Generation and Dynamic Update of Attack Graphs in Cloud Providers Infrastructures

Pernelle Mensah

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

CentraleSupélec

Thèse de Doctorat

Generation and Dynamic Update of Attack Graphs in Cloud Providers Infrastructures

Par :
Pernelle MENSAH

CentraleSupélec

COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Thèse présentée et soutenue à Rennes, le 13 Juin 2019
Unité de recherche : IRISA (UMR 6074)
Thèse N° : 2019-06-TH

Rapporteurs avant soutenance :

Eddy CARON	Maître de Conférences, HDR, ENS Lyon/LIP
Vincent NICOMETTE	Professeur à INSA Toulouse/LAAS-CNRS

Composition du Jury :

Examinateurs :	Hervé DEBAR	Professeur à Telecom SudParis
	Marc LACOSTE	Ingénieur à Orange Labs
	Eddy CARON	Maître de Conférences, HDR, ENS Lyon/LIP
	Vincent NICOMETTE	Professeur à INSA Toulouse/LAAS-CNRS
Dir. de thèse :	Eric TOTEL	Professeur à CentraleSupélec
Co-dir. de thèse :	Christine MORIN	Directrice de Recherche Inria
Encadrants :	Guillaume PIOLLE	Maître de Conférences à CentraleSupélec
	Samuel DUBUS	Responsable du Département ESTA à Nokia Paris-Saclay

Résumé : Génération et mise à jour dynamique de graphes d'attaque dans les infrastructures des prestataires d'informatique en nuage

L'avènement du Cloud a profondément transformé la manière dont les utilisateurs et les entreprises appréhendent les services informatiques. Avec la virtualisation comme pilier central, le Cloud permet des gains économiques conséquents en permettant de louer les ressources matérielles sous-utilisées à de multiples clients. Il met à disposition de ces clients des ressources informatiques et des capacités d'automatisation seulement limitées par les capacités des fournisseurs d'environnement Cloud. Ce faisant, il contribue à la destruction des barrières à l'entrée par la dynamisation du tissu économique, en permettant aux petites, moyennes et grandes entreprises de réduire les coûts budgétaires et de se concentrer sur leurs différentiateurs de base, par l'externalisation de la gestion de l'infrastructure. Cependant, les systèmes de communication modernes représentent des infrastructures critiques dont la croissance constante en taille et en complexité menace grandement la sécurité. La prolifération et la sophistication des attaques modernes nécessite d'analyser l'ensemble de l'infrastructure dans le but de découvrir les différentes étapes utilisées par un attaquant afin de pénétrer le réseau. Ces compromissions ont de graves implications dans le contexte du Cloud, qui représente une cible de grande valeur pour les attaquants, car il regroupe les actifs de plusieurs entreprises. L'amélioration de la sécurité dans les environnements informatiques traditionnels a pu reposer avec succès sur l'utilisation des graphes d'attaque couplée à une méthodologie de gestion des risques. Nous envisageons donc d'appliquer une approche similaire dans le contexte du Cloud. Cela implique de prendre en compte les nouveaux défis rencontrés dans ces infrastructures modernes, la majorité des méthodes de graphe d'attaque ayant été conçue pour des environnements traditionnels.

En premier lieu, il est nécessaire de considérer l'existence de nouveaux scénarios d'attaque dans le Cloud. En effet, les relations existant entre machines virtuelles et machines physiques ainsi que les vulnérabilités présentes dans le logiciel de virtualisation affectent la propagation d'une attaque, permettant potentiellement à un attaquant de contourner l'isolation fournie par l'hyperviseur et d'infiltrer les machines virtuelles voisines, ou même la machine hôte. La capacité pour un attaquant de tirer parti d'une vulnérabilité particulière pouvant être entravée par la configuration des machines physiques ou virtuelles, il est essentiel d'identifier les conditions de succès de l'attaque, pour une meilleure évaluation des actions d'attaque possibles et une extension des modèles de graphes existants, qui ne prennent généralement pas en compte l'aspect virtualisation.

D'autre part, la construction d'un graphe d'attaque adapté au Cloud doit tenir compte non seulement de sa nature dynamique, mais également de son échelle. En effet, les propriétés

