behaviours. Each node must alert other nodes when an attack occurs. A node contains the resources (through middleware), the service provides functionality, the event auditor monitors the data to analyze and the storage service uses Behavior and Knowledge Analysis : data mining, artificial neural networks, artificial immunological systems and expert system. Data from both the logs and the communication system are used to evaluate the Knowledge-Based System. A series of rules was created to build a security policy that should be respected.

**IDS and Cloud Fusion Unit.** Lonea et al. [34] propose a solution to combine Intrusion Detection Systems (IDSs) deployed in VMs of the Cloud system with a data fusion methodology at the front-end using the Dempster's combination rule (DST). The Intrusion Detection Systems (IDSs) are installed and configured in each VM. A mysql database is installed in the Cloud Fusion Unit (CFU) of the front-end server. An alert in IDSs will be stored in the database. The Cloud Fusion Unit (CFU) comprises 3 components : Mysql database (storing the alerts), basic probabilities assignment calculation (DST) operations and attacks assessment. Their solution is not associated with any experimentation.

**IDS and Queueing Theory.** Yu et al. [55] propose an Intrusion Prevention System (IPS)

between a Cloud data center and the Internet to monitor incoming packets. To mitigate DDoS attacks on individual Cloud customers, the mechanism will automatically and dynamically allocate extra resources from the available Cloud resources pool. To estimate the resource allocation a queuing theory is used. The mitigation problem is an optimization problem : minimizing the resource investment (CPU ; memory ; IO ; bandwidth) while guaranteeing the average time in the system of packets. However, some statements in the article are false : (1) the attack capability of a botnet is usually limited. Consequently, the authors find it reasonable to expect that a Cloud can manage its reserved or idle resources to meet demand, (2) all attack packets are filtered and all legitimate packets go through the IPS system.

**Firewalls.** Modi et al. [36] explain that firewalls protect the front access points of Clouds and are treated as the first line of defense. Firewalls filter (1) by inspecting only header information such as source or destination address and the port number, (2) with a state table (request and server responses), (3) by analyzing the protocol syntax by breaking off the client/server connection.

**Clusterized firewall.** Liu et al. [33] propose a clusterized firewall framework for Cloud Computing. They divide the Cloud services into application layers in which the servers are grouped into clusters, for a type of Cloud data service center. Each cluster has a firewall. The firewall for each cluster protects applications according to the arrival rate and thus guarantees QoS for legitimate users. Each cluster can be modeled as a M/G/1 queueing system to obtain the key measures : (1) the request response time and (2) how many resources are needed to guarantee the QoS. These key metrics evaluated the Cloud defense.

**Attack Detection.** With statistical machine learning techniques, Khoshed et al. [29] propose a Support Vector Machine technique to identify top attacks. It should also warn the system administrators and data owners of the type of attack, and suggest possible actions to take. Eventually, customers would be aware of the attack type even if Cloud providers are reluctant to divulge information about the attack.

### 2.6.3 Security Architecture

**Security Aware Cloud Architecture.** Hwang et al. [21] highlight that the abstraction level of the Cloud model (SaaS being the most abstract and IaaS being the least abstract) influences the number of security aspects that will be handled by the provider (more abstraction means the provider will be in charge of more security aspects). In the intermediate case of PaaS, users remain in charge of confidentiality and data privacy, yet the provider is responsible for data integrity and availability. Generally speaking, they propose a Security Aware Cloud Architecture that offer protection to secure public Clouds and data centers. The

mechanisms are (1) trust delegation and negotiation architecture, (2) worm containment and DDoS defense, (3) reputation system of resource sites, (4) fine-grain access control and (5) collusive privacy prevention. No tests assess the performance of their architecture.

**DDoS Attack Mitigation Architecture : DaMask.** Wang et al. [51] propose a DDoS attack mitigation architecture. The Software-Defined Networking (SDN) is an approach that allows network administrators to manage network services by way of abstraction of lower-level functionality. The authors find that the SDN and Cloud Computing can enhance the DDoS attack defense. The DaMask architecture has three layers : network switches, network controllers, and network applications. Two separate modules are (1) DaMask-D, a network attack detection system, and (2) DaMask-M, an attack reaction module. DaMask-D already has an efficient attack detection algorithm with a very low overhead. DaMask-M defines three basic operations : forward, drop and modify the packet. Those operations are implemented as a set of APIs, consequently the defenders can customize the countermeasures.

### 2.6.4 Defenses against XML-DoS and HTTP-DoS Attacks

**Filtering Tree.** Karnwal et al. [26] developed a filtering tree, which works like a service. The XML consumer request is converted into a tree form and uses a virtual Cloud defender to defend against these types of attacks. The Cloud defender basically consists of five steps : sensor filtering (check number of messages from a particular user), hop count filtering (number of nodes crossed from source to destination – this cannot be forged by the attacker), IP frequency divergence (the same range of IP addresses is suspect), puzzle (send a puzzle to a particular user : if it is not resolved, the packet is suspect) and double signature. The first four filters detect HTTP-DDoS attacks while the fifth filter detects XML-DDoS attacks.

**Limitation.** Karthigeyan et al. [27] explain that an acceptable solution to prevent attackers from exhausting the victims' network bandwidth and computing power is to route the requests to the service providers only once they have been authenticated and validated. First, limit the payload size. Then, limit the time allocated to a SOAP request. Third, limit the number of requests a particular user can send within a given time frame. Packets that do not match those criteria are discarded and the service is blocked for the user for a certain period of time. They also propose to impose limits for the XML parser. For example, limit the number of attributes an element can have, the quantity of bytes in a XML message, the depth of nested elements and the size of all nodes in the XML document. Furthermore, to minimize the impact on the QoS for the end user in terms of delays for instance, this could take place only when the system is under attack, which is detected by the service provider.

**Cloud Protector and Decision Theory.** Chonka et al. [7] [6] develop (1) a Cloud Trace-

back (CTB), that uses a Service-Oriented Traceback Architecture (SOTA) approach. CTB is deployed at the edge routers in order to be close to the Cloud network source end to mark all outgoing packets. If an attack is detected, the use of a back propagation neutral network, called Cloud Protector, allows to retrieve the source of the attack. The Cloud Protector is a trained back propagation neural network, which means that there is a set of connected units associated with a given weight, spread between input, hidden and output layers. Then, the weights are summed up to see if the result is above a certain threshold, which means an attack is taking place. (2) a method relying on decision theory, called ENDER (Pre-Decision, Advance Decision, and Learning System). It uses two decision theory methods to detect attack traffic and mark the attack messages. If an attack message is detected, a Reconstruct and Drop (RAD) system removes the message before a victim is harmed.

**Defense in Cloud Broker.** Vissers et al. [50] divide the concept of Cloud Computing into three parts. First, the Cloud providers deploy the VM and their web services. Then the Cloud broker makes the link between the user and the available resources of the Cloud providers in order to allocate the necessary resources. The users who request resources in the Cloud infrastructure through the Cloud broker to eventually use the web services hosted by the Cloud providers. As the author points out, the Cloud broker introduces a single point of failure, since its unavailability makes the Cloud infrastructure unusable. To make this architecture more secure, a DDoS defense system is placed with the Cloud broker to decide whether or not the application request should be rejected. The defense system, that uses DDoS datasets, is incorporated in all the broker entities. The filter is based on the definition of a normal profile usage constructed with previous requests. The filter, aimed to be scalable to overcome a DoS, is transparent for the user. A request must go through the HTTP header filtering (HTTP floods, non-existing SOAP Action usage and content-length outliers), and then through actual XML content filtering (SOAP feature outlier detection and SOAP Action or WS-Addressing spoofing). This defense mechanism has proved to be successful at detecting and mitigating all listed vulnerabilities. In addition, it might even be able to handle unknown vulnerabilities with minimal time overhead.

**Flexible, Collaborative, Multilayer, DDoS Prevention Framework (FCMDPF).** Saleh et al. [44] propose a framework composed of (1) an outer attack blocking (OB) at the edge router, (2) a service traceback oriented architecture (STBOA), (3) and a flexible advanced entropy based (FAEB) layer. From a blacklist database table (IP source), the OB layer blocks or forwards the incoming request. The STBOA layer is designed to validate whether the incoming request is launched by a human (real web browser) or by an automated tool (bots). A puzzle or random number is sent to the client or the requester to solve. If after verification, the puzzle or random number are correct, the request is forwarded to the next

level. Otherwise, it will be blocked immediately and a blacklist is updated. The FAEB layer computes entropy of overall requests to determine flash crowd or HTTP attack. The entropy of incoming requests that are launched towards hot pages of the website determines the flash crowd. In the case of an HTTP attack, the blacklist is updated. The disadvantage of this framework lies in the information on the blacklist and its updates.

### 2.6.5 Section Summary

The defenses to protect the Cloud availability need some information ; blacklist, the normal profile, threshold determination and limit fixation. Hence, if those values are wrong, the defenses can lead to false positives or negatives.

## 2.7 How to Evaluate Defense Systems

The proposed defenses for DDoS attacks must prove their efficiency. For this purpose, research authors must design experiments and determine metrics to assess performance. Table 2.2 we have listed all the information for the proposed defenses. Some defenses are compared with others. However, the following authors have tried to implement, simulate and test their proposed solutions.

**Theoretical Evaluation**

Zhao et al. [56] propose to monitor the VM's available resources to decide if it is under attack, in which case the system selectively duplicates tagged applications and operating systems. Their defense system has yet to be tested, but the authors have identified four benefits and one disadvantage with this method. The isolated environment exists whether or not there is an attack, so that most of the time it is likely to be idle. From a performance point of view, they show that their system is not necessarily more costly than VM migration. As far as performance is concerned, they theoretically evaluate the total time consumption of their defense system.

**Data Collection**

Vieira et al. [49] propose a refinement to traditional IDS to be more efficient in a Cloud environment. To test their system, they use three sets of data. The first represents legitimate actions. In the second, they altered the services and their usage frequency to simulate anomalies. The last set simulates policy violation. To evaluate the event auditor that monitors the requests received and the responses sent on a node, they chose to examine the communication elements, since data from log present little variation, making attacks hard to detect. A feed-forward neural network is used for the behavior-based technique, and the simulation

| Experiment | Efficiency |
|---|---|
| **Design** | **Impact of defense system** |
| Data collection | Overhead bandwith |
| Simulation | Processing time |
| TestBed | Detection attack |
| | False alarm |
| | False negative |
| **Parameters** | **Mitigation** |
| Attack Rate | Filtration |
| Attack Duration | Limitation |
| | QoS |
| | Migration VM |
| | Traceback |

Table 2.2 Summary of Evaluation Defense System in DDoS attack

includes five legitimate users and five intruders. Their scenario simulates ten days of usage. Although the results yielded a high number of false negatives and positives, its performance improved when the training period of the neural network was prolonged. To evaluate their behavior-based system, they looked for the number of false positives and negatives. They show that their system always had more false negatives than false positives. With still a high level of uncertainty, the false system alarms disappeared within 16 days of simulation training. While further extending the training period, they noticed even lower false positives, but also the very non-deterministic nature of neural networks, since false positives were not stable after several iterations. To evaluate the Knowledge-Based System, they took data from both the logs and the communication system. They created a series of rules to build a security policy that should be respected. They conclude that their system could allow real time analyses, provided the number of rules per action is not too high.

**Simulation**

Liu et al. [33] use three parameters : the attack rate, the attack duration and the rule processing time. They showed that a larger matching probability (that is to say rules that are easier to match) means a reduced response time. Hence, they encouraged Cloud defenders to put those rules at the top of the rules list so as to increase users' satisfaction. They demonstrated, both analytically and experimentally, a direct correlation between the response time and the number of rules and attack rates. To estimate the cost of their system, they rented 20 VMs from Amazon EC2. In the end, running their clusterized firewall turned out to cost 38$ US for one day and 266$ US for one week, while keeping in mind that long attacks are extremely rare, given that they are easily detected.

**Testbed**

Wang et al. [51] set up a hybrid Cloud, using Amazon EC2 as a public Cloud, and simulating a private Cloud in their lab. The private Cloud consists of two Linux machines, one of which hosting DaMask and the network controller while the other emulates a virtual network to extend the private Cloud. They wanted to measure the communication costs as well as the computation overhead of DaMask. They began by computing the network bandwidth between the private and the public Cloud, with and without DaMask. For the communication overhead, caused by the traffic being examined by DaMask, they showed the overhead was a constant if the link status of network remained stable. As for the computation overhead caused by the detection algorithm, they ended up with an interference time of 80 ms, which they consider quite efficient. DaMask also proved successful at adapting to a topology change in real time, by monitoring the effects of migrating a web server from one switch to another.

Chonka et al. [7] evaluated their Cloud Traceback (CTB) and Cloud Protector in multiple ways. They conducted experiments to see how CTB marked packets and could determine if they lead to a X-DoS, but also how accurate CTB was at identifying the source of the attack. Moreover, they wanted to make sure their method was better than traditional security mechanisms such as WS-Security, when it comes to X-DoS attacks. The experimentation for the Cloud Protector consisted in determining if it could detect and filter H-DoS and X-DoS messages. To generate the attack traffic, they opened up three virtual servers that contained 20 Firefox browsers and 20 open tabs to each browser, in addition with a page refresher tool and targeted a particular website. First, they studied the impact of the attack without their architecture and witnessed its tremendous and lasting effect as the web server was quickly not able to handle more than a few requests. Then, they divided the evaluation of CTB in two parts : one to simulate X-DoS attacks against a web server and the other to compare CTB to traditional mechanisms. In their simulation, 100 messages were sent, and if one was a X-DoS attack, it was supposed to crash the server with a probability of 1/2. Out of those 100 messages, 9 were successful at crashing the server, and CTB was able to identify 7 of them, with a response time between 480 and 550ms. When comparing CTB with SOAP authentication, CTB proved to be far more effective in terms of response time and the same was true for WS-Security. Then, they assessed the performance of the Cloud Protector (a neural network), by training and testing it with a dataset they developed. On the trained dataset for H-DoS, the Cloud Protector was able to identify 91% of the attack (9% of false positives), but with significantly different response times (between 20ms and 1s). On the test dataset, the accuracy decreased at 88%, with the same variation in response time. For X-DoS, most of the attack messages were identified, detecting and removing them taking between 10 and 140ms, but the response times were not as scattered as in the H-DoS case.

In [6], Chonka et al. come up with a new defense system called ENDER, that has no more than 1% of false positives on the same dataset.

Vissers et al. [50] used the Eucalyptus middleware to manage resources in their experiment. The Cloud Resource Broker along with the DDoS defense system is installed on a 4 CPU server. After they set up Eucalyptus and the web services, they wanted to evaluate the impact of the attack, the mitigation capabilities of their system and the cost induced by this defense mechanism (extra response time). When the defense system was not present, a flooding attack with 4,900 legitimate requests during 60 seconds was enough to exhaust the CPU, whereas with the defense system, only the first request was accepted, the others were discarded because they exceeded the request rate. Moreover, some addresses were blacklisted. All in all, 10% of CPU was necessary to drop malicious packets. With an oversized XML attack, ten 12MB messages were sufficient to keep the CPU busy for 15 minutes, while also significantly increasing memory usage. With the defense system, the CPU load was only around 10% for 20s, while blocking the attack, and with no additional memory usage. The mechanism also successfully handled a coercive parsing attack, an oversize encryption attack and a spoofing attack. To evaluate the response time of their system, they sent 200 SOAP requests at a rate of 2 requests per second, and then 10,000 SOAP requests at 50 requests per second, both with and without the filter in place. In the first load, introducing the defense system made the processing time of the web service go from 3 to 5ms, whereas in the high load, it went from 3 to 9ms. Those results outperform the current existing solutions (for example the Cloud Protector mentioned above).

Saleh et al. [44] conducted four experiments. First, they evaluated how the framework could protect against flash crowd attacks, then against high rate DDoS attacks. Third, they studied the ability to validate clients and trace the true IP source of the attack. Eventually, they evaluated the proper blocking of the attacking IP address, as close as possible to the network entrance. They used the following parameters to make the evaluations : the number of incoming requests and number of detected and prevented attacks against the web application (conducted by way of the Apache log, based on the response code number to the incoming request – an error code means the attack was detected). Finally, 420,000 incoming requests were generated to cause a DoS. The AntiDDoS_Shield system detected and prevented all high rate HTTP-based DoS/DDoS attacks : 369,726 out of 420,000 flash crowd (FC) attacks, at the edge router. AntiDDoS_Shield system succeeded in validating and tracing back 369,726 out of 420,000 incoming requests.

In summary, apart from defense architectures that have no evaluations whatsoever, most defenses came with experiments on design and performance, whether theoretical or practical.

By using several sets of data and carefully choosing the parameters of interest, the authors were able to study the response of their system under different workloads, while reproducing real life situations. They discussed the possibility of adopting their system regarding the cost, scenario (real time), best practices (how to correctly set up the system for maximal performance), overhead, precision (rate of false positives or negatives), effectiveness to detect and filter attack messages, and ability to detect new attacks.

## 2.8 Conclusion

This article provides an overview of Cloud Computing : what it is, what it can achieve, what technologies it uses and why it is so promising in terms of costs, performance and adaptability. Being a combination of existing technologies such as VM, web services, servers, network links, etc., this new paradigm comes with known vulnerabilities, but also new kinds of attacks because of the innovative way services are presented to the user and because of the growing success and adoption of Cloud Computing, both by companies and individuals. Taking advantage of its great scalability and elasticity, Cloud Computing apparently offers adequate resistance to attacks. This review proves that many attacks can still cause great harm to Cloud Computing, impacting all the important security aspects (confidentiality, integrity, isolation, availability, etc.). Among those attacks, the DoS attacks are arguably the easiest to mount and the most destructive, yet huge gaps still exist to efficiently deal with those attacks. We presented some of the state-of-the-art solutions : some were quite easy to incorporate in existing Cloud infrastructures but could not detect nor perfectly mitigate all the possible attacks ; others were much more efficient, but much more complex. In all cases, and as always in the security field, no solution is perfect. Eventually, it all comes down to what compromise the system administrators are willing to make. We also gave an original focus on the different facets of the attack and attacker applied to Cloud Computing, a key parameter to know in order to provide the best security solutions. The extensive study of XML-DoS and HTTP-DoS allowed to show all the available countermeasures. Also, DDoS defenses are evaluated with appropriate metrics and experimental design.

# CHAPITRE 3 ATTAQUES DE TYPE XML-DOS ET IMPLÉMENTATION DANS UN SIMULATEUR D'INFONUAGIQUE

L'implémentation des attaques et des défenses ont eu pour cadre le simulateur GreenCloud. Comme nous allons le voir, ce simulateur s'est imposé comme le meilleur choix pour l'étude qui nous concerne. Mais commençons par nous interroger sur ce qui rend un réseau d'infonuagique vulnérable à des attaques DoS, et en quoi la problématique des attaques XML-DoS est différente sur un réseau d'infonuagique comparativement à un réseau classique.

## 3.1 Vulnérabilité des réseaux d'infonuagique aux attaques XML-DoS

### 3.1.1 Les équipements vulnérables dans un réseau d'infonuagique

#### Les machines virtuelles

Comme il a été indiqué dans l'introduction, l'utilisation de machines virtuelles (plusieurs systèmes d'exploitation coexistants sur une même machine physique) pour délivrer les services aux utilisateurs implique que ceux-ci doivent se partager les ressources à leur disposition. Notamment, une machine qui aurait une utilisation abusive des ressources (notamment CPU) aurait un impact sur toutes les autres machines virtuelles situées sur la même machine physique. Une attaque directe sur une machine donnée (un client dont la machine a été infectée ou bien une machine que l'attaquant possède) pourra se transformer en attaque indirecte sur les autres utilisateurs de la machine physique, qui pâtiront eux aussi d'une dégradation des performances. Attaquer une machine virtuelle est donc d'une grande efficacité, notamment pour monopoliser le CPU et en priver les autres utilisateurs. Par ailleurs, il est à noter que les méthodes de virtualisation utilisant un "hypervisor" induisent un surcoût de traitement important. Il a notamment été montré qu'un serveur web hébergé sur une machine virtuelle servait significativement moins de requêtes que le même serveur web hébergé sur une machine physique, cet effet s'accentuant nettement sous l'effet d'une attaque de déni de service. Ainsi, l'attaquant aura besoin de moins de ressources pour atteindre un déni de service, lorsque sa cible se situe sur une machine virtuelle. Néanmoins, il est difficile de rendre compte de cet effet dans un contexte de simulation (le simulateur GreenCloud ne prend pas cette caractéristique de la virtualisation en compte), nous avons donc décidé de laisser de côté cette caractéristique pour l'instant. Il pourra s'agir d'une amélioration possible future à notre travail. La difficulté est alors de trouver une façon d'y accéder. La portée de cette attaque reste néanmoins confinée à la machine physique ciblée.

#### Le "broker"

Le "broker" a pour rôle d'écouter la requête initiale de l'utilisateur, de surveiller quelles sont les machines du réseau disponible pour recevoir ces requêtes, et d'allouer les requêtes sur ces machines. Puisque ce "broker" est la porte d'entrée des utilisateurs dans le réseau, si cette entité venait à être rendue indisponible à la suite de l'attaque, cela rendrait de fait le réseau inutilisable, signifiant alors une indisponibilité complète de tous les équipements du réseau. Contrairement à l'attaque d'une machine virtuelle, cette fois-ci, c'est tout le réseau qui est menacé par une seule et même attaque (avec l'attaque précédente, pour obtenir le même résultat, il faudrait attaquer l'ensemble des machines physiques qui composent le réseau). Cette entité est donc qualifiée de "weak spot" ou encore "single point of failure", c'est-à-dire que son bon fonctionnement est requis sous peine de faire tomber tout le réseau, et la protéger est donc d'une importance primordiale.

#### Conclusion sur les équipements vulnérables

Nous avons donc montré que les machines virtuelles déployées dans un réseau d'infonuagique ainsi que le "broker" apparaissent comme des équipements vulnérables dans les réseaux d'infonuagique. C'est le cas plus généralement de toute machine hébergeant un serveur web, à la différence près que le bon fonctionnement des machines virtuelles et du "broker" est crucial pour les utilisateurs légitimes. C'est au travers de ces équipements qu'un attaquant pourra lancer avec succès une attaque de déni de service et priver les autres utilisateurs légitimes des services offerts par un réseau d'infonuagique. Intéressons-nous maintenant aux différentes possibilités d'attaque de ces équipements.

### 3.1.2 Attaque de ces équipements : les différentes façons d'accéder aux services d'infonuagique

L'infonuagique se divise en plusieurs modèles : IaaS, PaaS et SaaS. Chaque modèle correspond à un service différent, et à une gestion à la fois logicielle et matérielle plus ou moins grande laissée à la charge de l'utilisateur. Ainsi, il convient de distinguer ces modèles en termes de vulnérabilités et de menaces de sécurité.

#### Le modèle IaaS : l'exemple de Amazon EC2

Afin de déterminer de quelle façon on peut prendre avantage des équipements vulnérables, intéressons-nous aux modes d'accès aux infrastructures d'infonuagique. Nous pouvons prendre par exemple le service Amazon Elastic Compute Cloud (Amazon EC2) [1], qui

s'inscrit dans le cadre du modèle IaaS, c'est-à-dire qu'il fournit l'infrastructure (virtuelle), comme la machine virtuelle, le stockage, les adresses IP, etc. Amazon EC2 fournit des environnements virtuels appelés "instances", et qui sont en fait des machines virtuelles utilisées comme serveurs privés virtuels. Il existe différentes façons de louer ces machines virtuelles. Par exemple, il existe une location tarifée à l'heure et sans engagement, ou une location avec engagement offrant des réductions sur le tarif à l'heure. Une instance correspond à une combinaison de CPU, mémoire, capacité de stockage et réseau. Chaque instance est disponible en plusieurs tailles pour pouvoir s'adapter à la charge (mise à l'échelle). Pour aller avec ces instances, il y a les Amazon Machine Images (AMIs). Les AMIs comprennent notamment le système d'exploitation ainsi qu'un certain nombre d'applications. Elles sont à choisir dans une liste d'images préconfigurées, mais il est possible d'en créer de nouvelles, contenant nos propres applications et données. Pour commencer à utiliser Amazon EC2, il faut donc sélectionner une AMI, choisir quelle(s) instance(s) nous convien(nen)t le mieux, et en démarrer autant que souhaité. L'interaction avec Amazon EC2 se fait au travers d'un navigateur web.

L'utilisation d'Amazon EC2 est bien entendu payante, mais il est possible de l'utiliser gratuitement pendant 1 an, à raison de 750 heures par mois, à condition de s'en tenir aux instances t2.micro Linux et Windows (c'est-à-dire la plus petite instance).

#### Le modèle SaaS

Ce modèle est à rapprocher du concept de "logiciel à la demande". L'utilisateur n'est plus en charge de l'installation, la configuration ou le fonctionnement de l'application. Les Google Apps [14] ou Office 365 [38] en sont des exemples. L'accès à ces applications se fait là encore grâce à des services web, c'est-à-dire que tout passe par un navigateur web.

#### Le modèle PaaS

Ce modèle se situe entre les deux précédents. En plus de ce qui est déjà fourni dans le modèle IaaS, le modèle PaaS ajoute un système d'exploitation, un environnement de programmation, et éventuellement une base de données ou un serveur web. Il peut s'agir par exemple de AWS Elastic Beanstalk [4].

**Conséquences sur les différents scénarios d'attaque possibles**

Concernant le modèle IaaS, pour mener à bien une attaque XML-DoS, il faut soit être soi-même détenteur d'une "instance", soit compromettre un utilisateur légitime. Une fois en possession d'une machine virtuelle du réseau, soit la cible de l'attaque est cette même machine virtuelle (l'attaquant attaque sa propre machine virtuelle ou celle qu'il a corrompue), soit la cible est un serveur web situé sur une autre machine virtuelle du réseau (et qu'il est possible de découvrir seulement une fois à l'intérieur du réseau). L'effet peut être décuplé en mettant à contribution plusieurs machines virtuelles, qui enverraient alors toutes leur traffic vers le même serveur web. De plus, dû au fait que plusieurs machines virtuelles doivent cohabiter sur la même machine physique, l'attaque pourrait aussi avoir pour conséquence de diminuer la qualité de service pour tous les utilisateurs situés sur la même machine physique que la machine virtuelle cible (interférence de performance entre machines virtuelles). Comme nous venons de le voir avec le cas d'Amazon EC2, il est très facile d'avoir accès à une machine virtuelle, et ce, même de façon gratuite.

Dans le cas du modèle SaaS, l'attaque sera moins compliquée à mener, l'attaque portera directement sur le serveur web délivrant l'application que l'on cherche à accéder : au lieu d'envoyer une requête légitime, il suffit alors d'envoyer une requête XML-DoS.

Mais au-delà de l'attaque d'un serveur web en particulier, l'attaque sera bien plus efficace et aura de bien plus grandes conséquences si l'architecture du réseau comprend un "broker", mal protégé de surcroît. En effet, l'objectif n'est alors plus de rendre les machines du réseau indisponibles individuellement, mais d'empêcher tout bonnement aux utilisateurs de rentrer sur le réseau.

### 3.1.3 Impact de ces attaques

L'impact de l'attaque va dépendre du modèle attaqué ainsi que des moyens mis en oeuvre par l'attaquant. Dans le cas du modèle IaaS, l'attaquant pourrait seulement attaquer la machine à laquelle il a accès, et donc empêcher l'utilisateur légitime de faire fonctionner sa machine virtuelle (si l'attaquant a eu accès à la VM de façon illégitime), ainsi que perturber les autres utilisateurs situés sur la même machine physique (interférence de performance). L'impact pourra être de plus grande ampleur si l'attaquant prend le contrôle de nombreuses machines virtuelles et lance une attaque XML-DoS contre un des serveurs web du réseau d'infonuagique, par exemple.
Dans le cas du modèle SaaS, une attaque pourrait rendre temporairement indisponible pour tout utilisateur une application web délivrée par un réseau d'infonuagique.

### 3.1.4 Spécificité des réseaux d'infonuagique quant aux attaques XML-DoS

Les réseaux d'infonuagique ont des caractéristiques intrinsèques que ne possèdent pas les réseaux classiques. Notamment, leur grande échelle et la possibilité de fournir des ressources à la demande avec une grande élasticité invitent à considérer les attaques de déni de service sous un nouvel angle. Nous pouvons légitimement penser que de telles propriétés rendent plus difficiles ces attaques, qui ont pour but d'épuiser les ressources d'un système : est-il encore réaliste d'espérer lancer une attaque de déni de service contre de telles infrastructures ? Notre étude viendra répondre à cette question. Par ailleurs, les réseaux d'infonuagique utilisent abondamment les machines virtuelles. Là encore, il serait intéressant de s'intéresser à ce que cela implique en termes de performance et de qualité de service pour l'utilisateur, en particulier en présence d'une attaque de déni de service. Il apparait donc qu'étudier les attaques DoS (et plus précisément les attaques XML-DoS) sur un réseau classique et sur un réseau d'infonuagique représente bel et bien deux choses différentes, leur impact et mise en place ne pouvant être comparable. Du fait des caractéristiques d'un réseau d'infonuagique, nous nous attendons à un impact plus faible de l'attaque. En revanche, l'utilisation de machines virtuelles aura probablement une influence négative sur la dégradation du service pour l'utilisateur (moindre performance comparativement à l'exécution de tâches directement sur une machine physique).

L'analyse présentée ici sur les modalités d'attaque et les spécificités des réseaux d'infonuagique répond entièrement à notre premier objectif : analyser la vulnérabilité des réseaux d'infonuagique aux attaques XML-DoS.

## 3.2 Justification de l'intérêt de la simulation

Le choix d'utiliser un simulateur n'est pas anodin. En effet, si nous avions voulu tester nos attaques sur des infrastructures existantes, il aurait fallu faire face à au moins deux problèmes : louer des infrastructures est payant et peut se révéler assez coûteux, et la plupart des fournisseurs des réseaux d'infonuagique n'autorisent pas à utiliser leurs infrastructures dans le cadre de test d'attaques, notamment d'attaques de déni de service. Utiliser un simulateur est donc le résultat d'une contrainte, mais présente aussi des avantages. En effet, rendre notre expérimentation abstraite nous autorise une plus grande liberté : depuis le choix de l'architecture du "data center", jusque dans les caractéristiques de chaque serveur. La possibilité d'ajuster ces paramètres permettra de rendre compte de leur influence sur les attaques. Par ailleurs, puisque nous avons une vue globale du réseau et que nous en "contrôlons" la

totalité, il devient aisé de suivre de façon extrêmement précise l'utilisation (abusive ou non) qu'il en est fait (CPU, mémoire, bande passante, etc..), chose qu'il aurait été impossible à obtenir avec de vraies infrastructures, où la plupart du temps nous ne pouvons obtenir que des délais de propagation et des mesures de débit. Ainsi, particulièrement dans notre cas et l'étude des réseaux d'infonuagique, la simulation est très appropriée. Il conviendra toutefois de garder à l'esprit les inconvénients intrinsèques d'une simulation : on ne peut refléter parfaitement la réalité, et donc déterminer le bon degré de fidélité et de complexité est une tâche à part entière, de plus la simulation peut devenir coûteuse en terme de temps pour des systèmes complexes (comme les réseaux d'infonuagique avec un haut degré de réalisme), et enfin les résultats sont à prendre avec précaution (conséquence du premier point). Concernant le simulateur, notre choix s'est porté sur le simulateur GreenCloud.

## 3.3 Comparaison des simulateurs de réseaux d'infonuagique

Après avoir mis de côté les simulateurs trop spécifiques qui ne visent que des objectifs bien particuliers qui n'entrent pas dans notre domaine d'étude, nous avons isolé deux simulateurs : CloudSim et GreenCloud.

### CloudSim

CloudSim [9] est un des simulateurs d'infonuagique les plus populaires. Il supporte la modélisation et la simulation de réseaux d'infonuagique à grande échelle, la prise en charge de serveurs virtualisés, avec différentes politiques pour allouer les ressources hôtes aux machines virtuelles [8]. Il est utilisé par de nombreuses entreprises et universités à travers le monde. Ce simulateur est dit "event-based", c'est-à-dire que l'on s'intéresse aux effets des interactions entre objets, contrairement à un simulateur qui descendrait au niveau paquet ("packet level simulator"), qui lui va construire une structure de type paquet avec des en-têtes, cette structure étant ensuite traitée en accord avec les protocoles spécifiés dans ces en-têtes. Le fait d'être "event-based" le rend très rapide en termes de temps de simulation, mais fait perdre en précision de la simulation (seul l'effet de l'interaction entre deux entités du simulateur est capturé, ce qui laisse par exemple de côté les traitements associés aux protocoles spécifiés dans le paquet).

### GreenCloud

Ce simulateur, tel qu'indiqué sur le site internet qui l'héberge [15], est un simulateur à la fois "bas niveau" et "discret". Bas niveau, car il permet l'étude du comportement du réseau en

fonction du type de paquet reçu (c'est un "packet level simulator"). Et discret, en permettant un suivi régulier du CPU, mémoire, bande passante, etc.. Si, au départ, il est prévu pour étudier le coût énergétique d'un réseau d'infonuagique, rien n'empêche de s'en servir comme base pour une tout autre étude. En effet, ce simulateur est basé sur le très populaire et modulable simulateur de réseau NS2 [37], qui permet de très nombreuses configurations et scénarios. NS2 est un logiciel libre de simulation de réseau à événements discrets lui aussi, développé surtout à partir de 1995 au laboratoire national Lawrence-Berkeley, au centre de recherche Palo Alto ainsi qu'au sein de l'université de Californie à Berkeley. Il utilise le langage TCL pour les scripts de simulation, et le C++ pour le fonctionnement intrinsèque du réseau. Alors que CloudSim ne nécessite que quelques secondes pour réaliser une simulation d'une heure avec quelques milliers de noeuds, il faudra plusieurs dizaines de minutes à GreenCloud pour réaliser la même simulation.

Finalement, ce qui nous a fait choisir GreenCloud, c'est la possibilité d'influencer le comportement du réseau à partir du type de paquet qu'il devait traiter (caractéristique essentielle dans notre cas), et également une plus grande maitrise du langage C++ comparé au langage Java utilisé par CloudSim. Ce choix s'est même révélé payant a fortiori, puisque GreenCloud étant moins populaire (à l'heure actuelle), j'ai pu contacter un de ses principaux développeurs à plusieurs reprises pour clarifier certains points de la simulation et demander conseil, chose qu'il aurait été difficile à faire avec CloudSim.

## 3.4 Description de l'architecture du simulateur GreenCloud

### 3.4.1 Organisation du code

Le code source est contenu dans le dossier src/. On y trouve notamment le dossier ns2/ qui contient tous les fichiers C++ décrivant le comportement du réseau, et le dossier scripts/, qui permet de choisir facilement des scénarios de simulation. Dans le dossier script/, on décrit notamment le datacenter dans un fichier dc.tcl : caractéristiques techniques des hôtes (CPU, mémoire, stockage...), nombre et types de switches, nombre de machines virtuelles par hôte ainsi que leurs caractéristiques. Le fichier main.tcl fixe le temps de simulation et les actions à exécuter à chaque instant. Le fichier setup_params.tcl décrit les paramètres d'une tâche (demande en mémoire, calcul, stockage...). Le fichier topology.tcl permet de donner le nombre d'hôtes, et le nombre de switches dans chaque partie du réseau (coeur, agrégation, accès). Enfin, le fichier user.tcl indique combien d'utilisateurs vont se partager les ressources du datacenter et quelle sera leur fréquence d'envoi de requêtes. Enfin, on trouve en plus de

ces deux dossiers, un dossier build/ qui contient l'exécutable obtenu après compilation des fichiers source, et un dossier traces/, qui contient les dossiers permettant de suivre l'évolution de la simulation dans le temps.

### 3.4.2 Fonctionnement de la simulation

Comme le fonctionnement et les mécanismes internes du simulateur sont complexes et influent sur la façon dont nous avons abordé les attaques XML-DoS, il nous est apparu important d'en expliquer les bases, afin de mieux comprendre les choix et hypothèses qui ont été retenus par la suite.

#### Les grands concepts

Une fois que le code source a été écrit, il faut le compiler, et ensuite ajuster les paramètres de la simulation au moyen des fichiers tcl, qui, eux, sont simplement interprétés et n'ont donc pas besoin d'être compilés à chaque changement.

4 types de ressources sont définis, ce sont ces ressources dont on suivra l'utilisation : calcul, mémoire, stockage, réseau. Par défaut, la mémoire et le stockage sont des ressources statiques, tandis que le calcul et le réseau sont des ressources dynamiques (ordonnancement CPU et couche réseau de NS2). Les serveurs sont hétérogènes, c'est-à-dire qu'on peut les configurer séparément et indépendamment. Afin de mesurer en temps réel l'utilisation qui est faite de ces ressources, une méthode située dans le fichier source DataCenter.cc, permet de traverser tous les serveurs pour calculer leur charge sur ces différentes ressources. Cette méthode est appelée à intervalles de temps réguliers depuis le fichier record.tcl, puis une moyenne est calculée en fin de simulation.

Concernant la virtualisation, les machines virtuelles sont considérées comme des intermédiaires entre les hôtes et les tâches. Deux abstractions ont été définies : les "ResourceConsumers" et les "ResourceProviders". Les tâches reçues sont des ResourceConsumers, tandis que les hôtes sont des ResourceProviders, et les machines virtuelles sont à la fois des ResourcesConsumers et des ResourceProviders. Ainsi, la simulation commence par allouer les machines virtuelles (alors considérées comme des ResourceConsumers) sur les hôtes, puis on allouera les tâches sur les machines virtuelles créées (qui sont alors vues comme des ResourceProviders). Le calcul de la charge d'un serveur (notamment en termes de mémoire et CPU) utilise les mesures de la charge de chacune des machines virtuelles qu'il héberge. Les systèmes multicoeurs et multiprocesseurs sont supportés. Notamment, l'ordonnancement se fait "par coeur", dans le sens que chaque coeur a sa propre liste de tâches, et que chaque demande de

ressources (matérialisée par la variable ResourceDemands) est allouée à un seul coeur.

Nous allons maintenant voir quelles sont les étapes par lesquelles transite un message envoyé par un utilisateur, entre sa réception par le datacenter, et son traitement sur une machine virtuelle.

#### Étapes de traitement d'un message

Les premières et principales étapes par lesquelles passe un message, avant et après modification du simulateur, sont décrites respectivement dans 3.1 et 3.2. Chaque "CoreScheduler" maintient en tout temps une liste de tâches actives. Lorsque cette liste n'est pas vide, le coeur correspondant fonctionne à 100% de ses capacités, et ce, jusqu'à ce que toutes les tâches actives aient été complétées. Une tâche est considérée terminée lorsque la capacité de calcul dédié à cette tâche (capacité qui est régulièrement réévaluée au fur et à mesure du traitement de la tâche), multiplié par le temps depuis lequel la tâche est "active", est égale à la quantité de calcul réclamé par la tâche en premier lieu, paramètre que l'on fixe par avance dans un des fichiers de scripts tcl. L'ordonnancement qui est demandé à la machine virtuelle qui héberge le coeur, consiste en réalité à indiquer dans combien de temps il faudra venir vérifier de nouveau sur le coeur, où en sont les tâches actives, et éventuellement retirer les tâches finies.

### 3.5 Choix des attaques à implémenter

Les attaques XML-DoS désignent de façon générale l'utilisation du langage XML pour mener une attaque de déni de service. De nombreuses attaques XML-DoS sont théoriquement possibles, mais en pratique seules quelques-unes d'entre elles représentent toujours une menace de sécurité sérieuse. Nous allons en lister deux, après avoir expliqué comment et pourquoi le langage XML permet de telles attaques.

#### 3.5.1 Quelques rappels sur le langage XML

Les informations qui suivent proviennent de [53].

#### Généralités sur le langage XML

Le XML est un langage de description de données (contrairement au HTTP qui sert à l'afficher), qui permet de véhiculer de l'information, peu importe sur quel matériel et avec quel logiciel. C'est un langage de balises, et c'est à l'utilisateur de définir des balises. Les do-

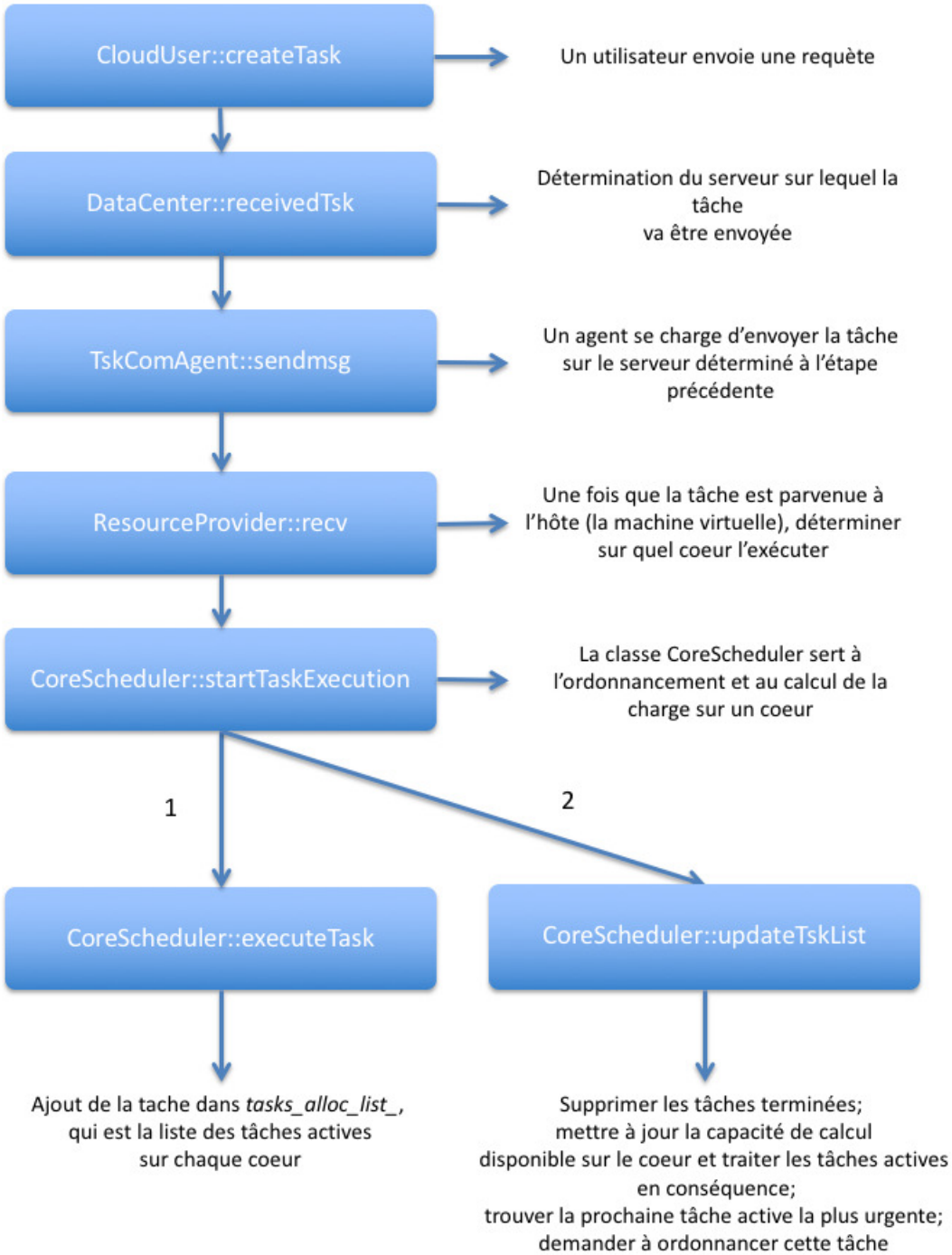


Figure 3.1 Schéma de fonctionnement avant modifications du simulateur GreenCloud

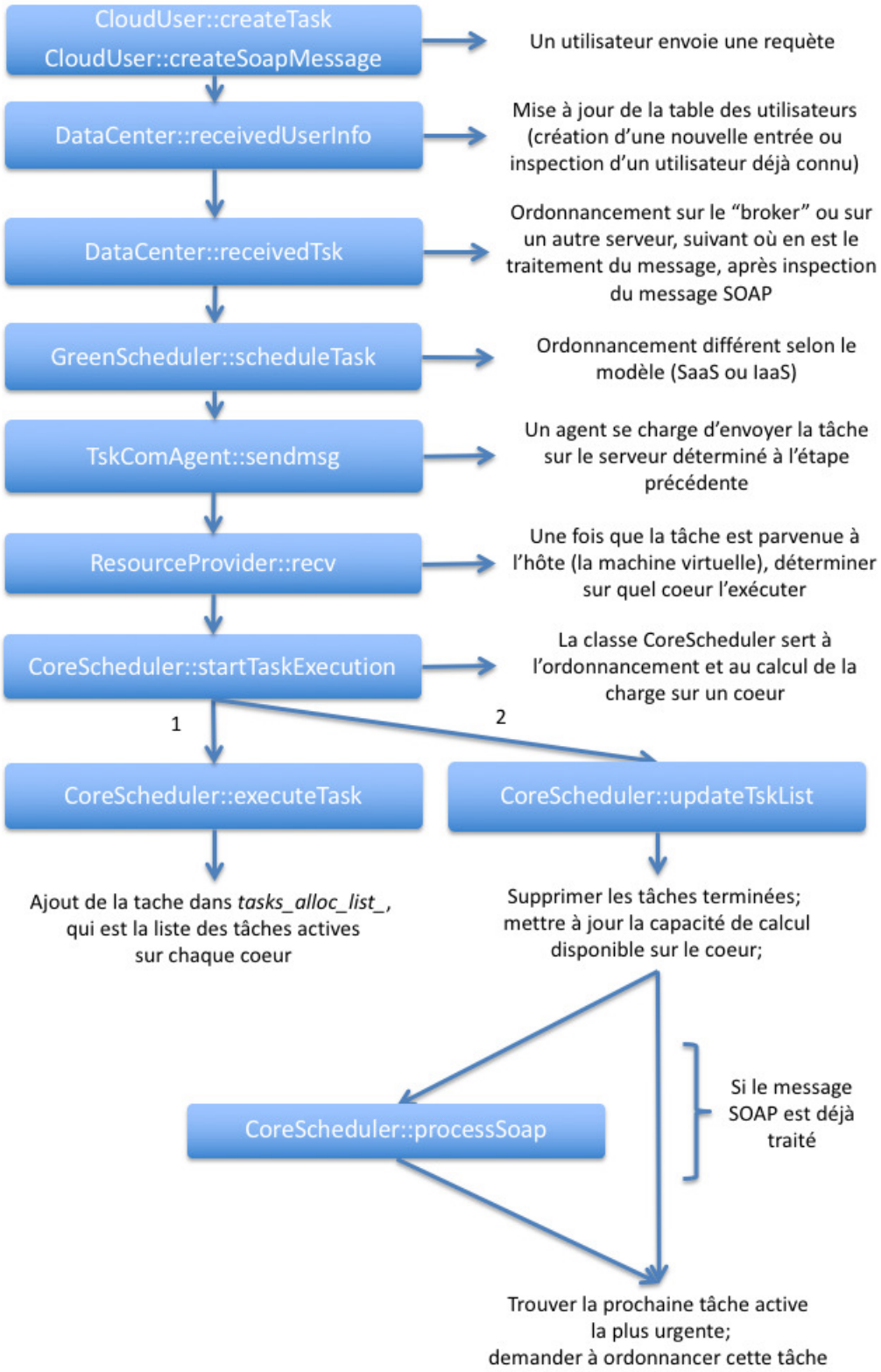


Figure 3.2 Nouveau schéma de fonctionnement après modifications du simulateur GreenCloud

cuments XML forment une structure en arbre, avec notamment un élément racine qui sera le parent de tous les autres éléments. Chaque élément enfant peut lui-même avoir des enfants. Chaque élément peut contenir du texte et/ou des attributs. On distingue un message "bien formé" d'un message "valide" (ce dernier impliquant le premier, mais la réciproque est fausse). Les règles qui définissent les éléments et attributs admissibles pour les documents XML se trouvent dans le "Document Type Definition" (DTD). Un DTD permet par exemple de s'accorder sur un standard pour échanger des données entre différentes personnes, ou pour s'assurer qu'un message reçu de l'extérieur est valide. Un document est donc considéré valide s'il est bien formé et se conforme aux règles définies dans le DTD. Concrètement, un DTD permet de décrire la structure d'un document XML, en listant les éléments rencontrés dans le document ainsi que leur définition. On peut déclarer le DTD soit dans un fichier séparé, soit au sein même du document XML (on parle respectivement de DTD externe et interne).

#### Les différentes façons de traiter les messages XML : le standard DOM

Afin d'accéder et de manipuler les documents XML, il existe par exemple le standard XML DOM [10], qui va présenter le document XML comme un arbre. Avec ce standard, tous les éléments du document XML sont considérés comme des noeuds. Tous les noeuds de l'arbre ont des relations les uns avec les autres, on parle ainsi de noeuds enfants, parents, et frères. Chaque noeud a exactement un noeud parent. Une feuille est un noeud sans enfant. Des noeuds frères ont le même noeud parent. La compilation d'un document XML consiste alors à convertir un document XML en objet XML DOM qu'on pourra ensuite traiter (c'est-à-dire lire, ajouter des noeuds, en supprimer, etc..) avec un langage comme le JavaScript [23].

#### Comparaison des différentes compilations possibles d'un document XML

Au moment de compiler un document XML, on a le choix entre deux techniques : le compilateur DOM et le compilateur SAX. Quand devrait-on utiliser l'un plutôt que l'autre ? (informations tirées de [45])

1. SAX
   SAX (pour Simple API for XML) [46] est un compilateur basé sur événements, c'est-à-dire que chaque élément rencontré dans le message déclenche un événement qu'il faut ensuite prendre en charge. SAX a pour principal avantage d'utiliser beaucoup moins de mémoire que les compilateurs DOM. Notamment, la consommation de mémoire n'est pas proportionnelle à la taille du message à traiter, ce qui le rend

particulièrement bien adapté au cas de très gros messages. SAX permet également d'arrêter le traitement du message à tout moment, ce qui est utile lorsque l'on veut uniquement récupérer une donnée et stopper le traitement une fois que c'est fait. Un autre avantage en lien avec les deux précédents est la possibilité de ne considérer que la partie du document qui nous intéresse. Ainsi, il n'est pas nécessaire de lire tout le document pour accéder à une zone d'intérêt en particulier, ce qui là encore économise les ressources du système. En conclusion, SAX est plus approprié pour les gros documents, ou lorsque l'on souhaite économiser les ressources du système. Mais bien que ce soit la recommandation, il est très probable que de nombreux services web utilisent toujours la méthode DOM alors que SAX serait plus approprié, soit par ignorance soit par négligence, mais également peut-être, car la méthode DOM n'est pas totalement à écarter dans certains cas.