inhérentes au Cloud, à savoir son élasticité et son dynamisme sont un sujet de préoccupation. Ce contexte impose de concevoir des algorithmes capables de faire face aux fréquents changements d'infrastructure dans les environnements virtualisés. Le but de cette démarche est de fournir à l'administrateur de Cloud un graphe d'attaque illustrant la vision la plus précise du niveau de sécurité actuel, afin de prendre les mesures correctives adéquates. Compte tenu de l'échelle considérée, ainsi que de la multiplicité des utilisateurs impliqués dans le Cloud, la clarté de la représentation du graphe d'attaque est aussi essentielle afin d'aider les administrateurs sécurité à visualiser et à comprendre rapidement les chemins d'attaque existants dans leur environnement. De plus, pour réduire les coûts de calcul, le graphe ne doit pas nécessiter une transformation intermédiaire avant d'être exploitable par les algorithmes d'analyse de risque. Pour réaliser ces différents objectifs, il est nécessaire d'identifier dans quelle mesure l'utilisation des technologies disponibles dans le Cloud, à savoir les systèmes de gestion du Cloud ou le SDN (Software-Defined Networking), peuvent aider à relever les défis présentés plus tôt, avec l'objectif final de contribuer non seulement à la construction des graphes d'attaque, mais aussi de contribuer à les maintenir à jour.

Divers modèles de graphes d'attaque sont disponibles dans la littérature, pour une application aux environnements traditionnels. Certaines approches considèrent l'ensemble des états du réseau à chaque nœud, quand d'autres considèrent des états partiels, se centrent soit sur les actions d'attaques ou soit sur les hôtes. Le fait de considérer l'ensemble des états du réseau au sein d'un seul nœud résulte en une complexité exponentielle, ce qui nuit à l'analyse du graphe d'attaque. Cependant, trop fragmenter les informations en des nœuds distincts est également nuisible, du fait de la représentation de données inutiles. Malgré une approche entièrement automatisée, un utilisateur de graphe d'attaque peut avoir besoin de visualiser les chemins d'attaque générés pour vérifier les mitigations proposées, ou comprendre l'anatomie des attaques à des fins d'audit. Il est donc nécessaire de trouver le bon équilibre dans la granularité de l'information représentée, exigence encore plus cruciale compte tenu de l'échelle du Cloud.

Un ensemble récurrent de données émerge de l'ensemble des algorithmes de génération de graphe d'attaque, avec des différences mineures introduites pour tenir compte des concepts particuliers des méthodes présentées. Ces méthodes reposent généralement, d'une part, sur un inventaire des vulnérabilités présentes dans l'infrastructure, et d'autre part, sur la connectivité réseau existant entre les équipements. Les approches traditionnelles considèrent généralement ces informations comme des données d'entrée, sans considérer la mise en œuvre de leur collecte. En effet, cette hypothèse est légitime dans ces environnements, car l'infrastructure y est essentiellement statique et peu sujette aux changements. Cependant, afin de concevoir une approche efficace de génération de graphes d'attaque dans le Cloud, il est nécessaire d'examiner comment ces données essentielles sont influencées par ce nouveau contexte. Concernant spécifiquement la connectivité réseau, il faut s'attendre à un taux de changement en adéquation avec la grande flexibilité fournie aux utilisateurs. En effet, créer, migrer ou supprimer des machines virtuelles, impacte directement la connectivité, qui doit donc régulièrement être mise à jour. La notion de co-localisation peut également être introduite, elle concerne les machines virtuelles hébergées sur le même hyperviseur. Des machines co-localisées peuvent disposer d'un canal de communication potentiel si la propriété d'isolation de l'hyperviseur fait défaut, en raison de l'existence de vulnérabilités dans le logiciel de virtualisation. L'existence de ce canal de communication est impactée par la migration de la machine virtuelle, d'un hyperviseur sécurisé à un hyperviseur

vulnérable, et vice versa.

Au cours de cette thèse, nous réalisons un ensemble de contributions permettant de rendre l'utilisation des graphes d'attaque viable dans le contexte du Cloud.

La première concerne la récupération de la topologie et de la connectivité du Cloud. En effet, ces données sont nécessaires au processus de génération du graphe d'attaque. Ceci nécessite la création d'un graphe de connectivité entre les machines virtuelles, que nous pouvons reconstruire grâce à la connaissance de la topologie globale et de la politique de sécurité réseau déployée. Une nouvelle approche s'impose dans le Cloud car les méthodes d'obtention de la connectivité dans les réseaux traditionnels sont inadaptées aux infrastructures Cloud , en raison de leur caractère intrusif ou de leur non complétude dans la découverte de connectivité. Nous proposons une méthode pour calculer le graphe de connectivité, qui repose à la fois sur les informations fournies par la plate-forme de gestion du Cloud et le contrôleur SDN. La connectivité peut d'abord être extraite des bases de connaissances, puis dynamiquement mise à jour lors de l'occurrence d'événements liés au Cloud.

Cependant, l'intégration de la plate-forme de gestion du Cloud et du contrôleur SDN pose un défi supplémentaire. En effet, le SDN permet aux administrateurs de déployer des applications dans le contrôleur. Ces applications peuvent alors, selon la logique programmée, modifier de manière réactive les règles de flux sur les commutateurs virtuels et impacter directement la topologie, sans aucun retour d'information à la plate-forme de gestion du Cloud. Il en résulte des incohérences de topologie entre le système de gestion de Cloud et le contrôleur SDN. Sans résolution, ces divergences entraîneraient une génération de graphes d'attaque incorrecte. Afin d'éviter ces inexactitudes nous nous appuyons sur une vue de la topologie et de la connectivité qui fusionne les informations de la plate-forme de gestion du Cloud et du contrôleur SDN.

Nous privilégions une approche passive pour la reconstruction de la connectivité sur la base des configurations extraites de la base de données du système de gestion du Cloud. Cela permet d'engager moins de perturbations sur les infrastructures virtuelles des clients, tout en rassemblant les informations pertinentes pour la construction du graphe de connectivité. Deux phases permettent l'obtention de ce graphe. Pendant la phase statique, l'objectif est d'établir une topologie et une connectivité de base, telle qu'elle est configurée au moment du fonctionnement de l'algorithme. S'ensuit alors une phase dynamique au cours de laquelle les évènements de modifications sont interceptés et traités afin de mettre à jour le graphe de manière localisée. En parallèle, une application de monitoring est déployée dans le contrôleur SDN afin de repérer les changements de connectivité non initiés par la plate-forme de Cloud et les intégrer dans le graphe de connectivité final.

Notre deuxième contribution concerne l'inclusion des vulnérabilités liées à virtualisation dans les graphes d'attaque. Pour ce faire, nous avons d'abord inspecté les descriptions de ces vulnérabilités présentes dans les bases de données de vulnérabilité existantes, afin d'en extraire les conditions nécessaires à un attaquant pour pouvoir les exploiter. En outre, nous évaluons également la portée exacte des impacts de ces vulnérabilités, une fois utilisées avec succès dans une attaque. Cette information est une étape cruciale pour étendre les modèles de vulnérabilité proposés dans les approches existantes pour prendre en compte l'impact de la virtualisation sur les scénarios d'attaque générés. Cela permet de comprendre précisément comment les faiblesses de l'hyperviseur peuvent être utilisées dans le but de compromettre la machine physique sous-jacente ou les machines virtuelles hébergées.

Cependant, tous les hyperviseurs ne bénéficient pas du même niveau de minutie dans la description des vulnérabilités qui les affectent. En conséquence, des informations pertinentes peuvent être manquantes, ce qui rend plus difficile la comparaison des résultats sur plusieurs logiciels. Néanmoins, les exigences et les catégories d'impact extraites de notre analyse nous ont permis d'étendre le champ d'application des vulnérabilités prises en compte dans la construction du graphe d'attaque jusqu'à la couche de virtualisation.

Il ressort de l'analyse de ces vulnérabilités que le succès de leur exploitation est dépendant de critères liés à la fois à l'hyperviseur, à la machine physique, à la machine virtuelle et à l'attaquant. Les pré-requis identifiés au cours de cette étude sont les suivants : la version de l'hyperviseur, ses outils de gestions, les options activées pour l'hyperviseur et les machines virtuelles, les quotas renseignés pour les machines virtuelles, les architectures des processeurs de la machine physique, les fabriquants des processeurs, les systèmes d'exploitation des machines physiques et virtuelles, les types de machine virtuelle, les périphériques autorisés pour les machines virtuelles et leurs drivers, et enfin, le niveau de privilège de l'attaquant.

Ainsi, des exigences très spécifiques doivent être satisfaites avant qu'un attaquant ne puisse exploiter efficacement l'une de ces vulnérabilités et compromettre l'infrastructure. Une compréhension claire de ces exigences est donc précieuse car elle permet une modélisation plus précise des vulnérabilités. Cette modélisation précise s'avère bénéfique, car intégrer des vulnérabilités qui ne peuvent pas être exploitées par un attaquant a le potentiel de créer un plus grand nombre de liens inutiles que dans les environnements traditionnels, l'impact pouvant être immédiat sur l'ensemble des machines hébergées par un hyperviseur vulnérable.

Troisièmement, nous proposons une modélisation hybride de graphe d'attaque. Afin de générer des graphes d'attaque qui sont non seulement complets mais suffisamment simplifiés pour gérer l'échelle du Cloud, nous tirons parti des avancées réalisées dans les modèles de graphes d'attaque antérieurs. Nous abordons cette question en proposant une extension du modèle de graphe d'attaque centré sur les hôtes de l'infrastructure, dans laquelle les sommets représentent les hôtes physiques et virtuels existants dans le Cloud, couplés, le cas échéant, aux états de l'attaquant et de la machine. La représentation choisie conserve la possibilité d'utiliser des algorithmes issus de la théorie des graphes pour l'évaluation des risques, sans nécessiter une transformation intermédiaire du graphe d'attaque.