2. DOM
   Avec les compilateurs DOM [10], il n'y a pas d'événements déclenchés lors de la compilation. Le document XML est compilé dans son ensemble et l'arbre correspondant est créé. Quels sont les avantages de cette méthode ? Cette méthode est bien appropriée lorsque l'on souhaite pouvoir accéder aléatoirement aux données. En effet, contrairement à SAX où il faudrait prendre en charge les données dans l'ordre où elles sont compilées, DOM stocke l'entièreté de l'arbre en mémoire. Par ailleurs, SAX est davantage orienté sur la lecture du document, mais n'est pas vraiment efficace sur l'écriture, contrairement à DOM qui permet de créer ou de modifier un document en mémoire.

Comme l'utilisation d'un compilateur DOM rend une attaque XML-DoS sur un serveur web possible, voire très facile (en laissant l'arbre XML grandir en mémoire, éventuellement de façon exponentielle), c'est cette technique que nous avons choisi d'implémenter par la suite dans notre simulateur.

L'objectif de l'attaque est de produire un message XML qui serait valide selon les règles définies par le schéma XML, mais qui ferait planter le programme cherchant à le lire. Nous avons pris comme références pour ces attaques celles indiquées comme les plus dommageables dans [48]. Ces attaques sont également classées dans les plus à risque dans [50]. Elles font partie de la catégorie des "bombes XML". Deux exemples sont proposés (toutes les variantes ne sont pas efficaces ni même valides du point de vue du XML) : l'attaque par expansion

exponentielle d'entités (Exponential Entity Expansion attack), et l'attaque par explosion quadratique (Quadratic Blowup attack).

### 3.5.2 Attaque n°1 : Exponential Entity Expansion attack ou "coercive parsing"

L'attaque par expansion exponentielle d'entités donne au message XML l'allure donnée en figure 3.3 [48].

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Figure 3.3 Exponential Entity Expansion attack

Le standard autorise à inclure un DTD (Document Type Definition) au sein même du document XML. Dans ce DTD, nous pouvons définir nos propres entités (ici lol, lol2, lol3, etc.), en remplacement d'un string qui est indiqué entre guillemets. Le DTD offre également la possibilité de faire s'emboiter les entités les unes dans les autres (ici lol9 fait référence à lol8 qui fait référence à lol7, etc..). Tout est donc parfaitement valide dans ce message du point de vue du standard XML. Le corps du message dit ensuite que l'on a affaire à un noeud de type "lolz", qui contient un texte défini dans le DTD (le lol9). L'attaque peut alors commencer, puisque cette entité est définie récursivement jusqu'à l'entité lol. Pour arriver là, il faudra parcourir tout le DTD, notamment étendre chacune des entités (lol8, lol7, etc..) en ses 10 entités "filles", comportant elles aussi 10 entités filles, et ce jusqu'à ce qu'on atteigne la définition de lol. Dans notre cas, cela revient à 1 milliard de "lol", soit environ 3GB de mémoire, avec un message qui en faisait à peine 1KB. Ce type d'attaque est également connu sous le nom de "coercive parsing", qui désigne plus généralement le fait de construire un message XML avec un nombre illimité de tags ouvrants dans le corps du message SOAP [52].

### 3.5.3 Attaque n°2 : Quadratic Blowup attack

Cette attaque prend la forme donnée figure 3.4 [48].

```
<?xml version="1.0"?>
<!DOCTYPE kaboom [
  <!ENTITY a "aaaaaaaaaaaaaaaaaa...">
]>
<kaboom>&a;&a;&a;&a;&a;&a;&a;&a;&a;...</kaboom>
```

Figure 3.4 Quadratic blowup attack

Cette fois-ci, le problème est pris dans l'autre sens : au lieu de définir de nombreuses entités emboitées et de petite taille, une très grosse entité est définie et référencée de nombreuses fois. Cela a par exemple pour avantage de passer les contrôles empêchant les messages avec trop d'entités emboitées les unes dans les autres. Avec une entité définie comme étant égale à 50000 caractères, et en y référant 50000 fois, 200KB de message permettent d'atteindre une utilisation mémoire de 2.5GB.

### 3.5.4 Intérêt de ces attaques dans le cadre de notre étude

Ce genre d'attaque (Quadratic Blowup attack ou Exponential Entity Expansion attack) vise également à saturer le CPU, qui atteint facilement 100% de manière prolongée pour ce genre d'attaque [50]. Pour tester l'efficacité du coercive parsing, Vissers et al. [50] ont envoyé 100 messages de 210KB chaque avec un contenu similaire à celui montré figure 3.3. Cela a demandé au serveur web hébergé sur une machine dont le CPU possède 4 coeurs à 2 GHZ et 16 GB de RAM, une minute au complet d'utilisation à 100% du CPU pour traiter toutes les requêtes, et finalement retourner un message de "stack overflow".

Ces deux attaques apparaissent ainsi comme des candidats idéaux pour tester les effets d'une attaque XML-DoS dans le cadre d'un réseau d'infonuagique.

## 3.6 Ajouts et modifications au simulateur GreenCloud

Le simulateur GreenCloud n'est pas prévu à la base pour étudier les attaques de déni de service, que ce soit en termes de configuration de l'infrastructure (services web, "broker", etc..), de suivi des paramètres d'intérêt lors de l'attaque (CPU, mémoire, charge individuelle

des serveurs), ou encore de ce qui fait certaines spécificités des réseaux d'infonuagique (choix du nombre de machines virtuelles par serveur, compétition pour les ressources entre ces machines virtuelles). Il a donc fallu procéder à de nombreux ajustements, voire à de profonds remaniements, en ajoutant de nouvelles fonctionnalités, en en modifiant certaines autres, afin de rendre le simulateur adapté à notre étude. Après avoir formulé quelques hypothèses sur les attaques, nous détaillerons les changements apportés.

### 3.6.1 Hypothèses sur les attaques

#### L'attaque en elle-même

Une requête destinée au réseau d'infonuagique va être constituée de trois parties.

1. Traitement d'une première requête SOAP par le "broker"
2. Traitement d'une deuxième requête SOAP par le serveur sélectionné pour exécuter la tâche
3. Traitement de la tâche (c'est cette tâche qui était envoyée initialement)

Il y a la tâche que l'utilisateur veut voir traitée par le réseau (c'est cette requête qui est présente par défaut dans le simulateur). A cette tâche, nous avons ajouté deux messages SOAP. Un est destiné au "broker", l'autre est destiné au serveur qui traitera la tâche. Ces messages SOAP ont pour rôle de tenir compte du fait que le réseau d'infonuagique (par l'intermédiaire du "broker" puis du serveur qui traite la tâche) offre ses services au travers de services web. Les scripts tcl permettent de définir lequel de ces messages sera légitime ou au contraire malveillant (l'attaque peut donc porter soit sur le "broker", soit sur le serveur, soit sur ces deux entités, selon le scénario de simulation choisi). L'attaque peut être de type "Exponential Entity Expansion attack" ou "Quadratic Blowup attack". Dans tous les cas, la requête intrinsèque, celle qui sera traitée par un serveur autre que le "broker", est inchangée (elle est donc identique que la requête soit légitime ou malveillante). Le pourcentage de requêtes malveillantes dans l'ensemble des messages envoyés peut être configuré depuis les fichiers de scripts décrivant les utilisateurs. Le but de l'attaque consiste à faire une utilisation abusive des ressources de calcul et de mémoire de la machine qui va traiter le message porteur de l'attaque. Le traitement du message doit se faire en deux temps : traiter le DTD, puis traiter le corps du message en utilisant l'information extraite du DTD. Le traitement du DTD va consister à stocker dans des structures de données toutes les entités définies, ainsi que les relations entre elles. Le traitement du corps du message doit quant à lui permettre d'étendre en mémoire l'intégralité du contenu du message, en adoptant une démarche de type "DOM parser". Le graphe résultant comportera quelques noeuds voire quelques dizaines de

noeuds dans le cas d'un message légitime, mais plusieurs milliers voire millions dans le cas d'un message malveillant. Cette construction de graphe doit alors trouver des répercussions tant du point de vue de l'utilisation CPU que de l'utilisation de la mémoire.

Les attaques décrites ci-dessous appartiennent au scénario "external to internal" décrit dans l'article, c'est-à-dire que l'origine de l'attaque est située à l'extérieur du réseau, et vise le réseau lui-même.

### L'attaque sur le "broker"

Le serveur qui va jouer le rôle de "broker" peut être constitué d'une ou deux machines virtuelles, ne possédant chacune qu'un CPU monocoeur à des fins de simplification. Toutes les requêtes reçues au niveau du centre de données sont d'abord envoyées sur ce "broker". Ce n'est qu'une fois le traitement achevé sur le "broker" qu'elles sont ordonnancées sur les autres machines du réseau. Le réseau garde dans une liste l'ensemble des tâches reçues et envoyées au "broker", et la consulte régulièrement pour en extraire les tâches qui ont été traitées par le "broker" et qui sont de ce fait prêtes pour la suite du traitement (envoi sur un serveur pour traiter la tâche proprement dite). Parmi toutes les tâches qui sont prêtes pour la suite du traitement, celle dont la "deadline" est la plus proche est choisie, et celles dont la "deadline" est passée sont supprimées, et seront considérées comme échouées.

— **Critère de succès du traitement du message SOAP par le "broker"**
Le traitement effectué par le "broker" simule celui d'un service web qui reçoit un message SOAP et essaie d'en extraire l'information. Le simple fait de réussir à extraire l'information contenue dans le message SOAP suffit à considérer que le "broker" a accompli sa tâche avec succès, et la requête peut donc continuer à être traitée sur un autre serveur.

— **Critère de succès de l'attaque sur le "broker"**
L'attaque est considérée réussie lorsque le "broker" est rendu indisponible, c'est-à-dire qu'il ne peut plus remplir son rôle et rend de ce fait impossible pour un utilisateur d'accéder aux ressources du réseau. Deux éléments vont empêcher de nouveaux messages d'être traité sur le "broker" : une saturation de la mémoire, et lorsque le nombre maximum de tâches simultanées que le "broker" peut traiter est atteint. Bien sûr, le premier point entraine le second, puisque les requêtes n'ayant plus les ressources nécessaires pour être complétées, elles vont stagner dans la liste des tâches actives du "broker", et empêcher de nouvelles requêtes d'y entrer.

**L'attaque sur une machine virtuelle**

Afin de rendre compte de l'attaque sur une machine virtuelle, et notamment de l'impact de l'attaque sur les autres machines virtuelles hébergées sur la même machine physique, nous avons décidé de placer deux machines virtuelles sur chaque serveur. Comme dans le cas du "broker", nous cherchons ici à saturer une machine virtuelle tant en termes de CPU que de mémoire.

— **Critère de succès de l'attaque sur la machine virtuelle**
L'attaque doit rendre indisponibles pour les autres utilisateurs les ressources de calcul et de mémoire, non seulement sur la machine virtuelle attaquée, mais également sur les machines virtuelles avec lesquelles elle cohabite.

### 3.6.2 Modifications réalisées sur le simulateur

Les modifications concernent à la fois les fichiers de script (.tcl, décrivant les scénarios de simulation) et les fichiers sources (.cpp, décrivant le fonctionnement intrinsèque du réseau).

Les modifications aux fichiers tcl ont principalement pour but d'enrichir les scénarios déjà proposés. Voici la liste des fichiers modifiés.

1. Fichier dc.tcl
   Alors que de base, GreenCloud ne supportait qu'une machine virtuelle par serveur, ajout de la possibilité d'avoir un nombre arbitraire de machines virtuelles par serveur (la seule limitation est causée par l'écriture "en dur" des spécifications des serveurs, qui doivent donc être adaptées lorsque l'on souhaite davantage de machines virtuelles. Par exemple, augmenter le nombre de coeurs, la taille de la mémoire, etc.). Un identifiant est associé à chaque machine virtuelle.

2. Fichier finish.tcl
   Distinction entre le "broker" et les autres serveurs au moment d'afficher les statistiques en fin de simulation.

3. Fichier user.tcl
   Distinction entre utilisateurs malveillants et légitimes en termes de profil de traffic généré et de type de messages envoyé. Ajout de paramètres permettant de définir si l'attaque portera sur le "broker" ou bien sur une machine virtuelle quelconque du réseau d'infonuagique. Ajout d'un paramètre dictant si le réseau se comportera comme

un "SaaS" ou un "IaaS", c'est-à-dire (pour simplifier dans notre cas) si un utilisateur se voit attribuer toujours la même machine virtuelle pour toutes les tâches qu'il soumet au réseau ("IaaS", l'utilisateur est alors propriétaire en quelque sorte d'une machine virtuelle qui lui a été attribuée à sa première connection) ou au contraire si sa tâche va à n'importe quelle machine virtuelle libre ("SaaS", la tâche de l'utilisateur est traitée par un serveur web situé sur une machine virtuelle quelconque que l'utilisateur ne possède pas).

La figure 3.5 présente un résumé des ajouts et modifications apportées.

Pour chaque méthode ajoutée et/ou modifiée, on indiquera la nature des ajouts/changements, et le contexte dans lequel ils s'opèrent.

1. Dans la classe 'CloudTask'
   Cette classe définit les caractéristiques des messages qui seront envoyés sur le réseau. Comme il s'agit désormais de rendre compte du traitement de messages SOAP par les machines du réseau, on associe à chaque 'cloudTask' deux messages SOAP : un qui est destiné au "broker", un autre destiné au serveur sur lequel la tâche sera traitée. Une méthode permet de retourner le message SOAP suivant le contexte (suivant que l'on est sur le "broker" ou non, cette méthode retournera le message SOAP approprié). Un booléen est ajouté pour savoir si ces deux messages SOAP ont été traités ou non.
2. Dans la classe 'ResourceProvider'
   Cette classe est l'abstraction des machines à la fois physiques et virtuelles du réseau. En effet, les tâches sont allouées sur les machines virtuelles, qui sont elles-mêmes allouées sur les serveurs physiques. C'est dans la classe "ResourceProvider" que se font ces allocations. C'est également dans cette classe que sont reçues les notifications de l'ordonnanceur pour exécuter les tâches actives sur la machine virtuelle en question, et qu'est calculée l'utilisation des ressources (CPU, mémoire, réseau, etc..). Voici les deux méthodes qui ont dû être modifiées :

   — Modification de la méthode 'trySchedulingTsk'
   C'est l'ordonnanceur qui va appeler cette méthode, pour savoir sur quel serveur il est possible d'ordonnancer une tâche. Nous distinguons alors le cas où la requête arrive pour la première fois sur le réseau, auquel cas il faut vérifier que le "broker" est capable d'accueillir la tâche, et le cas où le "broker" a déjà traité sa partie de la requête (et doit donc passer la main à un autre serveur). Cette méthode permet alors de savoir quel serveur choisir ("broker" ou non). Pour chaque serveur, déterminer s'il peut être candidat ou non pour accueillir la tâche prend la forme

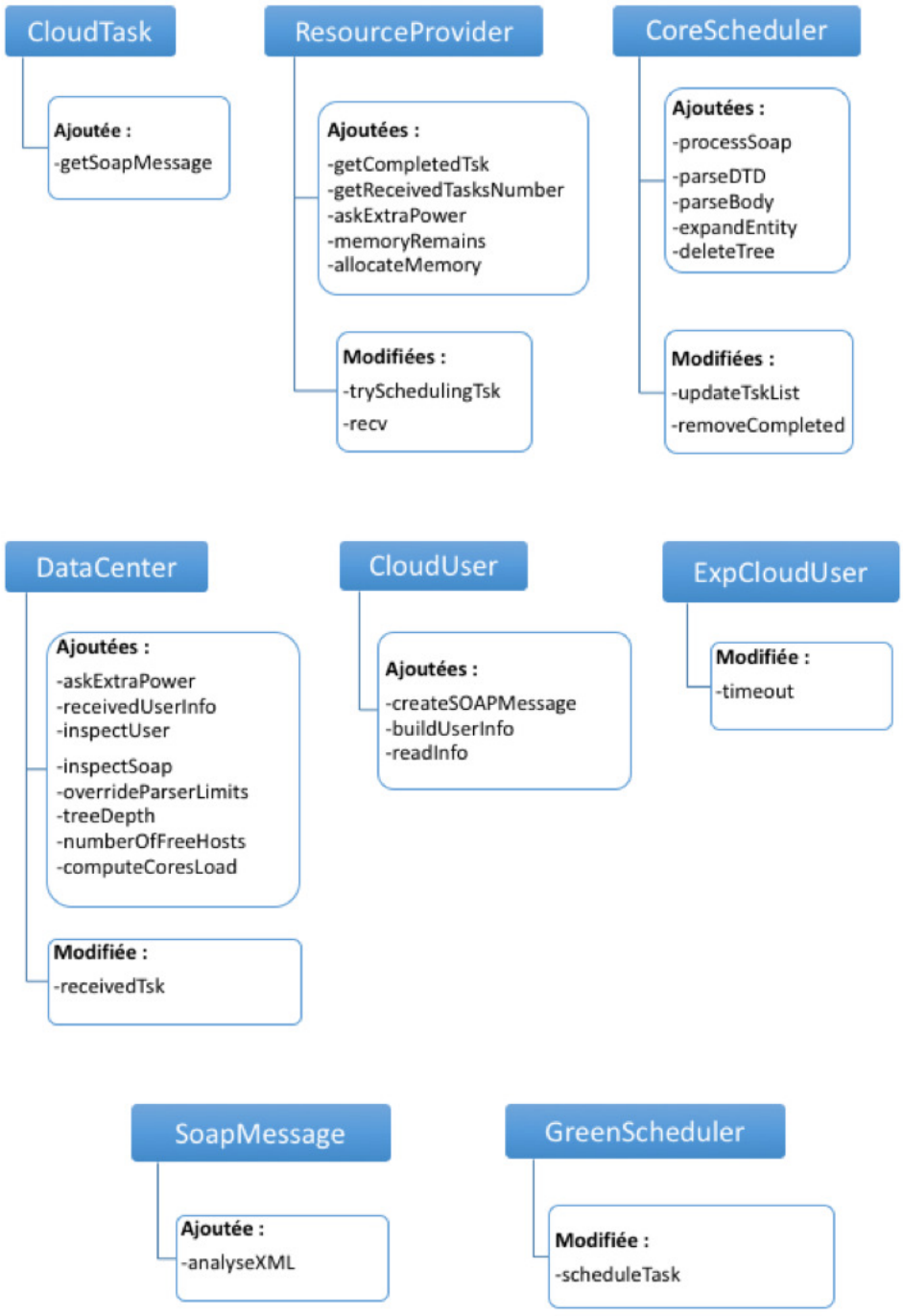


Figure 3.5 Schéma des ajouts et modifications au simulateur GreenCloud

suivante. Si la méthode est appelée sur le "broker", et que le message SOAP qui lui est destiné n'est pas encore ordonnancé, on commence par vérifier qu'il reste de la mémoire disponible, puis pour chaque coeur du "broker", que le nombre de tâches actives est inférieur à une limite que l'on s'est fixée. Si toutes les vérifications sont passées avec succès, on assigne la tâche à un des coeurs du "broker", et le message est marqué comme étant ordonnancé. Dans le cas contraire, le message est marqué comme échoué. Si la méthode est appelée sur un serveur autre que le "broker", il faut prendre en compte à la fois la tâche proprement dite, ainsi que le message SOAP qui est destiné à ce serveur. Pour cela, la vitesse à laquelle la tâche doit être exécutée est calculée, en fonction du coût CPU statique de la tâche (défini dans les fichiers de scripts), et du temps restant pour traiter la tâche, temps qui est égal à la différence entre l'échéance de la tâche et le temps courant dans le simulateur. Pour chaque coeur de la machine virtuelle, si cette vitesse nécessaire pour traiter la tâche est possible avec les ressources de calcul restantes, s'il reste de la mémoire disponible et que la liste des tâches actives n'est pas pleine, la tâche peut être assignée au coeur correspondant (la tâche passe dans la liste des tâches assignées de ce coeur). Auquel cas la tâche ainsi que le message SOAP sont marqués comme étant ordonnancés, sinon ils sont considérés comme échoués. Aucun coût CPU n'est associé au traitement du message SOAP, puisqu'on ne peut le prévoir à l'avance, mais est très faible dans le cas d'un message légitime (le message est supposé légitime par défaut).

— Modification de la méthode 'recv'
Cette méthode est appelée après le 'trySchedulingTsk', c'est-à-dire une fois qu'un serveur a été trouvé pour traiter la tâche, et va permettre de lancer l'exécution de la tâche qui a été fournie en paramètre (c'est-à-dire que la tâche doit être transférée de la liste des tâches assignées à la liste de tâches actives du coeur en charge de cette tâche). Dans cette méthode, une vérification va permettre de déterminer s'il est effectivement possible d'allouer la tâche sur un des coeurs du serveur (vérification supplémentaire par rapport à celle effectuée dans la méthode 'trySchedulingTsk'), en utilisant pour cela une méthode dédiée (la méthode 'tryToAllocate'). Cette vérification est élargie pour que le test sur la possible allocation de la tâche sur le serveur avec la méthode 'tryToAllocate' se fasse sur les serveurs autres que le "broker". Sur le "broker", le simple fait qu'il reste de la mémoire disponible suffit à y allouer la tâche. En effet, pour un message SOAP légitime, la consommation CPU est presque négligeable, et

il faut seulement disposer d'un peu de mémoire. Si aucune contremesure contre les attaques XML-DoS n'est implémentée, on peut donc étudier avec ce scénario l'effet de ces attaques lorsque rien n'est prévu pour les arrêter.

— Ajout de la méthode 'getCompletedTsk'
Interroge l'ensemble des coeurs de tous les CPU de la machine virtuelle, afin d'en récupérer le nombre de tâches qui ont été terminées avec succès (cette méthode sera utilisée plus tard pour faire des statistiques en l'absence et en présence d'une attaque XML-DoS).

— Ajout de la méthode 'getReceivedTasksNumber'
Détermine le nombre de taches qui a été reçu sur chaque coeur du 'ResourceProvider'. Là aussi, cette méthode servira au moment d'établir des statistiques.

— Ajout de la méthode 'askExtraPower'
'askExtraPower' permet à une machine virtuelle de demander une partie des ressources de calcul normalement réservées à une autre machine virtuelle sur la même machine physique. Cette méthode est appelée au moment du traitement d'un message SOAP, au cas où ce traitement serait anormalement long (cas d'une attaque XML-DoS), et simule la contention pour les ressources qui peut exister entre machines virtuelles situées sur une même machine physique. Lors d'une attaque XML-DoS, la machine qui ne sera pas victime de l'attaque verra un ralentissement du traitement de ses tâches.

— Ajout de la méthode 'memoryRemains'
Renvoie un booléen indiquant si la mémoire restant sur cette machine est au-dessus ou non d'un certain seuil. Cette méthode est utilisée lors de l'allocation d'une tâche ou avant d'envoyer un message SOAP sur le "broker" par exemple.

— Ajout de la méthode 'allocateMemory'
Appelée depuis la classe 'CoreScheduler', cette méthode simule les besoins en mémoire de la compilation "DOM" d'un message SOAP. Il n'est plus possible d'allouer de la mémoire pour un message SOAP lorsque la mémoire restante passe en dessous d'un seuil qui a été prédéfini.

3. Dans la classe 'CoreScheduler'

Cette classe effectue le traitement du message SOAP à proprement parler. Elle est l'abstraction d'un des coeurs du processeur. Cette classe va recevoir un objet de type 'SOAPMessage', et va devoir construire le DTD (c'est-à-dire créer une structure de données associant à chaque entité sa définition en termes de chaines de caractères ou de références à d'autres entités), puis construire le graphe du message SOAP à l'aide du corps du message et des relations entre entités issues du DTD. Ceci donnera lieu à la création de noeuds qui sont des structures de données définies dans la classe 'SOAPMessage'.

— Modification de la méthode 'updateTskList'
  Cette méthode est appelée régulièrement et permet de mettre à jour la liste des tâches actives du coeur. Il s'agit de supprimer les tâches terminées, ainsi que de mettre à jour la capacité de calcul qui va être rendue disponible pour traiter les tâches restantes. Dans la version originelle de cette méthode, toutes les tâches actives étaient ensuite parcourues pour trouver celle qui avait la "deadline" la plus proche, cette tâche étant alors envoyée à l'ordonnanceur. Nous avons modifié ce comportement de la façon suivante. Pour chacune des tâches de la liste active, s'il reste de la mémoire disponible sur la machine à laquelle le coeur appartient, on récupère le message SOAP associé à la tâche (et qui sera donc différent selon que l'on est sur le "broker" ou non) et on exécute la méthode permettant de traiter un message SOAP (voir ci-dessous la méthode 'processSoap'). Si le message a été traité avec succès et que l'on est sur le "broker", on supprime cette tâche de la liste des tâches actives, et on envoie une demande de mise à jour au "centre de contrôle" c'est-à-dire au "DataCenter", qui reçoit les requêtes. Il sera ainsi mis au courant que le "broker" vient de traiter un message et pourra donc transférer la tâche correspondante à un serveur afin qu'elle soit exécutée.