Fort des résultats obtenus sur l'extraction de la topologie et de la connectivité du Cloud, de l'extension de la base de vulnérabilités pour inclure celles liées à la virtualisation et du modèle de graphe d'attaque établi pour le Cloud, nous disposons de l'ensemble des données nécessaires pour nous atteler à la génération du graphe d'attaque. En utilisant un algorithme de construction basé sur les événements, le graphe d'attaque obtenu correspond à l'état de sécurité de l'infrastructure le plus actuel à tout instant. Les données contenues dans les messages de notification d'événements sont analysées pour déterminer leurs impacts, soit sur la connectivité, soit les vulnérabilités existantes et, par conséquent, le delta à répercuter sur le graphe d'attaque. En supervisant en permanence les modifications et en adaptant le graphe d'attaque en conséquence, nous sommes capable d'éviter de le reconstruire à partir de zéro à chaque fois qu'un changement se produit.

Le prototype conçu pour évaluer notre proposition dans un environnement Cloud réel avec des charges de travail plausibles présente des perspectives prometteuses au niveau des performances et confirme les avantages et la faisabilité de notre approche. Les approches mises en oeuvre dans cette thèse permettent donc aux admistrateurs de Cloud de disposer d'un outil d'évaluation des risques dans leur infrastructure, permettant une priorisation des contre-mesures à apporter pour une amélioration de la sécurité.

Remerciements

Je tiens à remercier mes encadrants de thèses Eric TOTEL, Christine MORIN et Guillaume PIOLLE pour leurs regards critiques et constructifs sur le travail effectué, contribuant ainsi grandement à son amélioration.

Un remerciement particulier à Guillaume PIOLLE pour ses relectures assidues du manuscrit et ses conseils de modélisation.

De même, je remercie mon encadrant de thèse à Nokia Paris-Saclay, Samuel DUBUS, pour le partage de ses connaissances, les nombreuses heures d'échanges et de discussions qui m'ont permis de progresser, ainsi que pour son soutien constant tout au long de cette thèse.

Je souhaite également remercier l'ensemble des membres du jury qui m'ont fait l'honneur d'évaluer ce travail. Hervé DEBAR, pour l'opportunité qu'il m'a offerte de pouvoir réaliser une thèse. Marc LACOSTE, Eddy CARON et Vincent NICOMETTE pour leurs regards extérieurs, leurs relectures minutieuses de ce document et leurs remarques pertinentes.

Je remercie également mes collègues de travail pour leur support et leur aide, aussi bien par des discussions fructueuses que par leur soutien moral. Ainsi, un grand merci à Serge PAPILLON, Haithem EL ABED, Waël KANOUN et Siwar KRIAA.

Je remercie l'ensemble de l'équipe sécurité à Nokia Paris-Saclay pour son accueil chaleureux, ses conseils et sa bonne humeur.

En dernier lieu, tous mes remerciements à mes parents Sidonie et Augustin, ma sœur Karell et mon frère Dietrich, qui en dépit de la distance ont su m'apporter un soutien permanent et indéfectible.

Je ne saurais oublier mes amis qui m'ont supportée et encouragée durant ces longues années de travail.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

The advent of the Cloud triggered a transformation in the way users and companies experience IT services. With virtualization as its core pillar, the Cloud enables economical gains by allowing the lease of underutilized physical server resources to multiple customers, also called tenants. By providing virtually unlimited computing resources and automation capabilities, it contributes to the shattering of barriers to entry. It fuels market disruption by allowing start-ups, small, medium and large companies alike to reduce budget costs by offloading infrastructure management to third-parties and focus on their core business differentiators. Over the years, we witnessed a tremendous growth in Cloud computing usage, as evidenced on Figure 1.1 by Gartner's 5-years revenues predictions [14, 15]. In 2017 and 2018, they performed 5 years revenue predictions for Infrastructure, Platform and Software as a Service (IaaS, PaaS and SaaS respectively). The market revenue is supposed to reach between 27 and 117 billions dollars by 2021 depending on the service type, which makes it a domain in full expansion.

1.1 Challenges Introduced by the Cloud

In this section, we will present the challenges posed by the Cloud, as well as some real-life security incidents in this environment.

1.1.1 The Cloud as a Virtual Environment

The key enabler of Cloud computing is virtualization, which comes with its fair share of challenges [16–18]. As a result, difficulties experienced in virtualized environments reverberate directly on the Cloud. We introduce in this section a number of security concerns due to the properties exhibited by virtualized environments.