— Ajout de la méthode 'processSoap'
  Dans cette méthode, le message SOAP est traité en deux étapes, avec l'analyse du DTD (via l'appel à 'parseDTD'), puis du corps du message (via l'appel à 'parseBody'). Un booléen est retourné pour chacune de ces deux fonctions, et si ces deux appels se sont exécutés avec succès, le message SOAP est marqué comme terminé avec succès, sinon il est marqué comme terminé et échoué. Avant d'effectuer le premier traitement, on enregistre le temps courant dans le simulateur afin de pouvoir calculer à tout moment le temps que prend le traitement du message SOAP. À la fin du traitement, le temps qui a été nécessaire pour traiter

le message (c'est-à-dire le temps pour traiter intégralement le message dans le cas d'un message légitime, et le temps au bout duquel le traitement s'arrête faute de mémoire dans le cas d'un message illégitime) est enregistré, et envoyé au "DataCenter" afin de mettre à jour l'utilisateur associé dans la base de données (en le bloquant éventuellement si ce temps de traitement est jugé trop long).

— Ajout de la méthode 'parseDTD'
Cette méthode reçoit en argument un message SOAP, qui a été préalablement analysé pour en extraire les zones d'intérêt (notamment supprimer la plupart des marques de formatage pour ne garder que la partie "utile" du message, c'est-à-dire le DTD et le corps du message à proprement parler. Elle permet d'analyser le DTD, en stockant dans un vecteur les entités (une entité est un noeud du graphe XML) définies dans le DTD, sous la forme du nom de l'entité suivi de sa définition en termes de chaines de caractères ou de références à d'autres entités (création d'une structure de données contenant toutes ces informations pour chaque entité rencontrée). Cette étape devrait se dérouler avec succès que le message soit malveillant ou non, car il ne s'agit que de faire une lecture et une traduction en terme de structure de données, sans avoir à interpréter le sens du message.

— Ajout de la méthode 'parseBody'
Cette méthode commence par analyser le contenu du corps du message. Pour chaque élément du corps du message, la structure de données précédemment créée est invoquée pour créer une représentation du message sous forme de graphe, chaque élément du corps est alors la racine d'un arbre. La méthode 'expandEntity' est utilisée pour développer chaque élément en ses constituants, selon leur définition dans le DTD.

— Ajout de la méthode 'expandEntity'
La méthode 'expandEntity' est récursive, car chaque noeud créé par cette méthode peut lui-même créer d'autres noeuds (s'il est à la racine d'un arbre), et modélise l'expansion en mémoire de l'arbre XML à partir de son noeud racine. Au début de la méthode, la date courante est comparée à la date de début de traitement. Si l'écart entre ces deux dates dépasse un certain seuil, il s'opère un transfert des ressources CPU depuis l'autre VM vers la VM effectuant le traitement (simulation de partage CPU entre deux VM situées sur la même machine physique). Ensuite, cette méthode crée dynamiquement un noeud, lui donne un identifiant, met à

jour son noeud parent (qui est donné en paramètre de la fonction), sa donnée, et son nombre d'enfants, tout cela grâce à la structure définie dans 'parseDTD'. Le recours à l'allocation dynamique s'explique par le fait que les besoins en mémoire ne peuvent être déterminés à la compilation, mais bien au moment de l'exécution, par le biais des fichiers de script tcl (la taille des arbres XML créés dépend des paramètres fixés dans ces fichiers de scripts). Pour chaque enfant qui est également identifié comme étant un noeud (c'est-à-dire une structure avec noeuds enfants et noeuds parents, et non une simple chaine de caractères), on appelle de nouveau 'expandEntity' (appel récursif). Afin d'accélérer nos simulations, nous avons appliqué un facteur d'échelle en terme de mémoire utilisée (un message prend 50 fois plus de place en mémoire que sa taille réelle). Cette implémentation ne pose pas vraiment problème, car elle rend seulement plus rapide une saturation de la mémoire qui aurait eu lieu quoiqu'il arrive. Pour que la simulation reste cohérente, ce sont tous les messages qui prennent 50 fois plus de place en mémoire que la normale, et pas seulement les messages servant à l'attaque. Il est ainsi possible de comparer l'évolution de l'utilisation de la mémoire dans le cas avec et sans attaque.

— Ajout de la méthode 'deleteTree'
Permet la suppression de tous les objets alloués dynamiquement lors la création du message SOAP en mémoire. Cette méthode est à distinguer de la libération de la mémoire pour les serveurs de la simulation. Ici, la mémoire libérée est celle de la machine effectuant la simulation. Cela permet d'éviter les fuites de mémoire, et également de ne pas rendre la machine sur laquelle est faite la simulation d'être elle-même victime de l'attaque simulée. Cette méthode est appelée lorsque le traitement du message est terminé, donc aussi bien si le message a pu être lu jusqu'au bout ou si le traitement s'est arrêté pour cause de ressources insuffisantes, et qu'il est donc possible de libérer la mémoire associée. Pour libérer la mémoire associée à un arbre, la méthode est appelée avec le noeud racine comme paramètre. Puis avec un appel récursif, va descendre jusqu'aux feuilles de l'arbre, supprimer ces feuilles, et remonter progressivement pour supprimer un à un tous les noeuds jusqu'à la racine.

— Modification de la méthode 'removeCompleted'
Grâce à cette méthode, les tâches terminées ou échouées sont supprimées de la liste des tâches actives. Auparavant, la méthode ne considérait que les tâches qui ne nécessitaient plus aucun calcul (ces tâches étaient donc considérées comme

terminées), et les tâches dont la "deadline" était dépassée (celles-ci étaient alors considérées comme échouées). Nous avons ajouté un nouveau critère de succès, à savoir que sur le "broker", une tâche dont le message SOAP a été traité est également considérée comme terminée, et à ce titre doit être retiré de la liste des tâches actives du coeur. Lorsque la tâche est terminée, on libère dorénavant la mémoire qui a été utilisée par les serveurs du réseau pour analyser le message SOAP. Si c'est la requête de l'utilisateur à proprement parler qui est terminée (et non le message SOAP préliminaire destiné au "broker"), on envoie la sortie de la tâche à la machine hôte sur laquelle la tâche a été allouée.

4. Dans la classe 'DataCenter'
   Cette classe sert de porte d'entrée dans le réseau, et va agir comme un 'centre de contrôle'. Une nouvelle tâche reçue de l'extérieur y est reçue pour ensuite être expédiée sur un serveur. C'est également dans cette classe que l'on va choisir l'ordonnanceur (suivant ce qui a été choisi dans les fichiers de scripts), et une des méthodes permet de calculer régulièrement la charge globale qui pèse sur le réseau (CPU, mémoire, bande passante, etc..).

   — Ajout de la méthode 'askExtraPower'
   Cette méthode est nécessaire, car lorsqu' une des machines virtuelles a besoin de plus de ressources de calcul, et cherche à en prélever depuis les autres machines virtuelles qui coexistent sur la même machine physique, il faut nécessairement remonter au niveau du "dataCenter" afin de savoir avec quelles machines virtuelles la machine à l'origine de la requête partage ses ressources. Connaissant le nombre de machines virtuelles par serveur (dans notre cas, nous avons choisi d'utiliser deux machines virtuelles par serveur), et le numéro de la machine virtuelle (chaque VM a un numéro d'identification unique, qui lui est donné au moment de sa création, par exemple sur la machine physique $n_1$, il y aura les machines virtuelles $n_1$ et $n_2$, le conflit d'identifiant n'étant pas gênant car en pratique l'identifiant de la machine physique n'est jamais utilisé), on peut en déduire quelles seront les machines virtuelles qui cohabitent avec la VM demandant le surplus de ressources.

   — Ajout de la méthode 'receivedUserInfo'
   Cette méthode intercepte en premier la requête d'un utilisateur. La classe 'DataCenter' possède une table contenant des informations à propos des utilisateurs qui envoient des requêtes (identifiant, mot de passe, adresse IP, fréquence d'envoi,

date de dernier envoi, etc.). La méthode 'receivedUserInfo' va mettre à jour cette table pour un utilisateur déjà connu du réseau, ou va créer une nouvelle entrée pour un utilisateur qui enverrait sa première requête. Un nouvel utilisateur est considéré légitime a priori, et pour un utilisateur déjà présent dans la table, la méthode 'inspectUser' va vérifier si l'utilisateur est autorisé à envoyer de nouvelles requêtes étant donné son comportement passé. Si l'utilisateur est jugé légitime, la requête est transmise à l'étape suivante (méthode 'receivedTsk').

— Ajout de la méthode 'inspectUser'
Cette méthode reçoit une entrée de la table contenant les informations sur chaque utilisateur, et va s'assurer que l'utilisateur ne fait pas une utilisation abusive du réseau. Il faut donc que la dernière requête ait été envoyée il y a suffisamment longtemps par rapport à la requête qui vient d'être reçue, qu'en moyenne, le nombre de requêtes reçues en fonction du temps ne dépasse pas un certain seuil, et enfin que les messages SOAP de cet utilisateur n'aient pas été signalés jusqu'à présent comme particulièrement demandant en ressources. Si une de ces vérifications échoue, l'utilisateur est bloqué. Cependant, l'utilisateur va être automatiquement débloqué lorsqu'une certaine durée s'est écoulée (l'utilisateur est bloqué pendant un temps limité).

— Modification de la méthode 'receivedTsk'
Dans cette méthode, le traitement est différent selon que la tâche reçue est destinée au "broker", ou qu'elle est déjà passée par le "broker" et est donc destinée à un autre serveur. Si elle doit passer par le "broker" (ce qui est dénoté par un booléen qui atteste qu'un message a, ou non, été traité par le "broker"), le message SOAP est vérifié (voir méthode 'inspectSoap'), avant d'étudier la possibilité d'ordonnancer la tâche sur le "broker". Si aucune machine virtuelle du "broker" n'est apte à recevoir la tâche, celle-ci est marquée comme échouée, sinon elle est envoyée sur la machine virtuelle correspondante. Si la tâche reçue à ce stade a déjà été traitée par le "broker", elle est ordonnancée sur un des serveurs autres que le "broker", là encore après que le message SOAP correspondant (celui destiné au serveur et non plus au "broker") ait été vérifié.

— Ajout de la méthode 'inspectSoap'
Cette méthode a pour vocation d'appeler une série de mécanismes destinés à valider le contenu d'un message SOAP. Pour l'instant, un unique mécanisme a été

implémenté, consistant comme énoncé précédemment à effectuer une validation de schéma (réalisée par la méthode 'overrideParserLimits'). Pour améliorer le système de défense, d'autres mécanismes pourraient venir s'ajouter à cette liste.

— Ajout de la méthode 'overrideParserLimits'
Cette méthode reçoit un message SOAP, et analyse son DTD afin d'y trouver d'éventuelles irrégularités. Il faut d'abord s'assurer que chaque élément XML n'a pas plus d'attributs qu'il est permis d'en avoir (selon une valeur prédéterminée). Au cas où cette condition ne serait pas respectée, un log d'erreur est enregistré dans un champ prévu à cet effet dans le message SOAP, et la fonction retourne avec une erreur. Ensuite, si la première vérification est passée, la 'profondeur' de chaque élément XML est considérée, autrement dit son nombre d'enfants dans le graphe du document XML. Il est important d'effectuer cette étape après la première, car ainsi il n'y a pas de risque de saturer les ressources en analysant le graphe (auquel cas la vérification serait rendue inutile, car conduisant elle-même à l'attaque). Enfin, la dernière vérification permet de s'assurer que la taille de chacun des noeuds du message ainsi que la taille globale du message lui-même ne sont pas trop grandes, pour ne pas saturer la mémoire de la machine qui traitera le message.

— Ajout de la méthode 'treeDepth'
Cette fonction est récursive et est utilisée dans la vérification de la profondeur de chaque élément XML défini dans le DTD (autrement dis la taille de l'arbre qui a pour racine le noeud en question). Cette fonction est une adaptation de la méthode utilisée pour déterminer la taille d'un arbre binaire. Pour trouver la taille d'un arbre binaire de façon récursive, il s'agit de calculer la taille du sous-arbre droit et celle du sous-arbre gauche, puis de donner au noeud racine la valeur maximale entre ces deux tailles, plus 1. Dans notre cas, chaque noeud a un nombre arbitraire de sous-arbres, donc on ne peut plus comparer directement les sous-arbres gauches et droits. Nous avons donc élargi les possibilités de cette fonction en calculant la taille de chaque sous-arbre, en stockant le résultat dans un vecteur (structure de données dont la taille peut varier à l'exécution, ce qui est nécessaire car on ne connait pas à l'avance le contenu des messages SOAP), puis en recherchant le maximum de toutes les valeurs stockées dans ce vecteur. Ce maximum plus un donne la taille au niveau du noeud pour lequel la fonction est appelée (la fonction est récursive).

— Ajout de la méthode 'numberOfFreeHosts'
Cette méthode renvoie le nombre d'hôtes à qui il reste de la mémoire disponible pour recevoir de nouvelles tâches. Elle est notamment utilisée au moment de déterminer s'il est possible de trouver un hôte pour exécuter la tâche dans la méthode 'receivedTsk'.

— Ajout de la méthode 'computeCoresLoad'
Il s'agit ici d'enregistrer en temps "réel" (comprendre en temps réel avec l'horloge du simulateur) la charge (CPU, mémoire) sur chacune des machines virtuelles du réseau, en utilisant pour cela les méthodes dédiées et disponibles de base sur le simulateur GreenCloud. Le nombre de tâches reçues et le nombre de tâches effectivement complétées sont également enregistrés.

5. Dans la classe 'CloudUser'
Cette classe décrit toutes les caractéristiques d'un utilisateur du réseau. Notamment, son identité (fonctionnalité ajoutée) et les propriétés de la tâche qu'il envoie sur le réseau.

— Ajout de la méthode 'createSOAPMessage'
Cette méthode commence par ouvrir les fichiers XML (fournis en ressources du projet), pour ensuite les analyser et en dégager le DTD et le corps, à l'aide de la méthode 'analyseXML' de la classe 'SOAPMessage'. Å chaque utilisateur est associé deux messages SOAP, un pour le "broker", et un pour le serveur qui traitera la tâche. Dans les fichiers de scripts, on fixe pour chacun de ces deux messages s'il est malveillant ou non, et cela influencera quel fichier ouvrir et analyser dans cette méthode.

— Ajout des méthodes 'buildUserInfo' et 'readInfo'
Ces méthodes permettent de lire les nom, mot de passe et adresse IP d'un utilisateur en se basant sur son identifiant, identifiant qui est fourni dans les fichiers de scripts tcl. Pour cela, la méthode 'buildUserInfo' appelle à trois reprises la méthode 'readInfo', qui prend en paramètre un fichier (créé par nos soins, et contenant des données fictives d'utilisateur à des fins d'exemple) et renvoie la donnée se trouvant à la ligne correspondant à l'identifiant de l'utilisateur. Ces informations seront ensuite envoyées avec la requête pour que le réseau puisse effectuer des contrôles sur les utilisateurs qui se connectent (fréquence d'envoi des

requêtes, etc.).

6. Dans la classe 'ExpCloudUser'
   Cette classe décrit le type d'utilisateur du réseau utilisé par défaut dans les simulations. Il s'agit d'une classe dérivée de la classe 'cloudUser', qui a pour particularité d'utiliser un traffic de type "on/off" avec une distribution de type exponentielle des moments "on" et "off". Ce traffic est paramétré par un temps moyen de repos, la taille des paquets, le temps moyen d'activité ainsi que la fréquence d'envoi des paquets durant ce temps.

   — Modification de la méthode 'timeout'
   Auparavant, cette méthode se contentait de créer la tâche associée à l'utilisateur, puis de l'envoyer au 'dataCenter'. Désormais, deux messages SOAP sont créés, un pour le "broker" et un pour le serveur qui traitera la tâche (avec la méthode CloudUser : :createSoapMessage). L'identité de l'utilisateur est construite avec CloudUser : :buildUserInfo, puis l'ensemble est envoyé au 'dataCenter'.

7. Création de la classe 'SoapMessage'
   Cette classe définit les concepts clés qui seront utilisés par la suite pour extraire l'information d'un message SOAP. On y définit notamment les structures telles que les noeuds du graphe ainsi que de quoi sont constituées les entités du DTD. Plusieurs booléens sont associés à un message SOAP, afin de connaître avec précision dans quel état le message se trouve à tout instant (ordonnancé, traitement commencé, terminé...). Cette classe contient ensuite une unique méthode : 'analyseXML'.

   — Création de la méthode 'analyseXML'
   Cette méthode stocke dans deux string distincts la donnée du DTD et du corps du message XML, en tenant compte du format d'un message XML, et en ne gardant que l'information essentielle en enlevant le superflu (c'est-à-dire les marques de formatage).

8. Modification de la classe 'GreenScheduler'
   Cette classe définit la façon dont sont allouées les tâches sur les différents serveurs du réseau. Elle possède une seule méthode, 'scheduleTask'. Auparavant, tous les serveurs étaient passés en revue, en commençant par le premier par ordre croissant d'identifiant. Maintenant, et afin de simuler de façon simple les modèles "SaaS"

et "IaaS", deux nouveaux types d'ordonnancement sont proposés. Nous faisons l'hypothèse que dans le cas "SaaS", la tâche de l'utilisateur peut être traitée par n'importe quelle machine du réseau ayant les ressources nécessaires. Cette tâche passe d'abord par le "broker", cette étape simulant le fait que le "broker" va écouter la requête initiale de l'utilisateur et trouver une machine susceptible de traiter sa tâche. Puis une fois ce traitement effectué, une machine disponible est sélectionnée aléatoirement, traite la tâche, et renvoie le résultat à l'utilisateur. Pour le modèle "IaaS", le début du traitement par le "broker" est identique. Ensuite, la machine virtuelle qui sera choisie pour traiter la requête de l'utilisateur dépend de l'identifiant de l'utilisateur, de telle sorte que ce sera toujours la même machine virtuelle qui sera attribuée à un utilisateur donné. Ce qui permet de simuler qu'un utilisateur loue une machine virtuelle du réseau. Dans les deux modèles, l'utilisateur interagit d'abord avec le service web placé sur le "broker", avant d'interagir avec le service web placé sur une des machines virtuelles.

Cette analyse du simulateur GreenCloud, et les modifications à son fonctionnement intrinsèque qui en découlent, répondent au deuxième objectif que nous nous étions fixé : utiliser et améliorer un environnement simulé pour décrire les attaques XML-DoS dans l'infonuagique.

# CHAPITRE 4 EXPLICATIONS ET JUSTIFICATIONS DES DÉFENSES RETENUES

Dans ce chapitre, nous exposons plusieurs défenses contre les attaques XML-DoS, pour ensuite justifier le choix que nous avons fait parmi toutes ces défenses. Nous présentons également avec quelles métriques cette défense sera évaluée par la suite.

## 4.1 Les différents types de défenses

De nombreuses défenses contre les attaques de type XML-DoS sont couvertes dans l'article. Nous ne revenons donc ici que sur quelques-unes d'entre elles, afin d'illustrer les principales stratégies applicables, et notamment celles qui nous ont inspirées dans notre travail.

### 4.1.1 WS-Security

WS-Security est un protocole assurant la sécurité d'un échange entre un émetteur et un destinataire dans une architecture de type SOA. WS-Security agit plus particulièrement sur la confidentialité, l'intégrité et la non-répudiation au sein de cet échange. En revanche, WS-Security ne garantit rien en termes de disponibilité du service, et ne peut donc pas empêcher une attaque DoS sur un service web à l'aide de ce protocole. Cette défense n'est donc pas adaptée à notre cas.

### 4.1.2 Validation et "durcissement" de schéma

Un schéma XML décrit la structure et le type de contenu d'un document XML. Un DTD est par exemple, à quelques différences près, un schéma XML. Tout comme pour le DTD, le but d'un schéma XML est de définir les briques de base d'un document XML. Il peut s'agir de définir des attributs, les liens de parenté entre les éléments (noeuds enfants et noeuds parents), ou encore de définir les types de données. La validation de schéma consiste alors à considérer seulement les messages qui correspondent à la description du service web telle que définie dans le WSDL. La plupart du temps, cette validation n'est pas mise en oeuvre, principalement pour des raisons de performance (coût mémoire et CPU). La validation de schéma peut être une brique fondamentale pour d'autres défenses, par exemple le "durcissement" de schéma.

Le "durcissement" de schéma consiste à analyser la description du service web, et tout particulièrement ce qui dans cette description autorise la construction d'arbres XML très grands

ou très complexes [24]. A partir de cette analyse, des points faibles et susceptibles d'être exploités par une attaque sont découverts et réparés. Il peut par exemple s'agir d'imposer des limites finies à certaines grandeurs qui n'étaient jusque là pas bornées, comme le nombre d'éléments que peut contenir une liste. Ou encore de remplacer certains types de données dont la taille n'est pas fixée (comme des strings), par d'autres types de données dont on contrôle la taille. Fixer de telles limitations est aisé la plupart du temps pour un service web donné. Ces limitations sont par ailleurs connues des utilisateurs pour peu qu'elles se trouvent dans la description publique du service web. Le seul problème avec cette méthode est que des limites qui seraient adaptées à un service web ne le seraient plus forcément pour un autre service web. Autrement dit, il n'y a pas de valeurs "universelles". Le défi n'en est que plus grand dans le cas de réseaux d'infonuagique ou de nombreux services web sont déployés en même temps. La question est alors de savoir s'il est possible de trouver des valeurs qui permettraient de limiter à la fois les "false positives" et les "false negatives".

Afin que la validation du message ne conduise pas elle-même à une attaque (en rendant indisponible l'entité en charge de la validation, ce qui empêcherait les utilisateurs d'accéder aux services web), une implémentation SAX est le plus souvent utilisée, c'est-à-dire que le message est envoyé événement par événement à l'entité chargée de le valider, et non dans son intégralité [18]. Lorsqu'un des événements contenus dans le message n'est pas considéré comme étant "valide" selon les nouveaux critères imposés aux requêtes destinées au service web, le processus de validation du message s'arrête, et la suite du message n'est pas analysée. C'est seulement une fois que le message a été analysé en totalité qu'il est envoyé au serveur web. En pratique, la validation de schéma est recommandée pour prévenir efficacement et simplement les attaques XML-DoS.

### 4.1.3 Adoption de "bonnes pratiques"

Reposant sur les notions vues ci-dessus de validation et/ou durcissement de schéma, Karthigeyan et al. [27] proposent l'adoption de "bonnes pratiques". Celles-ci s'articulent autour de deux axes : réécrire les limites autorisées par le compilateur XML, et surveiller l'interaction entre le service web et les utilisateurs.

Pour ce qui est de réécrire les limites du compilateur XML, il s'agit de vérifier le nombre d'attributs pour chaque élément, le nombre d'octets que peut contenir le message, la profondeur des éléments imbriqués, et enfin la taille de chaque noeud du message XML. Si un de ces éléments n'est pas conforme aux limites qui ont été fixées, une exception est levée et le traitement du message cesse. Les utilisateurs vont être surveillés sur leur fréquence d'envoi de messages, sur le temps que met une de leur requête à être traitée, et sur la taille des mes-

sages envoyés. Lorsqu'une de ces grandeurs dépasse un seuil fixé au préalable, l'utilisateur est interdit d'accès au service web pendant un temps donné.

### 4.1.4 Systèmes intelligents construisant un modèle de requête "normale"

Une façon de faire à la fois suffisamment simple, efficace et adaptable à de nombreux services web, consiste à construire un modèle de requête "normale", en extrayant certaines caractéristiques de ces requêtes [50]. Ainsi, toute requête déviant "suffisamment" (cette notion reste alors à préciser) de la requête "normale" sera rejetée. Les caractéristiques utilisées sont par exemple la longueur du message, le nombre d'éléments, le degré d'imbrication, longueur du plus grand élément, du plus grand attribut et du plus grand "namespace". Le système de défense comprend deux phases : la première traite l'en-tête HTTP (afin d'éliminer directement les requêtes s'inscrivant dans une attaque de type "flooding" par exemple), la seconde s'occupe du contenu XML proprement dit. Ce système est prévu au départ pour protéger le "broker" (l'entité chargée d'écouter la requête initiale de l'utilisateur et d'allouer les ressources nécessaires sur une des machines virtuelles disponibles), mais un tel système de défense peut être placé plus généralement devant tout serveur web SOAP susceptible d'être attaqué. Il agit alors comme un "proxy", et son emplacement réel importe peu, car il agit sur la couche application (HTTP). Pour l'utilisateur, tout se passe comme s'il interagissait directement avec le serveur web, alors qu'en réalité, il envoie ses requêtes à un couple (adresse IP, numéro de port) qui est celui du "proxy". Cependant, ces systèmes intelligents, que ce soit celui-ci ou d'autres qui ont été proposés, sont en général assez complexes à mettre en oeuvre, et coûteux à la fois pour les ressources du réseau, et pour la qualité de service dégradée pour l'utilisateur. Il est donc rare de les voir déployés dans la pratique.