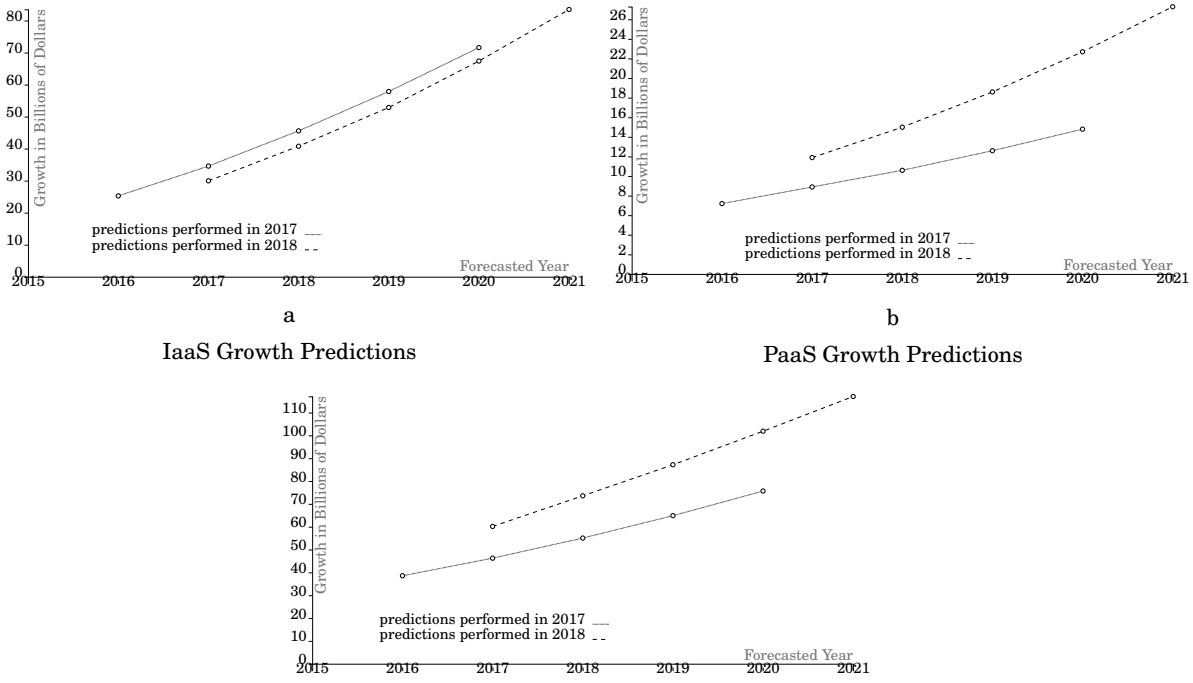


Figure 1.1: Gartner's revenues growth predictions for Cloud services.

The first security concern is scaling. Indeed, virtualization offers flexibility to the users for the instantiation of the virtual machines constituting their virtual infrastructures. As a result, the workload in these environments is highly dynamic and unpredictable. This ease of deployment can lead to virtual machines (VMs) sprawl. As a result, the quantity of VMs may exceed the management abilities of the administrators, when some administrative tasks such as upgrades, patch management and configuration still remain a combination of automated and manual processes in current networks.

VMs transient states represent an additional challenge. Special purpose machines (e.g., test, development or production machines) can easily be provisioned in a virtual environment. As a consequence, users can switch between them according to their current needs, deciding to either stop, launch, pause, restart, etc., the idle ones, depending on the capabilities provided by the virtualization software. These ephemeral states are an hindrance to the enforcement of security measures in virtualized environments, as knowledge about the network becomes rapidly obsolete. Additionally, the detection of vulnerabilities or application of patches is not possible for offline machines, leaving lingering weaknesses and security holes in the infrastructure.

The third issue is diversity. Even if at the scale of a particular Cloud user, the virtual machines can follow a certain homogeneity in the operating systems deployed and running services, a great heterogeneity can be observed at the Cloud macro level. Besides, more often than not, even inside

a single user's infrastructure, variety is noticeable due to the ease of deploying virtual machines with older versions for custom purposes. Even after an update, VMs can still be rolled back to previously unpatched states. This requires stricter patch enforcement policy and upgrade cycle to manage the risk of running vulnerable equipment.

Cloud environment must also face the mobility issue. Due to maintenance or workload consolidation, virtual machines can be migrated from one physical host to another. In that situation, not only can the virtual machine be intercepted during its transfer to the destination host [19], it can also see an increase in the number of vulnerabilities available to an attacker. Indeed, the ability of an attacker to exploit virtualization software vulnerabilities relies on specific requirements involving the configuration and characteristics of both physical and virtual machines. Thus, the fact of moving a previously safe virtual machine to a different physical server might create the required conditions for an exploit.

Concerning network monitoring in the context of virtualization, virtual network configurations can change dynamically and are no longer static, which makes it difficult to track changes occurring in the infrastructure. As traditional monitoring approaches are generally implemented at the granularity of physical servers, they are unable to monitor inter virtual machine traffic on the same physical infrastructure. This implies the use of novel methods for this objective.

1.1.2 Real-life Security Incidents in the Cloud

Despite the public's growing confidence in the security of the Cloud [20], the fact remains that, similar to traditional infrastructures, this environment is subject to security incidents, and the shift to shared infrastructures yields various security concerns. On one hand, virtualization vulnerabilities add up to the ones in traditional environments, hence increasing the attack surface in a context growing ever more complex. This complexity can be explained by multi-tenancy, the scale of the infrastructure and mobility of virtual servers and networks. Weaknesses in virtualization software (hypervisors) have potentially critical impacts as they can break the isolation property and enable attackers to steal confidential data or cause Denial of Service (DoS) attacks. On the other hand, hardware vulnerabilities, that would otherwise be confined to equipment inside a single company, have an increased dangerousness. Indeed, due to the promiscuity of companies in the Cloud, they enable attackers to impact assets of different organizations located on the same vulnerable physical server, as the largely publicized Meltdown [21] and Spectre [22] vulnerabilities illustrate. Additionally, beyond the inherent bugs existing in the hypervisor code base or hardware acceleration instructions, a trivial misconfiguration of the virtualized infrastructure can have dramatic impacts on security. For illustration purposes, we briefly present some real-life vulnerabilities and misconfigurations threatening the security of the Cloud.

Configuring traditional environments is already a challenging task, even for experts, but with

an infrastructure that delegates such flexibility to end users, errors of configurations are bound to happen, especially when these users are neophytes.

Dow Jones, Accenture, FeDex and many more companies*[1] *customers data exposed: These incidents are a mere illustration of the plethora of similar security mishaps witnessed over the recent years in the usage of Cloud services. In all those cases, the common denominator turns out to be incorrectly configured Cloud-hosted databases left publicly accessible by the infrastructure managers. This leads to the exposure of either business critical data or sensitive personal information, like medical customers data.

VENOM*[2] *(Virtualized Environment Neglected Operations Manipulation): This vulnerability existed in the QEMU (Quick EMUlator) code base since 2004, but was only discovered in 2015 by Jason Geffner, senior security researcher at CrowdStrike at the time. He described the vulnerability as follows:

> "[It] is a security vulnerability in the virtual floppy drive code used by many computer virtualization platforms. This vulnerability may allow an attacker to escape from the confines of an affected VM guest and potentially obtain code-execution access to the host. Absent mitigation, this VM escape could open access to the host system and all other VMs running on that host, potentially giving adversaries significant elevated access to the host local network and adjacent systems."

Compared to previous virtual machines escape vulnerabilities, such as Cloudburst [23] in 2009, the criticality of this one is much higher. Indeed, since the QEMU emulator is cross-virtualization platforms, i.e., it can be used by several vendors' hypervisors, so is the compatibility of VENOM. Additionally, it can be directly exploited in default configurations, independently from the underlying host operating system and allows direct arbitrary code execution.

Several security advisories with similar consequences have been released concerning the Xen hypervisor[3], a classic virtualization software used by Cloud providers. The Xen Security Advisory (XSA) ***XSA-213***[4] is one of them. It was disclosed by the security team of Qubes OS, a security-oriented operating system based on Xen for application compartmentalization. This vulnerability involves the memory virtualization of Xen, and enables user hypercalls to be run as kernel hypercalls (a software trap from a virtual machine to the hypervisor, just as a syscall is a software trap from an application to the kernel), leaving the guest with write-access on otherwise protected memory pages. It only affects a certain type of virtual machines, which use software-based virtualization instead of hardware-assisted virtualization.

[1] https://www.peerlyst.com/posts/publicly-accessible-amazon-s3-buckets-a-list-of-Cloud-misconfiguration-breaches-claus-cramon

[2] http://venom.crowdstrike.com/

[3] https://www.xenproject.org/

[4] https://xenbits.xen.org/xsa/advisory-213.html

Meltdown and Spectre[5]***:*** These are the same family of hardware vulnerabilities affecting modern processors, namely every processor made in the last 20 years. They are based on branch prediction and speculative execution, which are optimization mechanisms allowing processors to perform tasks before they are needed: if a processor is executing a set of instructions that branches depending on the input, then it will try to guess which branch is most likely to be executed and load the necessary data into its cache. These mechanisms are leveraged by the Spectre attack, in which the attacker tricks the processor into loading a value from protected memory into the cache, then attempts to load known data from unprotected memory. If one piece of this known data loads far more rapidly than the others, then they can infer that this data is being retrieved from the cache, and therefore is related to the value stored in protected memory, hence breaking the isolation between programs.

On the other hand, Meltdown works slightly differently by breaking the isolation between the applications and the operating system. It exploits a privilege escalation flaw that allows users to access protected memory. These vulnerabilities jeopardize the sharing principle of the Cloud, as they can allow, in theory, a Cloud user to access the data of another one hosted on the same physical server.