## 4.2 Justification de la pertinence des défenses retenues

Nous venons donc de comparer de façon théorique quelques défenses existantes, en expliquant les différences entre elles et ce qu'elles apportent les unes par rapport aux autres. Nous reviendrons plus en détail dans le chapitre suivant sur ces défenses, notamment leur efficacité en termes de "false positives" et "false negatives", et sur leur implémentation de façon pratique sur de vrais réseaux. Nous avons d'ores et déjà répondu à la première partie de notre troisième objectif : l'étude des défenses classiques contre les vulnérabilités des services web. Afin de tester la faisabilité et l'efficacité des défenses existantes contre les attaques de type XML-DoS, nous avons choisi d'implémenter dans notre simulateur les "bonnes pratiques" énumérées ci-dessus. En effet, ces mesures sont à la fois rapides à mettre en oeuvre, ne représentent pas un surcoût important au moment de la validation ([18] revendique un coût en mémoire constant

et dépendant de la taille du schéma, et une exécution en temps linéaire), et sont très efficaces pour peu que les bonnes bornes soient appliquées dans la description du service web. Pour un fournisseur de services d'infonuagique, elles sont donc très intéressantes, mais notre objectif sera de déterminer leur efficacité réelle, notamment si le choix des bornes est mal choisi, car c'est là que réside toute la difficulté : comment se prémunir de toutes les attaques tout en s'assurant que les utilisateurs légitimes auront, eux, accès aux services ? Dans un contexte de simulation, ces mesures sont également celles qui paraissent les plus sensées, grâce à leur relative facilité de mise en oeuvre (comparativement à des systèmes intelligents qui seraient très complexes à modéliser dans le cadre d'une simulation), et la rapidité avec laquelle on peut ajuster les paramètres de la défense en fonction des scénarios de simulation choisis, afin de déterminer leur efficacité. L'implémentation de ces "bonnes pratiques" dans le simulateur (voir chapitre 3 pour plus de détails sur l'implémentation) finit de répondre à notre troisième objectif, qui consistait à comparer les défenses existantes contre les attaques XML-DoS, et à en sélectionner certaines pour les tester dans la pratique.

## 4.3 Justifications des métriques évaluées

Afin de déterminer avec quelle efficacité les défenses retenues permettent d'atténuer les effets d'une attaque XML-DoS, nous avons retenu trois métriques : l'utilisation CPU, mémoire, et le nombre généré de "false positives" et "false negatives".

### 4.3.1 CPU et mémoire

Une attaque XML-DoS est une attaque de déni de service visant un épuisement des ressources CPU et mémoire. Ces deux grandeurs sont donc les principaux paramètres d'intérêt lors d'une telle attaque. Un système de défense efficace doit donc avoir pour priorité d'atténuer ces utilisations de ressources abusives. Il doit permettre une détection et un retour à la normale rapide. Il faut remarquer que tout système de défense, et en particulier celui retenu, induit lui-même un surcoût CPU et mémoire pour la machine, potentiellement d'autant plus grand que le message à vérifier est compliqué. Cependant, cette prise en compte est laissée pour des travaux futurs. Pour l'instant, nous avons jugé suffisant d'effectuer les vérifications de messages en suivant un ordre bien déterminé, qui permet de s'assurer que chaque étape de la vérification se fera rapidement et à un coût presque négligeable. Nos défenses seront donc évaluées principalement sur cette capacité d'atténuation de l'utilisation CPU et mémoire lors d'une attaque.

### 4.3.2 "False positives" et "false negatives"

Un autre paramètre presque aussi important que le précédent est le nombre d'utilisateurs considérés à tort comme étant légitimes ou au contraire illégitimes par le système de défense. L'un comme l'autre sont tout autant à éviter. En effet, si une attaque est extrêmement efficace pour filtrer une attaque, mais qu'en contrepartie aucun utilisateur ne peut entrer sur le réseau, car ses requêtes sont perçues à tort comme des attaques (taux élevé de "false positives"), l'effet est le même que lorsqu'il n'y a aucun système de défense : les utilisateurs légitimes n'ont plus accès aux services. D'un autre côté, si le nombre de "false negatives" est trop important, cela signifie que de nombreux messages illégitimes arrivent à pénétrer le réseau et sont alors en mesure de perpétrer des dénis de service. Un système de défense efficace se doit donc, en plus d'atténuer sensiblement voire totalement le coût CPU et mémoire induit par une attaque XML-DoS, d'atténuer un maximum à la fois le nombre de "false negatives" et le nombre de "false positives".

Dans ce chapitre, nous avons donc vu quelles étaient les différentes approches envisageables pour contrer les attaques XML-DoS. Un choix s'est imposé dans notre cas : l'adoption des bonnes pratiques. Système de défense très prometteur sur le papier, en reposant notamment sur une stricte validation du schéma XML, avec un surcoût faible en temps et en espace (mémoire), il sera intéressant d'en mesurer l'efficacité dans la pratique. Pour cela, trois métriques seront analysées : utilisation CPU, mémoire, et nombre de "false negatives" et de "false positives".

# CHAPITRE 5 RÉSULTATS THÉORIQUES ET EXPÉRIMENTAUX

Les changements effectués sur le simulateur et présentés dans le chapitre 3 permettent dorénavant d'étudier le comportement d'un réseau d'infonuagique lorsque certaines des requêtes reçues sont en réalité des attaques de type XML-DoS. Notre démarche est la suivante : tout d'abord, nous présentons les paramètres de la simulation, notamment la topologie du réseau et les caractéristiques des utilisateurs. Ensuite, nous mettrons en évidence l'efficacité des attaques XML-DoS en l'absence de défense pour les modèles SaaS et IaaS. Nous commenterons également les effets de l'interférence de performances entre deux machines virtuelles situées sur la même machine physique lors d'une attaque XML-DoS. Enfin, nous évaluerons les défenses retenues au chapitre précédent. Cette évaluation tiendra compte des différents paramètres possibles pour la défense, et des 3 métriques identifiées au chapitre 4, à savoir l'utilisation CPU, mémoire, ainsi que le nombre de "false positives" et de "false negatives". Nous discuterons ensuite des limitations de la défense proposée.

## 5.1 Les paramètres de la simulation

Les résultats obtenus dans nos simulations dépendent à la fois des caractéristiques des serveurs (caractéristiques techniques et topologie), que des caractéristiques des utilisateurs du réseau. Nous détaillons ci-après les choix que nous avons retenus dans ces deux domaines.

### 5.1.1 Topologie et caractéristiques techniques des serveurs

#### Topologie

La topologie retenue pour le réseau d'infonuagique est décrite dans la figure 5.1

Il y a donc 144 serveurs en tout. C'est la topologie par défaut du simulateur GreenCloud. Nous aurions pu utiliser des topologies plus compliquées, avec plus de serveurs et plus de "switches", mais cela n'aurait pas eu d'influence dans notre cas, car une attaque de type XML-DoS se comporte de la même façon et a les mêmes effets peut importe la topologie : le but est de rendre inopérant un serveur web. S'il y a trois fois plus de serveurs web, l'attaque prendra trois fois plus de temps à se mettre en place, et l'interconnexion des serveurs n'entre pas non plus en compte.

Le "broker", qui est un élément remarquable du réseau, se confond néanmoins avec les autres

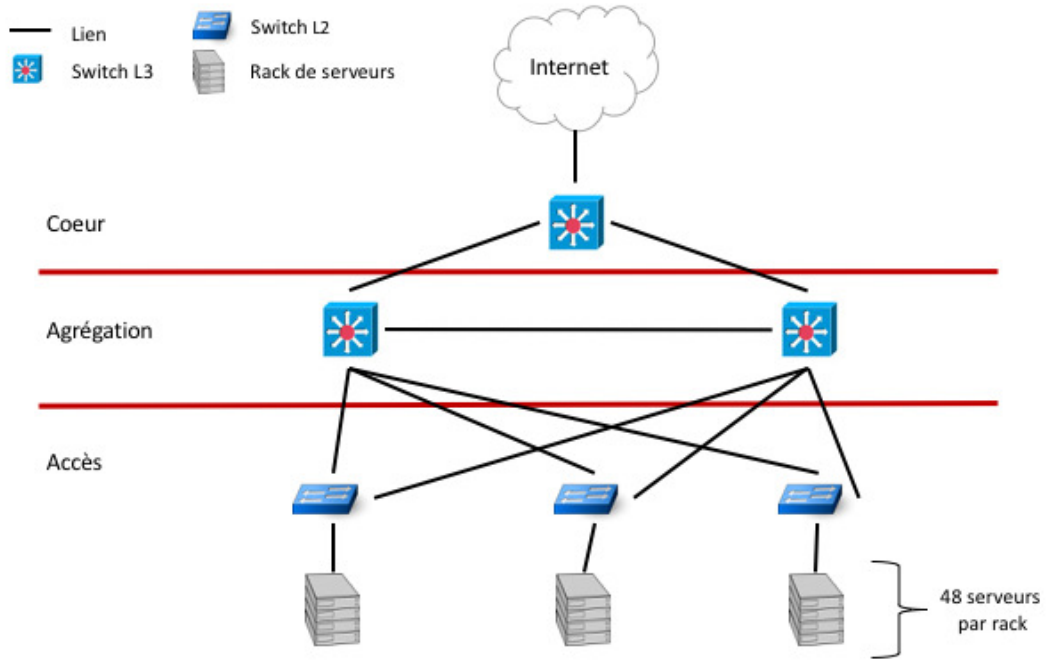


Figure 5.1 Topologie du réseau simulé dans GreenCloud

serveurs dans la topologie. Il n'est pas physiquement placé différemment des autres serveurs. Il s'agira toujours du serveur n°0, qui ne se distingue donc des autres que par son identifiant. Tous les paquets entrants pour la première fois au sein du réseau emprunteront donc le même chemin et passeront par les mêmes liens et les mêmes switches. Une amélioration possible à ce scénario serait alors de réserver un chemin à part pour parvenir au broker, ce chemin ayant par ailleurs des liens de plus grande capacité afin de ne pas créer de goulot d'étranglement.

L'interaction des utilisateurs avec le réseau se fait via les machines virtuelles installées sur chacun des serveurs (et non avec le serveur lui-même). Le nombre de machines virtuelles installées sur chaque serveur est paramétrable. Nous avons choisi d'installer deux machines virtuelles sur chaque serveur dans le cas "IaaS", afin de mesurer l'impact de l'interférence de performances entre deux machines virtuelles, et seulement une dans le cas "SaaS" pour simplifier l'interprétation des résultats expérimentaux.

Chaque machine virtuelle (y compris sur le "broker") est supposée disposer d'un serveur web avec lequel l'utilisateur peut interagir. C'est ce serveur web qui est le vecteur de l'attaque. Les caractéristiques du serveur web n'entrent pas en ligne de compte, mais tous doivent avoir

recours à une compilation de type "DOM" en accord avec ce qui a été vu au chapitre 2.

Intéressons-nous maintenant plus précisément aux caractéristiques techniques des serveurs et des machines virtuelles installées.

#### Caractéristiques techniques des serveurs

Les principaux paramètres d'intérêt étant le processeur et la mémoire, nous laisserons de côté les détails concernant les "switches" et les capacités des liens. Étant donné que le nombre de machines virtuelles par serveur varie selon le modèle que l'on souhaite simuler ("SaaS" ou "IaaS"), nous avons retenu deux configurations.

— Configuration des serveurs dans le cas "SaaS"
Le serveur dispose d'un unique processeur monocoeur et de 4GB de mémoire. Chaque machine virtuelle utilise 100% des capacités du serveur hôte (ceci est permis dans la simulation), avec 4GB de mémoire et un processeur virtuel de même capacité que le processeur du serveur hôte.

— Configuration des serveurs dans le cas "IaaS"
Comme nous voulons mesurer l'interférence de performance liée à la contention pour accéder au processeur, nous avons choisi d'accorder à chaque serveur un processeur double coeur, et 8GB de mémoire. Chaque machine virtuelle dispose alors de 4GB de mémoire (comme précédemment), et d'un processeur virtuel utilisant 100% des capacités de chacun des coeurs du processeur du serveur.

Les derniers paramètres de notre simulation concernent les utilisateurs du réseau.

### 5.1.2 Caractéristiques des utilisateurs

Les utilisateurs possèdent de nombreux paramètres que nous n'avons pas modifiés, par exemple :

— Le type de traffic
— Les caractéristiques de chaque tâche : la place prise en mémoire, la charge de calcul associée ou encore la date limite pour exécuter la tâche
— La fréquence d'envoi des messages

Ainsi, que l'utilisateur soit légitime ou non, il enverra toujours le même nombre de paquets,

et au même rythme.

Les paramètres modifiés correspondent au caractère malveillant ou non de l'utilisateur, et si la cible des utilisateurs malveillants est le "broker" ou alors une autre machine, quelconque, du réseau. Nous pouvons alors ajuster le pourcentage de paquets malveillants envoyés, et aussi décider si un utilisateur est toujours associé à la même machine virtuelle ou non ("IaaS" ou "SaaS").

Dans les simulations qui suivent, le nombre d'utilisateurs total dépend du modèle choisi. Ainsi, dans le cas du modèle "SaaS", 300 utilisateurs se partagent les ressources du réseau (c'est-à-dire 144 serveurs à une machine virtuelle par serveur), et le nombre d'utilisateurs malveillants varie entre 1 et 10. Ce pourcentage de traffic malveillant est une bonne approximation de ce qui est observé lors d'attaques DoS réelles. Dans le cas du modèle "IaaS", et puisque nous voulions garder la même capacité globale en calcul et en mémoire que pour le cas "SaaS", et donc le même nombre de machines virtuelles, et puisqu'enfin chaque utilisateur dispose de sa propre machine virtuelle, ce sont 142 utilisateurs qui se partageront le réseau (144 moins les deux machines virtuelles utilisées par le "broker").

Nous pouvons à présent présenter les résultats expérimentaux obtenus.

## 5.2 Mise en évidence de l'efficacité de l'attaque en l'absence de défense pour les modèles "IaaS" et "SaaS"

Afin de mettre en évidence l'efficacité d'une attaque XML-DoS contre les réseaux d'infonuagique, nous avons mesuré l'évolution de l'utilisation CPU et mémoire globale dans les modèles "SaaS" et "IaaS". Le message SOAP utilisé est celui correspondant à l'attaque "Exponential Entity Expansion". Dans le cas "SaaS", chaque tâche d'un utilisateur est ordonnancée sur un serveur qu'il n'est pas possible de déterminer à l'avance. Ainsi, un utilisateur malveillant qui enverrait suffisamment de requêtes pourrait progressivement rendre indisponible un nombre de plus en plus grand de serveurs. Par ailleurs, un seul message peut suffire à monopoliser le CPU et la mémoire d'un serveur.

La figure 5.2 permet de constater l'efficacité d'une attaque XML-DoS contre les réseaux d'infonuagique de type "SaaS". Alors qu'en l'absence d'attaque, la charge globale se maintient autour de 0.2, elle augmente de façon linéaire lorsqu'un utilisateur malveillant commence à envoyer des paquets sur le réseau, et à saturer les hôtes un à un (rappelons qu'aucun système de défense n'est pour l'instant mis en place). L'effet est encore plus marqué lorsque le nombre d'utilisateurs malveillants passe à 10 (soit environ 3% du traffic global), ou quelques secondes suffisent à rendre l'ensemble du réseau inopérant.

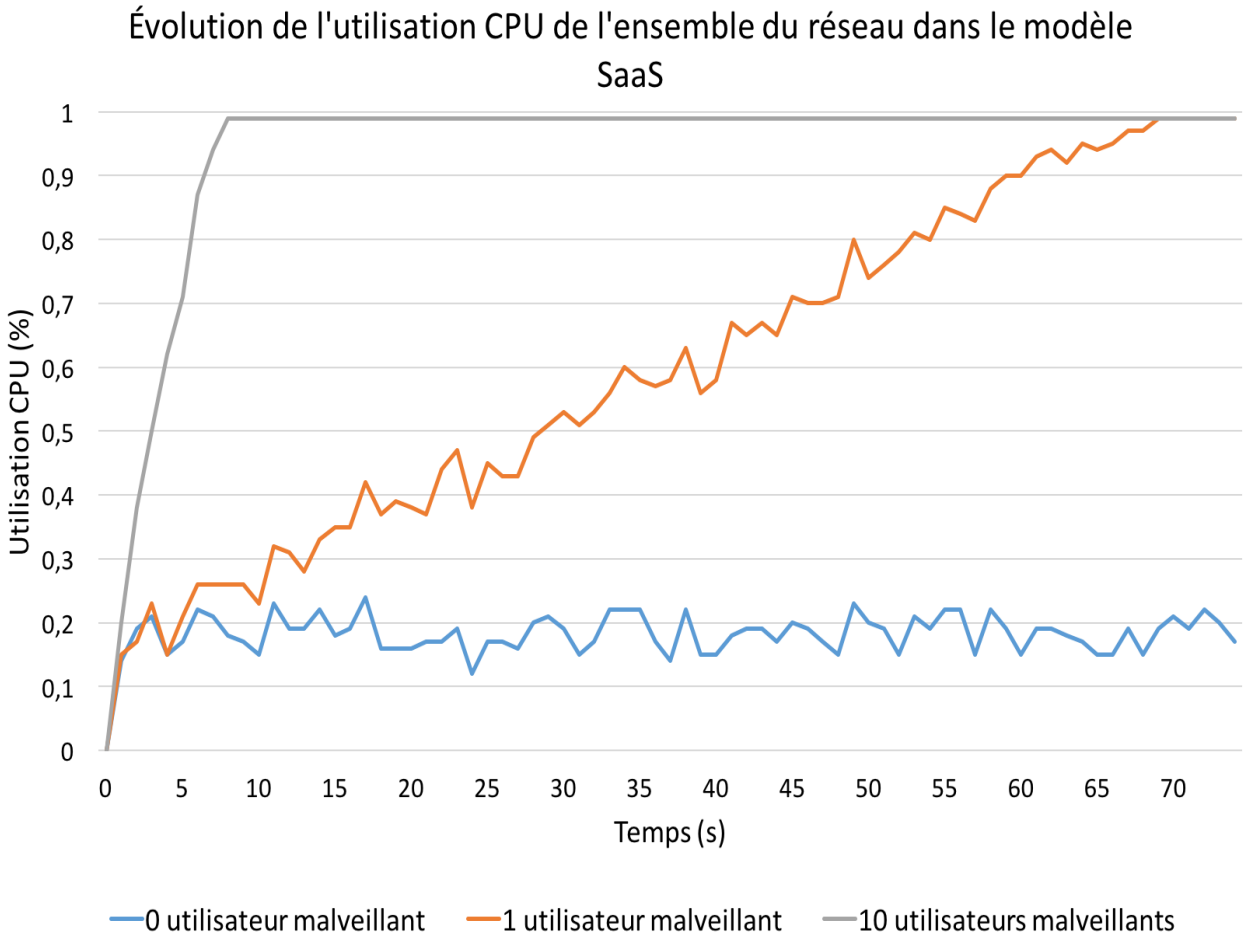


Figure 5.2 Évolution de l'utilisation CPU de l'ensemble du réseau avec le temps pour "SaaS"

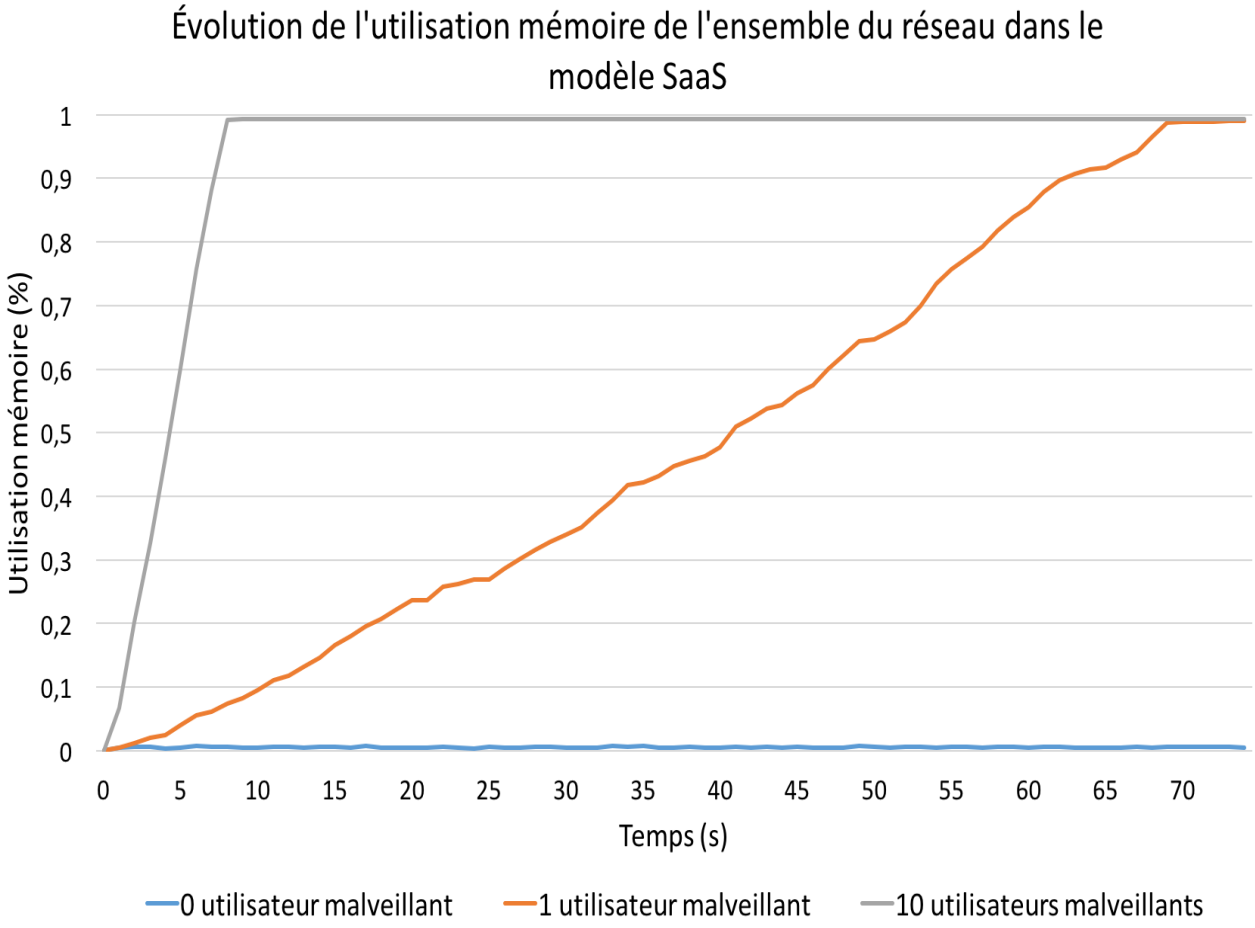


Figure 5.3 Évolution de l'utilisation de la mémoire de l'ensemble du réseau avec le temps pour "SaaS"

Quant à la mémoire dont l'évolution est retracée dans la figure 5.3, les conclusions sont en tout point similaires à celles dressées ci-dessus. L'effet de l'attaque est extrêmement sensible, et augmente de façon exponentielle avec le nombre d'attaquants impliqués. Il est à noter que la saturation mémoire de l'ensemble du réseau se produit environ au même moment que la saturation CPU. Comme nous allons le voir, les conclusions vont être bien différentes dans le cas du modèle "IaaS".

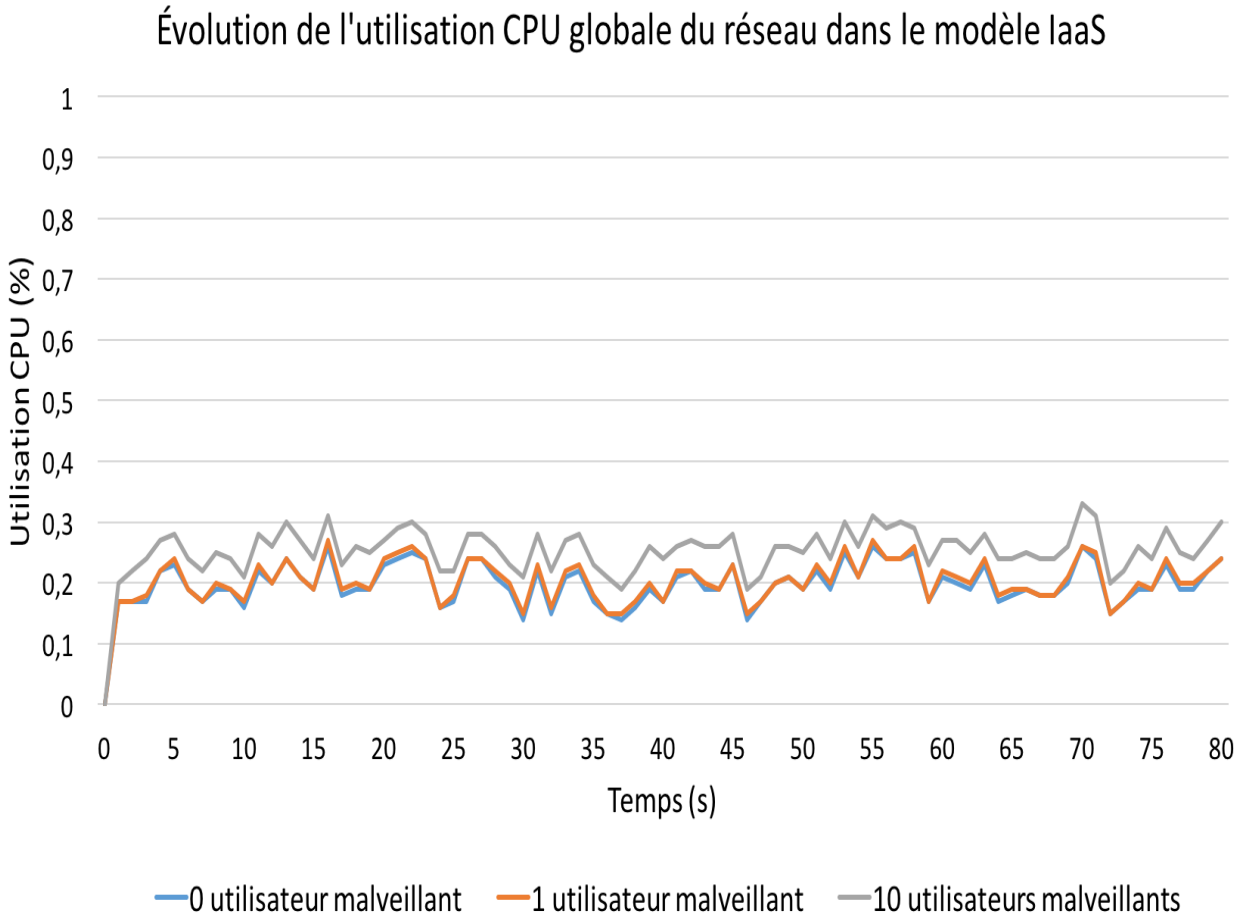


Figure 5.4 Évolution de l'utilisation CPU de l'ensemble du réseau avec le temps pour "IaaS"

Étant donné qu'un utilisateur n'a accès qu'à une seule machine virtuelle, un attaquant sera dans l'incapacité de rendre indisponible l'ensemble du réseau, à moins qu'il ne loue ou ne corrompe un très grand nombre de machines virtuelles (ce qui sera soit extrêmement coûteux soit extrêmement complexe). Ainsi, les attaquants ciblant une seule machine virtuelle ne sont pas en mesure de perpétrer une attaque de déni de service sur l'ensemble du réseau, mais bien seulement sur la machine qui leur a été allouée. Ainsi, la charge globale du réseau augmente

très légèrement, mais les utilisateurs situés sur les autres serveurs non attaqués ne seront pas affectés par ces attaques (contrairement aux utilisateurs dont la machine virtuelle cohabite avec la machine virtuelle attaquée, comme nous le verrons plus loin).

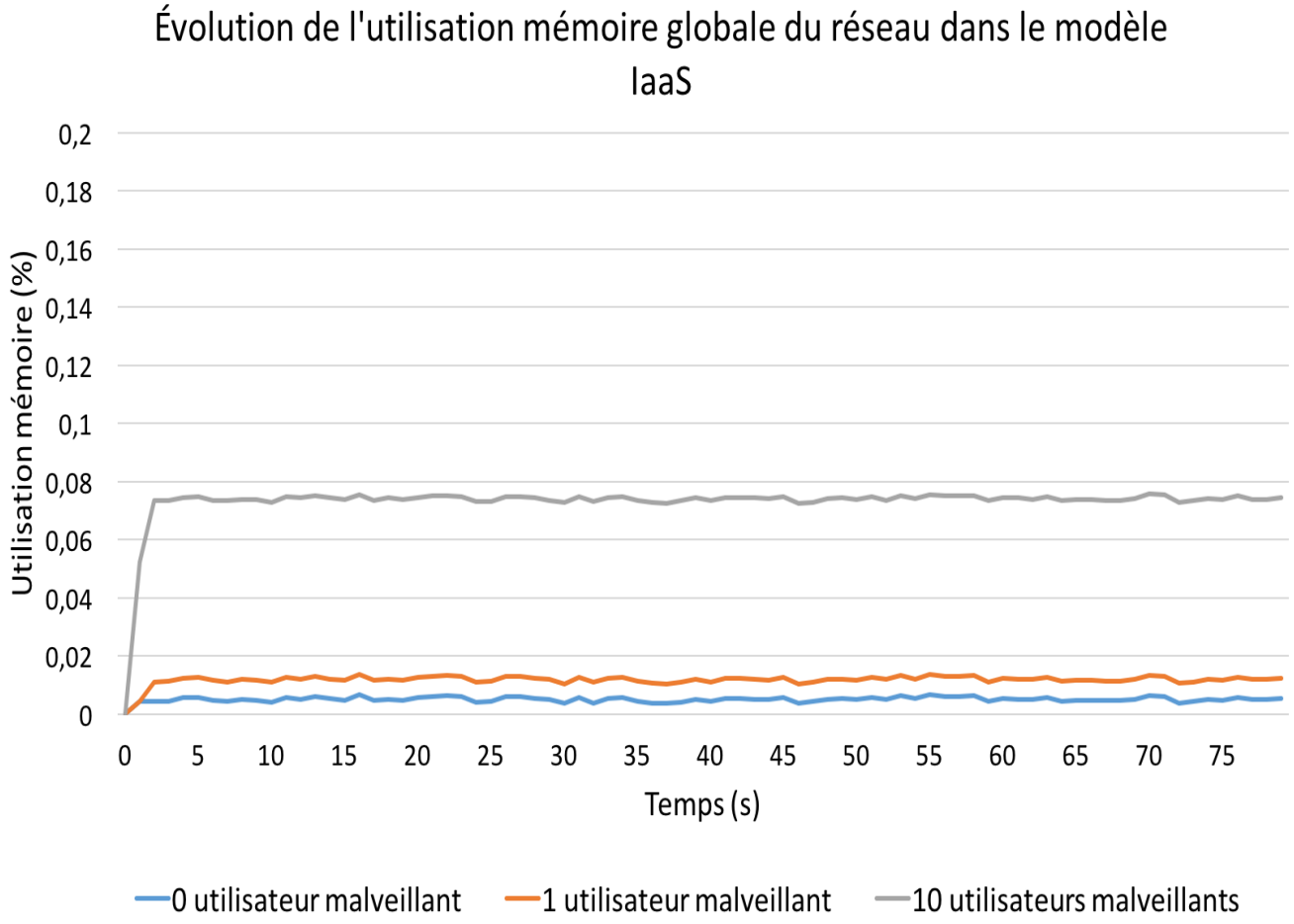


Figure 5.5 Évolution de l'utilisation de la mémoire de l'ensemble du réseau avec le temps pour "IaaS"

Les résultats présentés dans la figure 5.5 concordent avec les conclusions sur l'utilisation CPU. L'impact de l'attaque est globalement extrêmement localisé et très faible à l'échelle du réseau, et les utilisateurs situés sur des serveurs non attaqués n'auront vraisemblablement aucune baisse de qualité de service (ralentissements, tâches échouées, etc.).

Tous les résultats présentés jusqu'à présent se basent sur une attaque des machines virtuelles situées à l'intérieur du réseau. Intéressons-nous à ce qu'il advient lorsque l'attaquant cible le "broker" avec comme objectif de rendre l'ensemble du réseau indisponible, en privant les

utilisateurs de la "porte d'entrée" du réseau. Les résultats sont présentés figure 5.6.

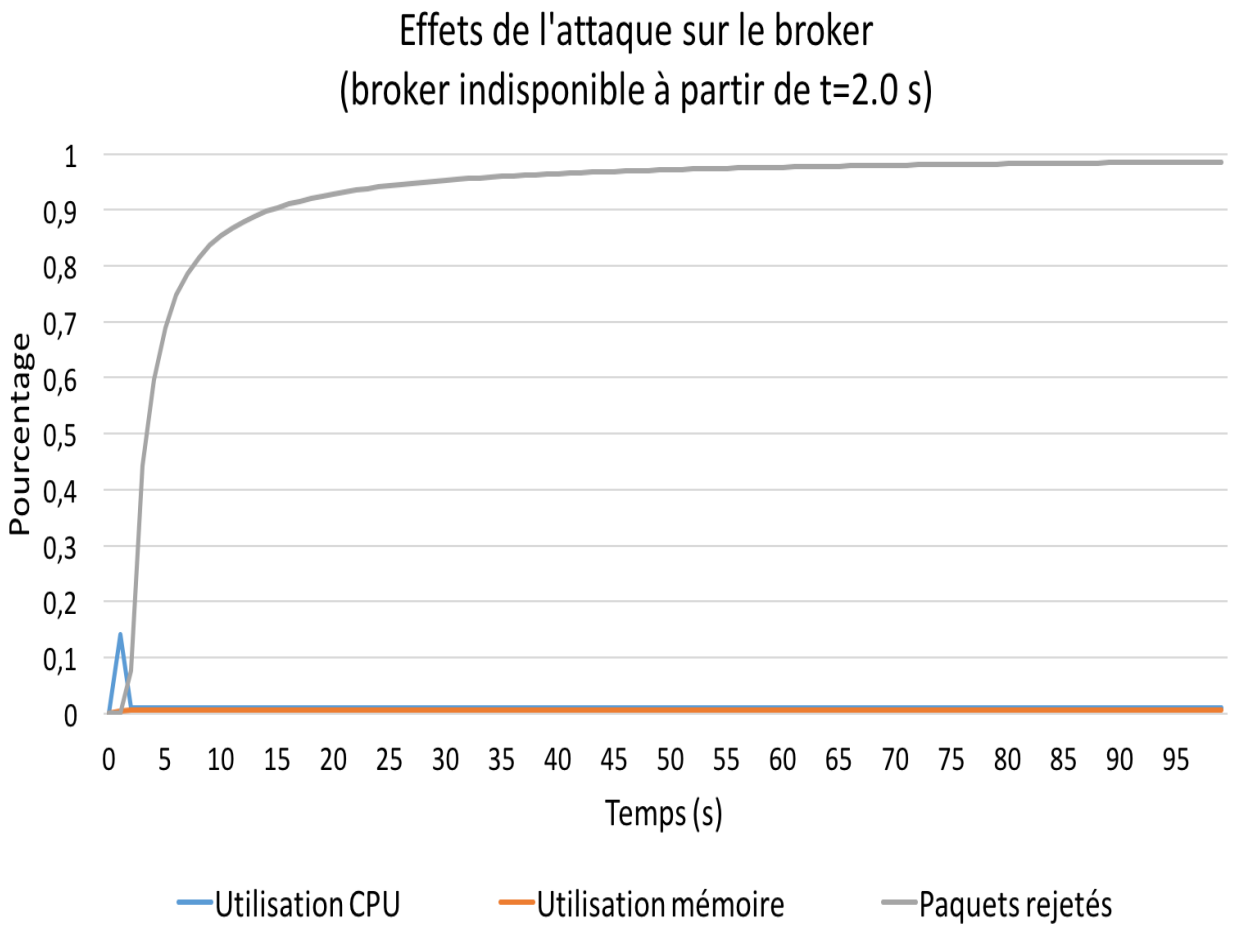


Figure 5.6 Conséquences de l'attaque du "broker" sur l'utilisation globale des ressources du réseau (broker indisponible à partir de t=2.0 s)

Comme attendu, nous apercevons que lorsque le broker est indisponible suite à une attaque XML-DoS, plus aucun utilisateur ne peut entrer sur le réseau, donc tous les serveurs sont à l'état de veille, avec aucune utilisation mémoire ou CPU, et l'ensemble des paquets reçus au niveau du "broker" vont être rejetés une fois celui-ci indisponible.

Puisque nous avons conclu que les attaques XML-DoS n'avaient pas d'impact au niveau global sur les réseaux de type "IaaS", étudions plus précisément l'impact de telles attaques au niveau des machines virtuelles attaquées, et la façon dont se manifeste l'interférence de performance entre deux machines virtuelles situées sur la même machine physique.

## 5.3 Impact de l'interférence de performances entre machines virtuelles lors d'attaques XML-DoS

Dans cette section nous montrons les effets de l'interférence de performance entre machines virtuelles, en analysant l'utilisation moyenne du CPU, le nombre moyen de tâches ordonnancées, et le rapport entre le nombre de tâches ordonnancées et le nombre de tâches effectivement traitées. Les résultats présentés sont des moyennes sur l'ensemble des machines virtuelles du réseau, en distinguant celles qui sont attaquées de celles qui ne le sont pas. Il n'y a toujours pas de système de défense à ce stade.

La figure 5.7 présente l'utilisation CPU moyenne des machines virtuelles du réseau, dans le cas d'une attaque et en l'absence d'attaque. Pour rappel, deux machines virtuelles cohabitent sur chaque serveur et doivent se partager un unique CPU.

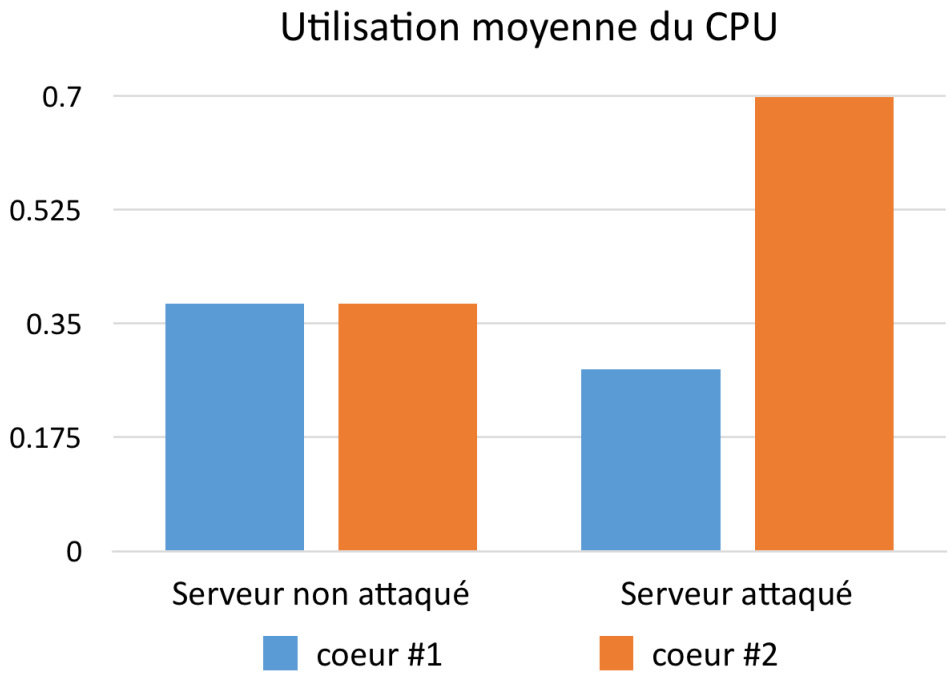


Figure 5.7 Utilisation moyenne du CPU dans le modèle "IaaS", chaque coeur du serveur est attribué à une des machines virtuelles (2 machines virtuelles par serveur).

Il apparaît donc une interférence de performance entre deux machines virtuelles devant se partager le même CPU hôte. Ainsi, lorsque le serveur n'est pas attaqué (histogramme de gauche), les deux machines virtuelles se partagent équitablement le CPU. En revanche, lorsqu'une des deux machines présente une charge particulièrement élevée (par exemple lors d'une attaque XML-DoS), cela a des répercussions sur l'autre machine virtuelle, qui présente alors

des performances moindres. Avec notre modélisation, nous constatons un grand déséquilibre entre les deux coeurs d'un serveur attaqué, avec une répartition 70%-30%, au lieu d'un partage équilibré avec utilisation de 40% de la capacité de calcul pour chaque coeur dans cette situation (nombre et taille de paquets à traiter). Ainsi, même si une attaque XML-DoS n'aura pas d'impact à l'échelle globale, elle pourra conduire à une nette dégradation de service, non seulement pour la machine victime de l'attaque s'il s'agit d'une machine corrompue, mais également pour le ou les utilisateurs des machines virtuelles voisines et partageant les mêmes ressources. C'est ce que nous retrouvons dans la figure 5.8, présentant le nombre de tâches qui ont pu être ordonnancées suivant que le serveur est attaqué ou non. Les tâches ordonnancées sont les tâches qui seront effectivement exécutées sur une machine. Comme nous pouvons le constater, ce nombre n'est pas forcément égal au nombre de tâches reçues sur le réseau, notamment lors d'une attaque. En effet, lorsqu'une machine n'a plus de ressources mémoire ou CPU, elle ne peut plus recevoir de nouvelles tâches, et celles-ci ne sont donc jamais exécutées.

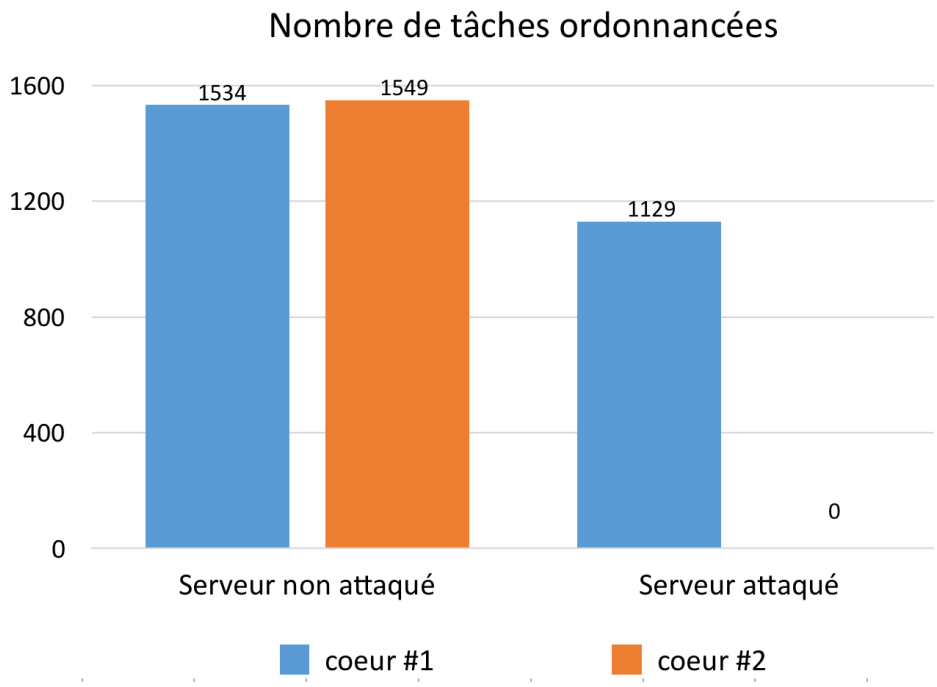


Figure 5.8 Nombre moyen de tâches ordonnancées dans le modèle "IaaS", chaque coeur du serveur est attribué à une des machines virtuelles (2 machines virtuelles par serveur)

Si le serveur est attaqué, la machine attaquée ne traite qu'un seul paquet (traitement qui n'aboutit jamais), et la machine virtuelle voisine, disposant de moins de ressources, ne pourra pas traiter toutes les tâches qu'elle aurait dû traiter (1129 tâches contre plus de 1500 normalement).

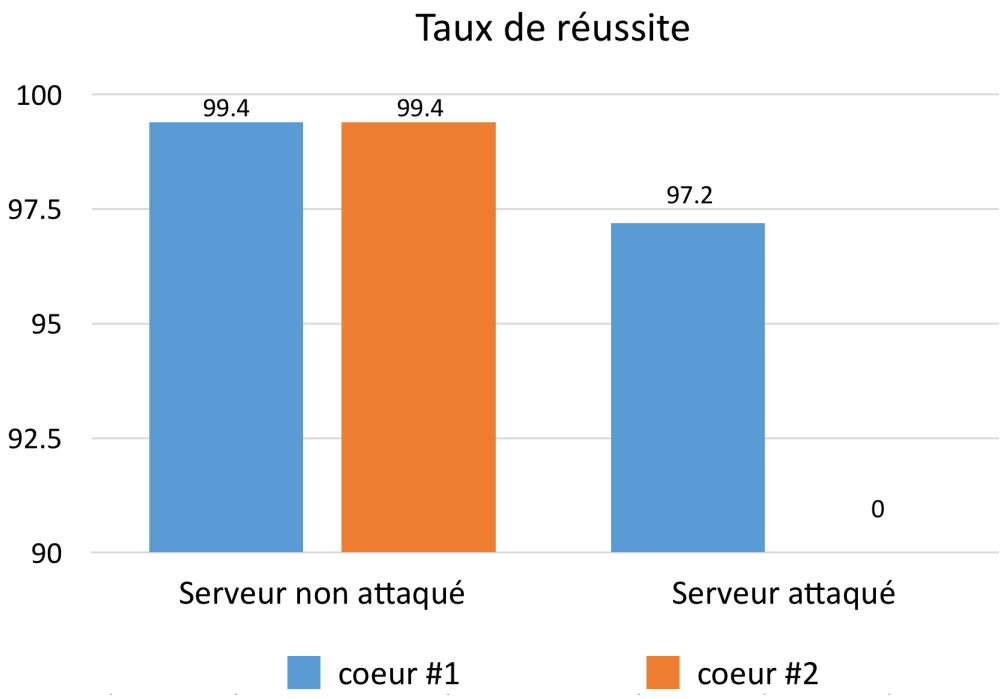


Figure 5.9 Taux de réussite : rapport entre nombre de tâches traitées et nombre de tâches reçues dans le modèle "IaaS". Chaque coeur représente une machine virtuelle du serveur

Dans la figure 5.9, le nombre de tâches reçues correspond au nombre de tâches ordonnancées sur une machine virtuelle. Comme nous venons de le voir, ce nombre est diminué lors d'une attaque, certaines requêtes n'étant jamais ordonnancées. Ainsi, non seulement la machine virtuelle voisine de la machine attaquée recevra moins de tâches que prévu, mais ces tâches ont également plus de probabilités d'échouer. En effet, alors que le ratio entre le nombre de tâches reçues et le nombre de tâches traitées avec succès est proche de 100% dans le cas d'un serveur non attaqué, il descend à 97% dans le cas d'une machine virtuelle voisine d'une machine attaquée (et il est bien sûr de 0 dans le cas de la machine virtuelle attaquée).

Les attaques XML-DoS apparaissent donc comme une réelle menace pour les réseaux d'infonuagique, que ce soit pour les modèles "SaaS" ou "IaaS". Il convient donc d'apporter des défenses adéquates, à la fois efficaces, simples à mettre en oeuvre, et précises (en terme de "false positives" et "false negatives"). Cette défense doit permettre de protéger à la fois le "broker" et chacune des machines virtuelles du réseau susceptibles d'être attaquées. Nous avons implémenté dans notre simulateur une défense qui s'apparente aux "bonnes pratiques" qu'il s'agit d'adopter pour limiter le plus possible le risque d'attaques XML-DoS, notamment avec une stricte validation du schéma du message. Nous ne nous occupons pas pour l'ins-

tant de vérifier le comportement des utilisateurs (fréquence d'envoi de messages et temps de traitement des précédents messages).

## 5.4 Évaluation des défenses contre ces attaques

Maintenant que nous avons vu l'impact des attaques XML-DoS sur les réseaux d'infonuagique, nous allons évaluer l'efficacité des défenses retenues et identifiées au chapitre 4. L'évaluation portera sur la capacité d'atténuation de l'utilisation CPU et mémoire abusive d'un message XML-DoS, mais également sur le nombre de "false postives" et "false negatives".

### 5.4.1 Les différents degrés de défense

Afin de tester le système de défense proposé, basé sur une stricte validation du message SOAP, nous avons défini trois degrés de protection. Chaque protection se définit par les valeurs limites permises de récursivité (la taille de l'arbre associé au noeud en question), de nombre d'attributs par élément, de taille maximale pour chacun des noeuds, ainsi que de taille limite pour le message au complet.

Le tableau ci-après résume les valeurs pour les trois défenses :

Tableau 5.1 Les 3 degrés de protection du système de défense, des valeurs plus petites correspondant à une vérification plus stricte

| propriété | défense 1 | défense 2 | défense 3 |
|---|---|---|---|
| Taille d'un noeud | 20 | 50 | 90 |
| Taille totale | 500 | 1000 | 2000 |
| Récursivité | 2 | 5 | 7 |
| Nombre d'attributs | 3 | 5 | 7 |

Ces valeurs ont été déterminées empiriquement à partir de la banque de données de messages SOAP que nous avons créée, et qui a pour but de représenter la diversité des services web cohabitant habituellement dans un réseau d'infonuagique (plus de détails à ce sujet dans la prochaine sous-section). La première défense aura tendance à bloquer davantage de paquets, mais aura l'avantage d'empêcher tout message malveillant de pénétrer à l'intérieur du réseau (voire tout message, même légitime, demandant trop de ressources). La dernière défense quant à elle sera plus souple au moment d'autoriser, ou non, un paquet, mais il faudra alors s'assurer que des messages malveillants ne puissent tout de même pas pénétrer à l'intérieur du réseau. L'évaluation des défenses va porter sur la faculté à atténuer l'utilisation moyenne du CPU et de la mémoire lors d'une attaque XML-DoS, et à limiter le nombre de "false negatives" et de "false positives".