1.2 Attack Graphs: A Model for Risk Analysis

In such a context, understanding the security risks faced by Cloud providers and individual tenants due to vulnerability exploitation is imperative.

The primary approach to ensuring the security of a network or of a piece of equipment is the identification of its weaknesses. This is oftentimes obtained by vulnerability scanning, which leads to an inventory listing known vulnerabilities present on the devices. However, addressing the vulnerabilities in the sole context of the concerned device is far from sufficient. Indeed, starting from a vulnerable machine, it is customary for attackers to use it as *foothold* or *plant*, from which they can pivot and further compromise the targeted network. They can not only scavenge for more accesses or weaknesses on the current machine or take advantage of its interconnections with other systems to remotely exploit the vulnerabilities of these distant pieces of equipment. It is critical for security practitioners to have an indication on the ways existing vulnerabilities in the infrastructure can interact between each other for the benefit of an attacker, in order to be able to properly secure their network.

In traditional environments, attack graphs can contribute to that awareness, as they are a model allowing to depict the many steps an attacker can take to jeopardize an asset. They solve that issue by presenting in a compact form the ways in which an attacker may compromise identified assets. Beyond a basic enumeration of the attack paths existing in the network, several approaches used attack graphs generation as a basis for automated risk assessment [24–26],

[5] https://meltdownattack.com/

relying on an identification and valuation of critical assets in the network. This association allows security administrators to design pro-active and reactive counter-measures for risk mitigation and can be leveraged for security monitoring and network hardening.

As presented by Jha *et al.* [27], attack graphs can be valuable as a reactive defense mechanism. By correlating alerts generated by Intrusion Detection Systems (IDS) with transitions existing in the graph (attacker actions), defenders increase their confidence in the identification of potentially ongoing attacks and can identify the probable attack goal.

In a proactive perspective, an inventory of all the attack paths in the infrastructure, leading to the compromise of a particular machine, permits to prioritize the remediation mechanisms for the vulnerabilities involved in the paths, according to the target valuation. Additionally, algorithms can be used to determine the minimum set of vulnerabilities to be removed in order to prevent attackers from reaching an asset. On the other hand, attack graphs allow to organize defense in depth, as transitions relative to the attacks used can be labeled with their IDS detectability. Hence, not only can the best IDS locations for enhanced coverage be determined, but also the suitable IDS to install at a given location in the network [28].

In a forensics context, alerts triggered by IDSs can also be analyzed a posteriori in combination with attack graphs models to uncover an attacker intent [29].

Through attack graphs and risk assessment, security practitioners have an overview of the global exposure of their infrastructure, which allows them to plan their mitigation actions. Indeed, software patches cannot always be deployed in due time given cost or operational constraints associated with continuity of service, or feature and version compatibility. This approach allows experts to incrementally apply the patches according to the restrictions dictated by their infrastructure. Thus, they can update the original attack graph to maintain an insight into the current exposure of their environment and analyze the impacts of the mitigation measures deployed [30].

1.3 Problem Statement

We showed in Section 1.1 that modern communication systems represent critical infrastructures whose continuous growth in size and complexity poses serious threats to their security. The proliferation and sophistication of modern attacks requires analyzing the infrastructure as a whole, to uncover the various steps used by an attacker in order to break into the network. These compromises have severe implications in the context of the Cloud, which is a high value target for attackers, as it aggregates the assets of multiple companies.

Since attack graphs coupled to risk management approaches have been considered successfully in traditional environments to enhance the infrastructures security, we envision to apply a similar approach in the Cloud context. This implies to consider new challenges incurred by these modern infrastructures, as the majority of attack graph methods were designed with traditional environments in mind. In the rest of this section, we introduce the ones we identified

and addressed during this thesis.

Novel attack vectors enabled by the hypervisor: First of all, shifting to a Cloud environment entails to consider novel attack scenarios. Indeed, it requires to address the relations existing between virtual and physical machines and identify how the characteristics of both types of equipment impact the propagation of an attack. Even if the ability to leverage a particular vulnerability can be hindered by the configuration of either the physical or virtual machine, it is critical to identify the conditions of the attack success, as it can result in a rogue access directly to the physical server, or to virtual machines co-located with the targeted virtual machines.