### 5.4.2 Évaluation de ces défenses quant au nombre de "false positives" : choix d'une défense optimale

Les "false positives" sont les paquets signalés comme dangereux alors qu'ils sont en réalité légitimes. Pour évaluer la performance de ces trois défenses sur les paquets illégitimement rejetés, nous avons considéré que 75% du traffic était constitué de messages SOAP très basiques qui ne surchargent aucunement le réseau, tandis que les 25% restant utilisaient des messages SOAP choisis aléatoirement dans une banque de données regroupant des messages légitimes, mais bien entendu adaptés à un serveur web en particulier, et qui ne suivent donc pas le même format, que ce soit pour le nombre d'attributs, la taille de chaque noeud, ou bien le degré de récursivité. En effet, chaque serveur web effectue un traitement spécifique, et il est donc impossible d'imposer un unique format. Le défi consiste donc à définir des règles suffisamment strictes pour empêcher les attaques XML-DoS, mais qui laisseraient tout de même assez de liberté pour ne pas bloquer des messages qui seraient légitimes. Voyons comment sont traités ces messages légitimes par notre système de défense, en calculant le nombre de "false positives". Nous avons considéré trois degrés de défense. Le premier est le plus restrictif, et aura tendance à bloquer plus de messages, tandis que le dernier est le plus souple. Le deuxième degré est un compromis entre la restrictivité du premier et la souplesse du dernier.

Comme nous le voyons figure 5.10, si l'on ne veut accepter aucun "false positives", le troisième degré de défense, c'est-à-dire le plus souple, est requis. En effet, les degrés de défense 1 et 2 conduisent respectivement à un taux de "false positives" de 5.5% et 2.5%. Analysons alors l'efficacité de la défense ainsi paramétrée (défense numéro 3) contre l'attaque menée précédemment.

### 5.4.3 Efficacité de cette défense optimale : protection contre l'attaque initiale

Nous allons de nouveau simulé l'attaque présentée au début de ce chapitre, en comparant les utilisations CPU et mémoire pour l'ensemble du réseau avec et en l'absence de défense, et en comparant ces résultats au scénario sans attaque. Les résultats sont présentés aux figures 5.11 et 5.12.

Nous pouvons conclure grâce aux figures 5.11 et 5.12 que le système de défense proposé (l'adoption de bonnes pratiques avec le troisième degré de protection, comme expliqué dans le tableau 5.1 page 77) est très efficace pour filtrer le message porteur de l'attaque XML-DoS, et ainsi garantir la disponibilité et leurs pleines ressources à l'ensemble du réseau, tout en garantissant que les messages légitimes et hétérogènes puissent parvenir à leur destination.

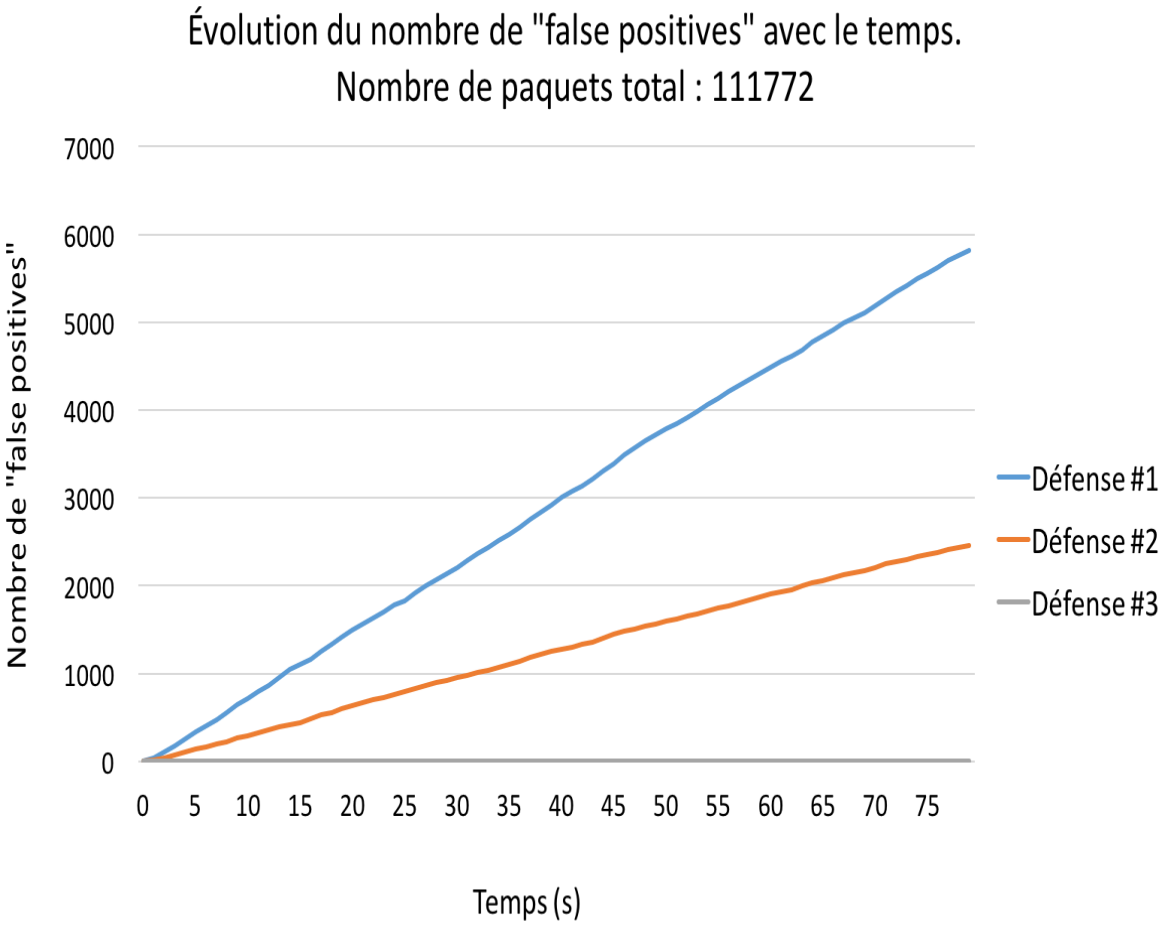


Figure 5.10 Évolution du nombre de "false positives" avec le temps, en fonction des différents degrés de défense

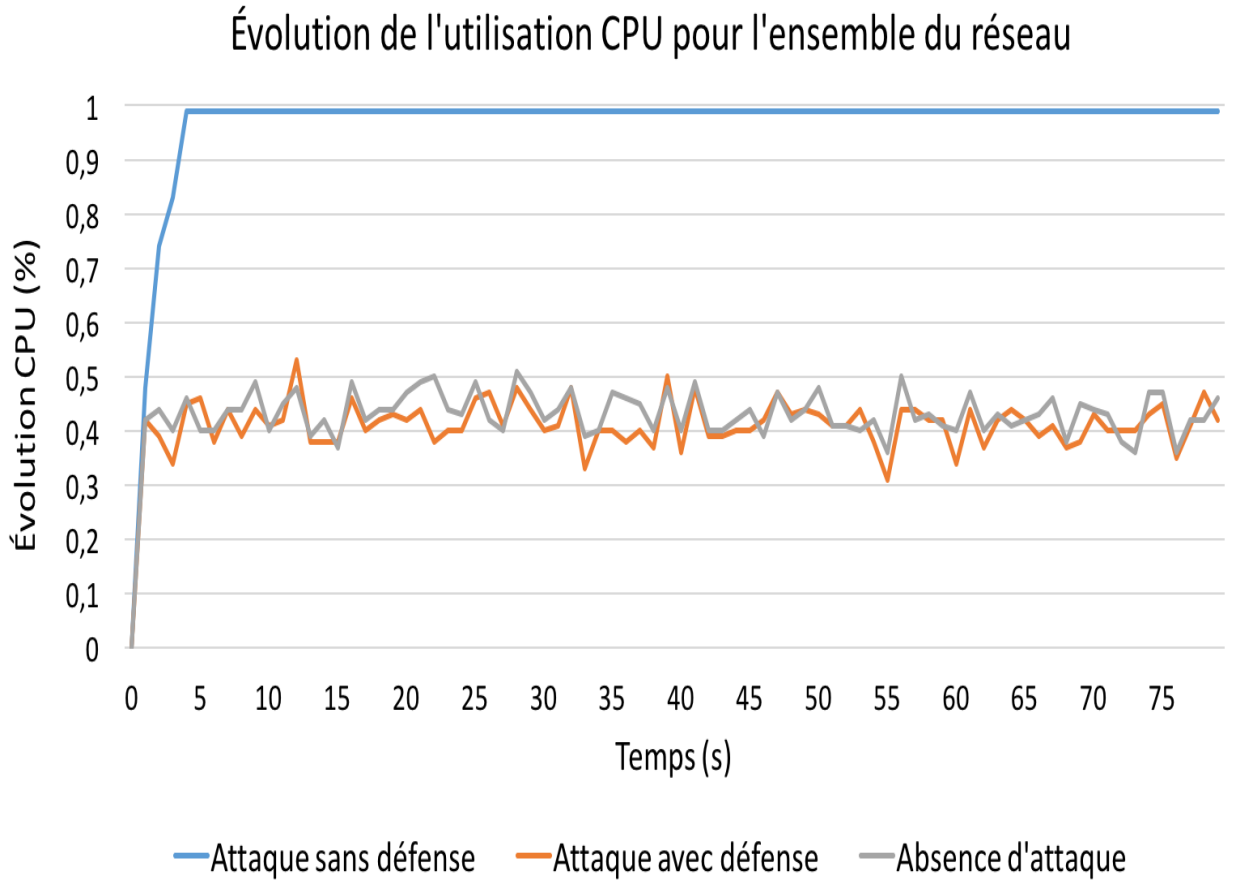


Figure 5.11 Efficacité de la défense : évolution de l'utilisation CPU pour l'ensemble du réseau (en pourcentage de la capacité totale)

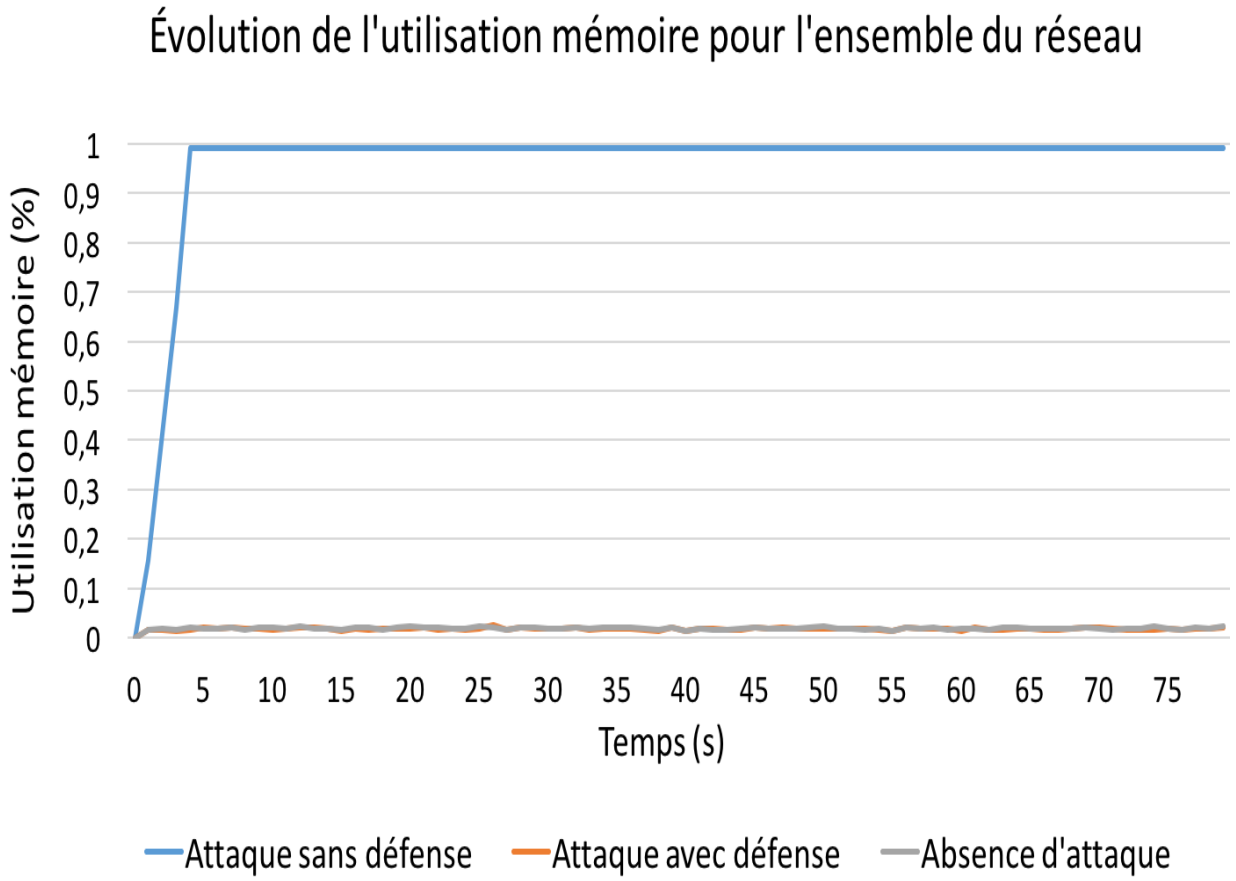


Figure 5.12 Efficacité de la défense : évolution de l'utilisation mémoire pour l'ensemble du réseau (en pourcentage de la capacité totale)

En effet, les courbes montrant l'impact sur l'utilisation CPU et mémoire dans le cas d'une attaque avec la défense mise en place, et celles où il n'y a aucune attaque, se confondent parfaitement, et contrastent nettement avec le cas d'une attaque sans défense. Les résultats sont identiques pour le "broker" : en présence du système de défense, le nombre de paquets traités, ainsi que l'utilisation CPU et mémoire sont identiques qu'il y ait une attaque (avec le message utilisé précédemment) ou non, donc la défense protège parfaitement les équipements vulnérables contre ce type de messages en particulier. Ce qui répond à la première partie de notre dernier objectif, à savoir l'étude de l'efficacité des défenses.

Cependant, qu'arriverait-il si des messages respectant chacune des contraintes définies, mais à chaque fois "à la limite", étaient envoyés sur le réseau ? Nous allons voir que de tels messages peuvent à leur tour être utilisés pour mener des attaques de type XML-DoS contre le réseau, et que notre système de défense se révèle alors complètement inefficace.

### 5.4.4 Limitations de la défense

Dans cette expérience, nous avons tenté de fabriquer un message SOAP dont toutes les caractéristiques restent dans les limites permises par le système de défense, tout en étant en mesure de conduire à un XML-DoS, afin d'étudier la possibilité d'obtenir des "false negatives". Ce qui ne veut pas nécessairement dire que tous les paramètres sont fixés à la limite permise, car alors il y a forcément des paramètres qui dépasseraient de la limite. Par exemple, si l'on fixe tous les paramètres à la valeur limite permise, sauf la taille totale, cette dernière a alors une valeur bien supérieure à ce qui est admis. Il faut donc jouer avec les paramètres pour obtenir le message qui mènera le plus efficacement à une attaque XML-DoS, tout en respectant les contraintes de la défense. Après de nombreux essais, nous avons conclu que les noeuds "de plus haut niveau", c'est-à-dire ceux ayant beaucoup de noeuds enfants, devaient avoir une taille assez restreinte, alors que les feuilles de l'arbre devraient avoir la taille la plus grande. Ainsi, à taille égale, l'arbre d'arrivée occupera une bien plus grande place en mémoire, car il y aura alors énormément de feuilles très volumineuses, et assez peu des noeuds racines qui sont bien plus petits. Comme nous allons le voir, ce type de message, qui passera sans encombre les vérifications, présente à certains égards la même efficacité que le premier message que nous avions utilisé. Il y a donc des "false negatives" avec ce système, puisque des paquets malveillants arrivent à pénétrer dans le système. Les résultats sont présentés figure 5.13.

Sur les 300 utilisateurs du réseau, 10 utilisateurs sont malveillants. La courbe d'évolution des ressources, que ce soit pour le CPU ou la mémoire, est assez différente de ce que l'on observait sur les figures 5.2 et 5.3. Le réseau va rester disponible pendant plus longtemps

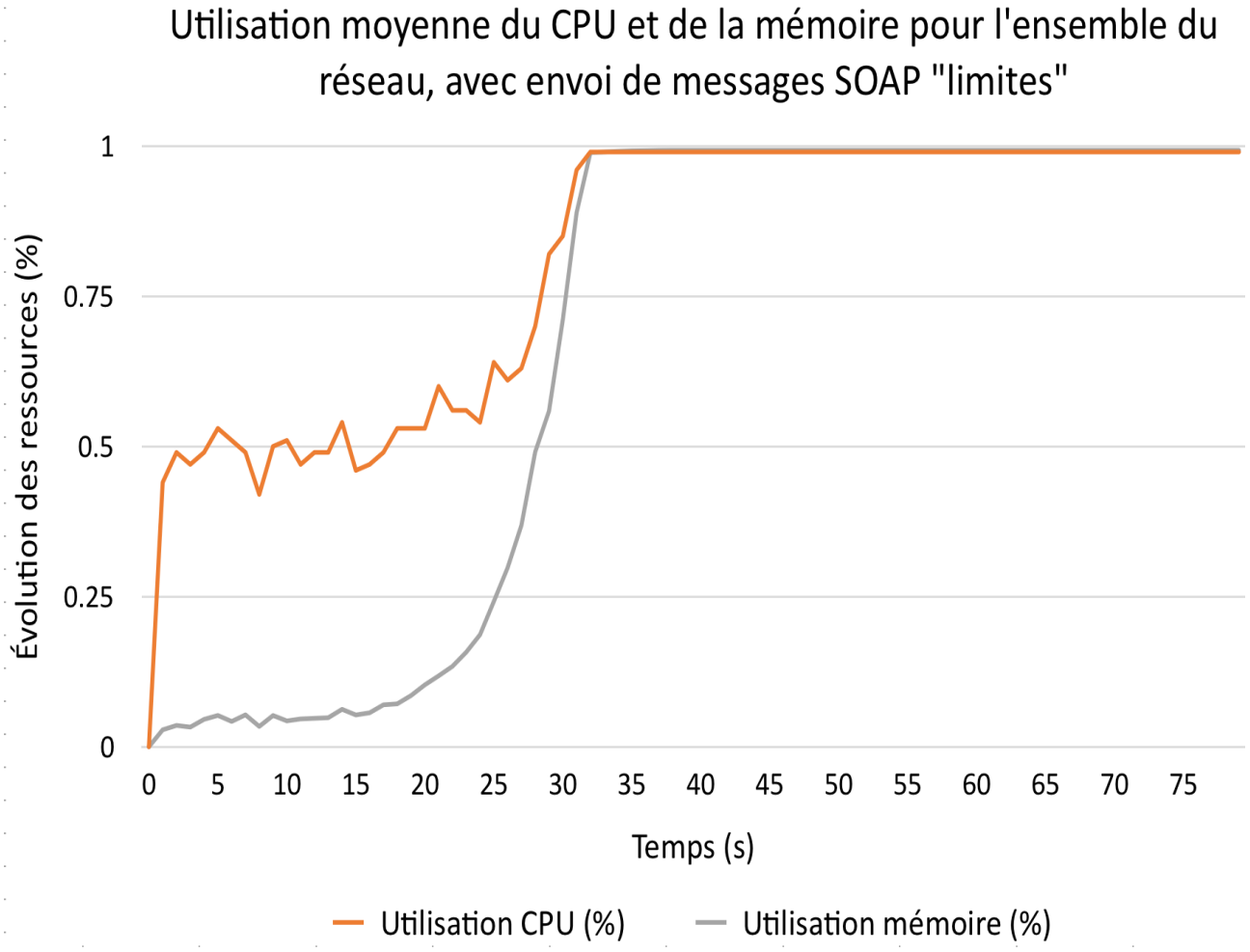


Figure 5.13 Évolution des ressources du réseau avec envoi de paquets à la limite de ce qu'autorise la défense

(une trentaine de secondes au lieu de 10 auparavant), mais à partir de là il ne suffira que de quelques secondes pour rendre le réseau totalement inopérant, et réaliser une attaque de déni de service.

### 5.4.5 Discussions sur l'effet du contrôle des utilisateurs

Nous avons voulu déterminer l'influence qu'avait le blocage d'utilisateurs dont les messages étaient trop longs à être traités. Nous laissons de côté la vérification de la fréquence d'envoi de messages, car dans notre simulation tous les utilisateurs, qu'ils soient légitimes ou non, ont la même fréquence d'envoi, étant donné que la fréquence n'est pas un paramètre pertinent dans le cas des attaques XML-DoS. Nous partons du principe qu'il n'y a pas de défense liée à l'inspection du message SOAP afin de ne considérer que l'aspect lié à l'inspection des utilisateurs. Nous avons par ailleurs gardé les paramètres utilisés lors des simulations sur les systèmes de défense liés au message SOAP, à savoir 300 utilisateurs, parmi lesquels le quart envoie des messages demandant plus de ressources au système, sans pour autant être des messages malveillants. L'utilisation de ces messages doit permettre de vérifier si bloquer un utilisateur en fonction du temps que prennent ses requêtes à être exécutées peut conduire à des "false positives". Les résultats expérimentaux obtenus nous ont fait constater que la plupart des messages avaient des temps d'exécution proches. Ceci est dû à un nombre assez restreint de messages, qui ne permet pas d'étudier avec précision l'influence de la limitation du temps de traitement sur le nombre de "false positives". En effet, des variations très faibles sur la limite imposée faisaient grandement varier le nombre de "false positives". Nous avons cependant constaté que tous les messages semblaient s'exécuter en moins de $10^{-5}$s (temps du simulateur). Notre message ayant servi à l'attaque ayant un temps de traitement supérieur, il sera bloqué, et le traitement peut même être interrompu lorsque le temps de traitement dépasse ce seuil (vérification à chaque nouvelle création de noeud). Cette vérification semble donc assez efficace, mais elle doit cependant être nuancée par notre manque de valeurs de référence, et donc par le caractère arbitraire de la limite fixée. De plus, une requête ne demandera pas forcément toujours le même temps de traitement, dépendamment de la charge globale du serveur sur lequel se trouve la machine virtuelle (à cause du partage CPU entre les différentes machines virtuelles), ou encore bien sûr de l'activité sur la machine hébergeant le service web. Ainsi, un utilisateur légitime pourra potentiellement voir sa requête rejetée ou acceptée dépendamment de conditions externes. Par ailleurs, comme cette défense se base sur le comportement passé d'un utilisateur, chaque utilisateur malveillant peut envoyer au moins une fois un message avant d'être bloqué (même si l'on peut implémenter un système forçant l'arrêt du traitement au bout d'un certain temps). Si ce blocage n'est pas permanent, et qui plus est assez court, alors le système de défense sera réellement inefficace, notamment

si ce temps de blocage coïncide ou est légèrement plus petit que le temps entre deux envois de message de l'utilisateur malveillant. Cette surveillance du comportement des utilisateurs, bien qu'efficace au premier abord, est donc elle aussi très complexe à mettre en oeuvre correctement dans le cadre de l'infonuagique.

### 5.4.6 Conclusions sur la défense proposée

Même si la défense proposée semble efficace et facile à mettre en oeuvre pour empêcher les attaques de type XML-DoS, elle se révèle finalement assez limitée dans la pratique. Non seulement elle requiert de fixer manuellement les paramètres de la défense, mais elle exige surtout de connaître précisément tous les services web délivrés par l'intermédiaire du réseau d'infonuagique, afin de trouver une valeur optimale pour ces paramètres de défense, ce qui en pratique est presque impossible, étant donnée la grande diversité des services web déployés en tout temps sur un réseau d'infonuagique. Cette défense permet toutefois de faire une première sélection des requêtes qui seront traitées par le réseau, et il s'agira par la suite d'ajouter un deuxième système de défense permettant de diminuer le taux de "false positives" et de "false negatives". Ces défenses plus sophistiquées devront faire face au défi majeur de transparence du point de vue de l'utilisateur, dont il ne faut pas pénaliser la qualité de service, et à la complexité de mise en oeuvre pour les administrateurs du système. Ceci répond bien à la deuxième partie de notre dernier objectif, l'analyse de la nécessité, ou non, de défenses plus complexes, et ainsi plus généralement à notre dernier objectif : l'analyse des défenses grâce à leur implémentation dans le simulateur.

# CHAPITRE 6 DISCUSSION GÉNÉRALE

L'article présenté au chapitre deux permet de présenter de façon exhaustive l'ensemble des menaces de sécurité qui peuvent peser sur l'infonuagique, notamment dans le cadre des attaques de déni de service, ainsi que les défenses existantes et leurs possibles limitations. Nous y démontrons l'écart entre la facilité de mise en oeuvre des attaques DoS et leur impact. Étant donnée la grande diversité des attaques et défenses répertoriées dans cet article, et souvent leur manque de mise en pratique, nous avons pris le parti de retenir une attaque, le XML-DoS, et des défenses basées sur la stricte validation de schéma, afin de les tester et de tirer des conclusions basées sur la pratique et non la théorie. Ces choix sont le résultat des conclusions de l'article que nous avons rédigé, qui met à jour des vulnérabilités très importantes, mais encore peu étudiées et pour lesquelles il n'existe pour l'instant pas réellement de solutions.

La méthodologie suivie dans ce mémoire diffère grandement de ce que l'on trouve dans la littérature. L'usage d'un simulateur est par exemple assez novateur, alors même que l'étude des attaques comme les XML-DoS a besoin d'un suivi méticuleux de l'évolution des ressources pour déterminer avec précision leur impact selon diverses métriques (CPU, mémoire, etc.). Ainsi, il n'est pas rare que des auteurs présentent uniquement de façon théorique l'importance de se prémunir contre les attaques XML-DoS et l'efficacité de leur système de défense.

Notre approche est donc assez originale, car elle considère un problème clairement identifié dans la littérature, et permet de tester les défenses généralement recommandées, pour vérifier si les défenses plus sophistiquées qui ont été proposées sont, ou non, une nécessité.

Parmi les autres approches qui ont été adoptées pour étudier les attaques XML-DoS dans le cadre de l'infonuagique, nous pouvons comparer les différences dans le protocole expérimental.

Vissers et al. [50] commencent par déterminer quelles attaques XML-DoS représentent toujours une menace. Leur but va alors consister à apporter un système de défense capable d'atténuer les effets des attaques toujours susceptibles de causer des attaques de déni de service. Ce système se base sur l'extraction des caractéristiques des requêtes, pour définir un comportement "normal", avec notamment l'examen de l'en-tête HTTP puis du contenu XML proprement dit. Ce système a pour objectif de protéger le serveur web SOAP devant lequel il est situé (ce système agit alors comme un filtre). Les auteurs mettent en évidence la vulnérabilité du "broker" dans l'infonuagique, et leur système n'est destiné pour l'instant qu'à protéger cet équipement. Ainsi, un utilisateur qui réussirait à déjouer la défense placée devant le "broker" pourrait toujours causer un déni de service à l'intérieur du réseau, alors que nous

avons placé notre défense devant chaque serveur web susceptible d'être attaqué. Ils montrent que leur système est efficace pour détecter et contrer les attaques XML-DoS montrées comme étant les plus dangereuses, mais le taux de "false positives" ou "false negatives" associé à leur défense n'est pas précisé, par manque de données. Leur simulation repose sur l'utilisation du logiciel Eucalyptus, permettant un réseau d'infonuagique sur une grappe de serveurs. Leur système est en revanche plus sophistiqué que le nôtre, car il peut adapter dynamiquement la validation de schéma (les auteurs estiment même que leur système pourrait détecter de nouvelles attaques), en se basant sur l'observation de requêtes aberrantes, alors que notre validation de schéma est fixe, et n'est pas capable de s'adapter automatiquement.

Karthigeyan et al. [27] proposent d'implémenter les "meilleures pratiques" pour se protéger contre les attaques XML-DoS. Ils imposent notamment des limites sur la taille maximale du message, sur la fréquence d'envoi de messages, et sur le temps maximum que doit prendre une requête à être traitée. Si une de ces limites est franchie, l'utilisateur est bloqué pendant un certain temps. Ils imposent également des limites sur la forme du message SOAP, sur le nombre d'attributs maximum pour chaque élément, la profondeur de chaque élément ou encore la taille maximale de chaque noeud. Cependant, aucun test pratique n'est réalisé, les auteurs se contentant de schémas explicatifs sur les différentes limitations proposées. Ainsi il n'est pas possible de connaître l'efficacité réelle de cette défense (protection contre les attaques XML-DoS, tout en limitant le nombre de "false positives"). De plus, les auteurs ne décrivent l'attaque que dans le cas simple d'un unique serveur web. Notre approche est donc bien plus globale et complexe, car il s'agissait alors de protéger tout un réseau, et de prendre en compte ses spécificités (topologie, machines virtuelles, etc.).

# CHAPITRE 7 CONCLUSION ET RECOMMENDATIONS

L'infonuagique est un paradigme informatique très prometteur et qui va probablement connaître un essor encore plus fulgurant dans les prochaines années. Ces réseaux extrêmement configurables et qui bénéficient d'une excellente mise à l'échelle, sont fait tout autant pour les petites entreprises que pour les individus souhaitant s'affranchir du superflu comme la maintenance ou la mise à jour de leurs systèmes, pour se concentrer sur leur travail, tout en gardant la possibilité de ne payer que pour ce qui est effectivement utilisé. Cependant, une des propriétés des réseaux d'infonuagique, l'ubiquité, c'est-à-dire la faculté d'accéder au réseau depuis n'importe quel emplacement et depuis n'importe quel terminal, nécessite le recours aux services web. Ceci permet en effet l'interaction de deux systèmes hétérogènes. Or ces services web présentent de très sérieuses vulnérabilités, comme les attaques de niveau application (couche 7 du modèle OSI), qui ne peuvent pas être détectées par des systèmes comme les pare-feux, qui opèrent principalement au niveau des couches réseau et transport (filtrage basé sur l'adresse IP ou le numéro de port par exemple). C'est le cas par exemple des attaques XML-DoS. Le problème vient de l'utilisation du protocole "SOAP", qui repose sur le XML. Le protocole SOAP permet la communication et l'envoi de message XML entre systèmes hétérogènes, en utilisant la plupart du temps le protocole HTTP pour assurer le transport. Cependant, on ne peut se contenter de sécuriser la couche transport (avec SSL par exemple), car rien ne garantit que ce sera le seul protocole utilisé, il y a donc un besoin de sécuriser le message SOAP directement. Des standards ont été proposés, mais ne s'intéressent pour la plupart qu'à l'aspect confidentialité et intégrité (XML-Signature, XML-Encryption ou WS-Security). Il existe donc un certain manque en termes de systèmes capables "d'introspection", c'est-à-dire en mesure d'examiner le contenu du message SOAP, et de valider le schéma du message XML. Tous les modèles d'infonuagique sont potentiellement vulnérables à ces attaques, que ce soit par l'intermédiaire d'un équipement comme le "broker", ou même en rendant indisponible un serveur sur lequel s'exécutent plusieurs machines virtuelles légitimes. Le but de ce mémoire a été de répondre à la question de savoir si ces attaques XML-DoS étaient vraiment des menaces dans la pratique, quels étaient les équipements les plus vulnérables, quels étaient les impacts pour les utilisateurs légitimes, et surtout quelles défenses permettaient de s'en protéger le plus efficacement.

## 7.1 Synthèse des travaux

Nous avons commencé par rédiger un article regroupant l'état de l'art des attaques de déni de service contre les réseaux d'infonuagique, afin de dresser un premier bilan des vulnérabilités inhérentes à ce type de réseaux, ainsi que les défenses possibles et leur efficacité. Nous y avons brièvement abordé la question des attaques XML-DoS.

Ceci nous a donc naturellement amener à vérifier si les attaques XML-DoS étaient effectivement des menaces pour les réseaux d'infonuagique. Pour cela, nous avons utilisé un simulateur, basé sur le célèbre simulateur NS2, et adapté à l'étude de l'infonuagique, nommé GreenCloud. Ce simulateur a dû être modifié en profondeur, car il n'est à la base prévu que pour mesurer le coût énergétique d'un réseau d'infonuagique sous différentes configurations. Nous avons commencé par quelques rappels à propos du langage XML, pour nous aider à mieux comprendre comment les attaques XML-DoS étaient rendues possibles, notamment les différentes façons de compiler un message XML, et comment ces façons rendaient possible ou non l'attaque (les parseurs DOM y sont vulnérables, mais pas les parseurs SAX). Nous avons ensuite sélectionné deux des attaques XML-DoS les plus efficaces répertoriées : le "coercive parsing" et l'attaque "quadratic blowup", comme base de travail pour la suite. Puis nous avons fait un inventaire de plusieurs des défenses existantes pour protéger les serveurs web, pour finalement en choisir une dont l'efficacité sera testée dans le simulateur. Nous nous sommes attachés à modéliser des services web, avec la prise en charge et le traitement de messages XML. Nous avons aussi ajouté dans la simulation les équipements dont la disponibilité est un point critique et que nous avons jugés vulnérables aux attaques XML-DoS, comme le "broker", en lui associant un traitement à part entière. Par ailleurs, le simulateur ne prenait pas réellement en compte l'utilisation de machines virtuelles dans l'infonuagique (pas de surcoût de traitement ni de contention pour les ressources). Nous avons donc ajouté la possibilité d'avoir autant de machines virtuelles qu'il est physiquement possible d'en avoir (étant données les spécifications des serveurs hôtes), et avons entamé une modélisation du partage des ressources entre deux machines virtuelles situées sur la même machine physique. Nous avons également modifié le simulateur afin de pouvoir simuler différents modèles d'infonuagique, comme les modèles "SaaS" et "IaaS", afin d'étudier comment chacun de ces modèles était impacté par les attaques XML-DoS. Les ajouts et modifications au simulateur GreenCloud constituent une des plus grosses contributions de notre travail. Ces modifications peuvent servir de base pour de futurs travaux et pourront être réutilisées dans d'autres cadres. Les modules ajoutés peuvent être complétés et améliorés, et d'autres modules pourraient venir s'y ajouter.

Les résultats expérimentaux ont confirmé que les attaques XML-DoS sont une réelle menace pour les réseaux d'infonuagique, notamment lorsque ceux-ci présentent des topologies à risque (par exemple un unique "broker" dont l'indisponibilité prive l'ensemble des utilisateurs d'accès aux services). Cependant, même avec une topologie plus robuste, ces attaques peuvent nuire grandement à la qualité de service perçue par les utilisateurs, en exploitant de façon abusive les ressources des serveurs victimes. Le plus remarquable est qu'il n'est même pas nécessaire d'avoir recours à une attaque distribuée pour parvenir à un déni de service. Une requête peut suffire rendre indisponible un serveur. Le système de défense proposé, qui se présente comme une stricte validation du schéma XML, éventuellement assorti de la surveillance du comportement des utilisateurs, s'est révélé très efficace et facile à mettre en oeuvre (que ce soit dans le cadre de cette simulation, mais également dans la pratique, puisque ne nécessitant aucun changement dans la topologie ou l'infrastructure du réseau). Cependant, si ce système de défense permet déjà de filtrer de nombreuses requêtes malveillantes, des messages proches des limites acceptées, donc en apparence légitimes, peuvent encore conduire à un déni de service ("false negative"). En effet, le besoin de protéger tous les services web indépendamment de leurs spécificités propres et des actions qu'ils réalisent, obligent, pour limiter le nombre de "false positives", à fixer des limitations assez souples. L'alternative serait d'adapter le système de défense à chaque service web à protéger, mais cela est difficilement réalisable dans le cadre de l'infonuagique, où de nombreux services web cohabitent en permanence, et cette option n'a donc pas été envisagée. Cela ouvre donc la discussion sur les améliorations qu'il faudrait apporter à cette défense, par exemple le contrôle du comportement des utilisateurs, même si ici encore des compromis seraient à trouver entre "false positives" et "false negatives". Pour parvenir à un système de défense bien plus performant, il semble donc qu'il faille se tourner vers des systèmes dits "intelligents", basés par exemple sur des modèles de requête "normale", construits en extrayant des caractéristiques pertinentes, ou encore des systèmes basés sur des réseaux de neurones, et entrainés à détecter et à filtrer les attaques XML-DoS.

Nous avons effectivement répondu à nos quatre objectifs, qui étaient d'étudier la vulnérabilité des réseaux d'infonuagique aux attaques XML-DoS, d'améliorer un simulateur d'infonuagique afin d'y simuler des attaques XML-DoS, d'analyser les différentes façons de se protéger contre les attaques XML-DoS, et enfin d'étudier certaines de ces défenses à la lumière des résultats obtenus dans notre simulateur. Notre apport a été de considérer les attaques XML-DoS, assez peu étudiées, dans le cadre de l'infonuagique, au travers d'un environnement simulé, permettant de mesurer précisément l'impact de ces attaques, tant du point de vue des ressources

utilisées que de la qualité de service pour les utilisateurs légitimes. En effet, la plupart des approches existantes sont soit seulement théoriques, soit hors du cadre de l'infonuagique.

## 7.2 Limitations de la solution proposée et difficultés rencontrées

La solution proposée pâtit pour l'instant de quelques limitations, qu'il s'agisse de simplifications ou même de caractéristiques laissées de côté. Par exemple, notre modélisation du partage CPU entre machines virtuelles cohabitant sur la même machine physique pourrait être plus fidèle à ce qui se passe dans la pratique. La modélisation des requêtes SOAP pourrait elle aussi être plus sophistiquée, et prendre en compte la structure complète d'un tel message, par exemple l'en-tête d'une requête SOAP, qui peut être utilisé pour prévenir certaines attaques.

Parmi ce que nous avons préféré laissé de côté pour l'instant, il y a par exemple la non-prise en compte du surcoût en performance lié à l'utilisation de machines virtuelles pour délivrer les services web au lieu de les délivrer directement sur la machine hôte. Ainsi, il faudrait prendre en compte que davantage de CPU est nécessaire pour parvenir au même résultat lorsqu'une tâche est exécutée sur une machine virtuelle (et qu'il est donc encore plus facile de parvenir à un déni de service). Cet aspect, bien que non crucial dans le cas de notre simulation, nous apparait pourtant comme un paramètre clé dans la pratique, et fait partie intégrante de la spécificité des réseaux d'infonuagique. Par ailleurs, nous n'avons pas associé de complexité (en termes de temps et d'espace) à notre système de défense. Là encore, cet aspect est extrêmement important dans la pratique, puisque certaines défenses mal implémentées pourraient devenir de nouveaux vecteurs d'attaques (comme c'est le cas par exemple du module WS-Security utilisé par Axis2, et qui construit l'ensemble du document en mémoire, alors même qu'Axis2 travaille sur des flots de données et non sur le document au complet). Enfin, les mécanismes tels que l'"autoscaling" ou le "throttling" n'ont pas été implémentés, mais il aurait été intéressant de mesurer leur efficacité à éviter que certains équipements, tels que le "broker", soient surchargés.

Nous aurions aussi pu utiliser des techniques comme le "honeypot" pour leurrer les attaquants et analyser des caractéristiques ou des comportements suspects, et ainsi avoir un meilleur contrôle des utilisateurs.

D'autre part, la topologie considérée est très simplifiée et ne rend pas forcément compte des systèmes utilisés dans l'industrie. Notamment, nous n'avons pas pris en compte (car le simulateur ne le prenait pas en charge) la réplication, qui est souvent un requis de base

des centres de données. Or cette réplication a une influence importante sur l'impact que peut avoir une attaque de déni de service. D'autres topologies sont par ailleurs en train de remplacer la topologie hiérarchique présentée dans ce travail. La topologie "Leaf-Spine" par exemple, qui présente moins de goulots d'étranglement.

La principale difficulté rencontrée a été la compréhension du simulateur GreenCloud. Aucune documentation technique n'est fournie (et ce simulateur ne jouit pas encore d'une grande communauté), et le code n'est presque pas commenté. Cela a donc été un réel défi de comprendre comment les différents objets interagissaient entre eux, quelles étaient les dépendances, par quelles étapes passaient une requête, et comment mesurer les ressources des machines. Le débogage a également été très fastidieux, étant donné le grand nombre de fichiers sources impliqués dans la simulation, et l'appel régulier des fichiers de scripts TCL qui complique encore la compréhension. Néanmoins, ayant pris l'initiative de contacter le principal développeur de ce simulateur, j'ai pu obtenir de précieuses informations qui m'ont considérablement aidé dans ma compréhension.

## 7.3 Améliorations futures

Les travaux futurs visent principalement à améliorer les limitations identifiées à la section précédente. Notamment, il faudrait ajouter au simulateur une modélisation pour le surcoût de performance lié à l'utilisation de machines virtuelles, ainsi qu'une façon de mesurer le coût d'exécution de la défense. Par ailleurs, nous pourrions développer davantage l'aspect qualité de service, étant donné que l'infonuagique est une architecture orientée services. Nous n'avons utilisé que peu de métriques (diminution de la part de CPU accordé, chute du nombre de paquets traités), et elles pourraient être complétées par de nombreuses autres, comme l'augmentation de la latence, si les serveurs géographiquement proches de l'utilisateur sont rendus indisponibles à la suite d'une attaque XML-DoS. La modélisation de l'interférence de performance entre machines virtuelles situées sur une même machine physique devrait aussi être améliorée pour être plus réaliste, avec éventuellement une façon de rendre compte de la compétition pour le cache qui peut causer de grands ralentissements lors d'une attaque. Enfin, il pourrait être intéressant de modéliser plus fidèlement ce qui fait la spécificité de chacun des modèles d'infonuagique : IaaS, PaaS et SaaS.

# RÉFÉRENCES


[1] Amazon elastic compute cloud. http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/concepts.html.

[2] J. Antunes, N.F. Neves, and P.J. Verissimo. Detection and prediction of resource-exhaustion vulnerabilities. In *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, pages 87–96, Nov 2008.

[3] Jerry Archer, Alan Boehme, Dave Cullinane, Nils Puhlmann, Paul Kurtz, and Jim Reavis. Secaas,implementation guidance category 1, identity and access management, 2012. https://downloads.cloudsecurityalliance.org/initiatives/secaas/SecaaS_Cat_1_IAM_Implementation_Guidance.pdf.

[4] Aws elastic beanstalk documentation. https://aws.amazon.com/documentation/elastic-beanstalk/.

[5] 2009. http://www.sensepost.com/blog/3797.html,.

[6] A. Chonka and J. Abawajy. Detecting and mitigating hx-dos attacks against cloud web services. In *Network-Based Information Systems (NBiS), 2012 15th International Conference on*, pages 429–434, Sept 2012.

[7] Ashley Chonka, Yang Xiang, Wanlei Zhou, and Alessio Bonti. Cloud security defence to protect cloud computing against http-dos and xml-dos attacks. *Journal of Network and Computer Applications*, 34(4) :1097 – 1107, 2011. Advanced Topics in Cloud Computing.

[8] Survey on cloud simulators. http://fr.slideshare.net/habibur01/survey-on-cloud-simulator.

[9] Cloudsim : A framework for modeling and simulation of cloud computing infrastructures and services. http://www.cloudbus.org/cloudsim/.

[10] Document object model (dom). http://www.w3.org/DOM/.

[11] Diogo A. B. Fernandes, Liliana F. B. Soares, João V. P. Gomes, Mário M. Freire, and Pedro R. M. Inácio. Security issues in cloud environments : a survey. *Int. J. Inf. Sec.*, 13(2) :113–170, 2014.

[12] Armando Fox, Rean Griffith, A Joseph, R Katz, Andrew Konwinski, Gunho Lee, D Patterson, Ariel Rabkin, and Ion Stoica. Above the clouds : A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28 :13, 2009.

[13] N. Gonzalez, C. Miers, F. Redigolo, T. Carvalho, M. Simplicio, G.T. de Sousa, and M. Pourzandi. A quantitative analysis of current security concerns and solutions for cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 231–238, 2011.

[14] Google apps for work. `https://www.google.com/work/apps/business/`.

[15] Greencloud - the green cloud simulator. `http://greencloud.gforge.uni.lu`.

[16] B. Grobauer, T. Walloschek, and E. Stocker. Understanding cloud computing vulnerabilities. *Security Privacy, IEEE*, 9(2) :50 –57, march-april 2011.

[17] N. Gruschka and L.L. Iacono. Vulnerable cloud : Soap message security validation revisited. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 625–631, July 2009.

[18] Nils Gruschka and Norbert Luttenberger. Protecting web services from dos attacks by soap message validation. In *in Proceedings of the IFIP TC11 21 International Information Security Conference (SEC)*, 2006.

[19] Keiko Hashizume, David Rosado, Eduardo Fernandez-Medina, and Eduardo Fernandez. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, 4(1) :5, 2013.

[20] Keiko Hashizume, David Rosado, Eduardo Fernandez-Medina, and Eduardo Fernandez. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, 4(1) :5, 2013.

[21] Kai Hwang, S. Kulkareni, and Yue Hu. Cloud security with virtualized defense and reputation-based trust mangement. In *Dependable, Autonomic and Secure Computing, 2009. DASC '09. Eighth IEEE International Conference on*, pages 717 –722, dec. 2009.

[22] Cory Janssen. `http://www.techopedia.com/definition/29545/resource-pooling`.

[23] Javascript.com. `https://www.javascript.com`.

[24] M. Jensen, J. Schwenk, N. Gruschka, and L.L. Iacono. On technical security issues in cloud computing. In *Cloud Computing. CLOUD '09. IEEE International Conference on*, pages 109 –116, 2009.

[25] 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, 2009.

[26] T. Karnwal, T. Sivakumar, and G. Aghila. A comber approach to protect cloud computing against xml ddos and http ddos attack. In *Electrical, Electronics and Computer Science (SCEECS), 2012 IEEE Students' Conference on*, pages 1–5, March 2012.

[27] A Karthigeyan, C Andavar, and A Jaya Ramya. Adaptable practices for curbing xdos attacks. *International Journal of Scientific & Engineering Research*, 3, June 2012.

[28] Issa M Khalil, Abdallah Khreishah, and Muhammad Azeem. Cloud computing security : a survey. *Computers*, 3(1) :1–35, 2014.

[29] 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 Gener. Comput. Syst.*, 28(6) :833–851, June 2012.

[30] Younggyun Koh, R. Knauerhase, P. Brett, M. Bowman, Zhihua Wen, and C. Pu. An analysis of performance interference effects in virtual environments. In *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 200–209, April 2007.

[31] J. Latanicki, P. Massonet, S. Naqvi, B. Rochwerger, and M. Villari. Scalable cloud defenses for detection, analysis and mitigation of ddos attacks. In *Towards the Future Internet*, pages 127–137. 2010.

[32] Huan Liu. A new form of dos attack in a cloud and its avoidance mechanism. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, CCSW '10, pages 65–76, New York, NY, USA, 2010. ACM.

[33] Meng Liu, Wanchun Dou, Shui Yu, and Zhensheng Zhang. A clusterized firewall framework for cloud computing. In *Communications (ICC), 2014 IEEE International Conference on*, pages 3788–3793, June 2014.

[34] Alina Madalina Lonea, Daniela Elena Popescu, and Huaglory Tianfield. Detecting ddos attacks in cloud computing environment. In *International Journal Computer Communication*, February 2013.

[35] Rafel Los, Don Gray, Dave Shackleford, and Bryan Sullivan. The notorious nine : Cloud computing top threats in 2013. In *CSA, Cloud Security alliance*, February 2013. `https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf`.

[36] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1) :42 – 57, 2013.

[37] The network simulator - ns-2. `http://www.isi.edu/nsnam/ns/`.

[38] Office 365 for business. `https://products.office.com/en-ca/business/get-latest-office-365-for-your-business-with-2016?WT.mc_id=PS_Google_O365SMB_office%20365_Text&WT.srch=1`.

[39] S. Padmanabhuni, V. Singh, K.M. Senthil Kumar, and A. Chatterjee. Preventing service oriented denial of service (presodos) : A proposed approach. In *Web Services, 2006. ICWS '06. International Conference on*, pages 577–584, Sept 2006.

[40] Maxim Raya and Hubaux Jean-Pierre. Securing vehicular ad hoc networks. *Journal of Computer Security*, 15 :39–68, 2007.

[41] D. Riquet, G. Grimaud, and M. Hauspie. Large-scale coordinated attacks : Impact on the cloud security. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 558–563, July 2012.

[42] 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 *CCS '09 : Proceedings of the 16th ACM conference on Computer and communications security*, 2009. `http://people.csail.mit.edu/tromer/papers/cloudsec.pdf`.

[43] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[44] Mohammed A. Saleh and Azizah Abdul Manaf. A novel protective framework for defeating http-based denial of service and distributed denial of service attacks. In *The Scientific World Journal*, 2015.

[45] Choosing between sax and dom. `https://msdn.microsoft.com/en-us/library/ms766563(v=vs.85).aspx`.

[46] Sax. `http://sax.sourceforge.net`.

[47] T. Sridhar. Cloud computing : Infrastructure and implementation topics. In *The Internet Protocol Journal of CISCO*, december 2009.

[48] Bryan Sullivan. Xml denial of service attacks and defenses. `https://msdn.microsoft.com/en-us/magazine/ee335713.aspx`.

[49] 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 2010.

[50] Thomas Vissers, Thamarai Selvi Somasundaram, Luc Pieters, Kannan Govindarajan, and Peter Hellinckx. Ddos defense system for web services in a cloud environment. *Future Generation Computer Systems*, 37 :37–45, 2014.

[51] Bing Wang, Yao Zheng, Wenjing Lou, and Y.T. Hou. Ddos attack protection in the era of cloud computing and software-defined networking. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 624–629, Oct 2014.

[52] WS-Attacks.org. Coercive parsing. http://www.ws-attacks.org/index.php/Coercive_Parsing.

[53] Xml tutorial. http://www.w3schools.com/xml/.

[54] Xinfeng Ye. Countering ddos and xdos attacks against web services. In *Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on*, volume 1, pages 346–352, Dec 2008.

[55] Shui Yu, Yonghong Tian, Song Guo, and D.O. Wu. Can we beat ddos attacks in clouds ? *Parallel and Distributed Systems, IEEE Transactions on*, 25(9) :2245–2254, Sept 2014.

[56] Siqin Zhao, Kang Chen, and Weimin Zheng. Defend against denial of service attack with vmm. In *Grid and Cooperative Computing, 2009. GCC '09. Eighth International Conference on*, pages 91–96, Aug 2009.

[57] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation Computer Systems*, 28(3) :583 – 592, 2012.