Building an attack graph with the following properties, namely an adaptation to the Cloud environment, as well as the ability to be automatically generated and updated: Addressing this issue requires to take into account two items:

- ***The dynamic nature of the Cloud:*** Indeed, inherent properties of the Cloud, namely elasticity and dynamism are a cause for concern. When previous attack graph approaches dealt with mostly static environments, the current context imposes to design algorithms able to cope with the frequent rate of change occurring in virtualized environments. The aim is to provide the Cloud administrator with an attack graph illustrating its most accurate security exposure, in order to take adequate mitigation actions;
- ***The Cloud scale:*** Given the scale considered, as well as the various ownerships involved in the Cloud, the clarity of representation for an attack graph model is key in order to help security practitioners visualize and quickly understand the attack paths existing in their environment. As such, the model must be tailored to only display the minimal set of information needed. In addition, to reduce computation costs, the graph must not require an intermediary transformation before being exploitable by the risk analysis algorithms.

In order to realize these different objectives, it is necessary to identify to what extents the use of technologies available in the Cloud such as management systems or SDN (Software-Defined Network) controllers can help in addressing the challenges presented in Section 1.1, with the final goal to contribute not only to the construction of attack graphs, but also help in keeping them in an up-to-date state.

1.4 Global Approach

In order to answer the issue introduced in Section 1.3, which is the adaptation of attack graphs to Cloud environments, we will present here an overview of the methods adopted in this thesis, which led to contributions pertaining to an event-based construction of attack graphs in the context of the Cloud.

As we will introduce in the state of the art in Section 2.2, attack graph construction relies on several type of information in order to provide an accurate view of the infrastructure:

- Firstly, an up-to-date inventory of the vulnerabilities present on the machines existing in the infrastructure with their associated modeling;
- Secondly, an up-to-date view of the topology and connectivity of these machines allowing to identify the ones able to communicate with each other.

In the Cloud context, these data have two distinct components which are on one hand the administrator-owned data and on the other hand, the customer-owned data. While the data from the administrator is relative to the physical infrastructure devices' vulnerabilities and connectivity, each customer detains a piece of information regarding the global map of the hosted virtual infrastructure, namely his virtual devices and virtual networks deployed within the administrator's infrastructure. With the separation of responsibilities existing in Cloud environments, administrators can only have a comprehensive view of the underlying infrastructure and its interconnections, and cannot theoretically access the tenants' virtual infrastructures.

In order to obtain such detailed information, administrators can either rely on the cooperation of their users, assuming that the latter have the ability and the will to provide such data, or they can try to gather the data from outside their tenants' infrastructure, by leveraging technologies available in the Cloud context. In doing so, it is paramount for administrators to incur the least amount of disruption and intrusiveness into their customers' environments, in order to maintain an acceptable quality of service.

In this thesis, we primarily focus on one aspect of the data retrieval, namely getting an up-to-date topology and connectivity view of the whole infrastructure, using Cloud technologies. Ideally, for a more comprehensive solution in Cloud environments, a similar effort should be devoted to obtaining the current inventory of devices vulnerabilities with as little interactions as possible with the users, by implementing vulnerability scanners based on virtual machine introspection [31–34] for instance, which is another complete area of research.

We will introduce in the following section the assumptions and choices that guided our reflection during this thesis.

1.4.1 Assumptions made in the context of this thesis

1.4.1.1 The Environment

We interest ourselves in the most flexible Cloud service model, which is the Infrastructure as a Service (IaaS) Cloud. As such, it is also the most challenging to apprehend, since the deployment, configuration and interconnection of virtual machines are under the responsibility of the tenants and not the one of the Cloud administrator. This IaaS Cloud is administered through a Cloud Management Platform (CMP), responsible for enabling the users to provision their virtual

infrastructures and providing a range of common services to the administrators and tenants. We perform an extended presentation of this component in Section 4.1.1.

We consider an additional component in our IaaS deployment. Since attack graphs are built with the objective to automatically determine suitable security countermeasures for risk reduction, we anticipate the need for an automated deployment of those countermeasures by introducing a Software-Defined Network (SDN) controller within our infrastructure. Indeed, one of the advantages of this approach is to be able to leverage the network programmability provided by the controller to deploy security applications responsible for ingesting the recommendations provided by attack graphs analysis algorithms and reacting accordingly. In that context, the CMP delegates the virtual network provisioning to the SDN controller. This comes at the price of consistency between the states retrieved from the Cloud Management Platform and the ones obtained from the SDN software. We present in Section 4.1.2.1 the method adopted to address those discrepancies.

1.4.1.2 Use cases

The attack graph construction represents a piece in the automation of the risk management process, whose final goal is to provide actionable intelligence to security administrators or executives in order to understand their exposure, increase the level of security of their infrastructures and ensure business continuity in the wake of a security breach.

In the context of the Cloud, both administrators and tenants can benefit from an attack graph solution. As provider of the infrastructure, an administrator can be interested in having an understanding of the global environment, to provide recommendations to customers that are more at risk than others and as such represent a security hazard for the infrastructure as a whole. As a tenant within the infrastructure, a customer can be interested in understanding its particular security exposure just like in traditional environments, with the specificity that this tenant's attack graph might include paths that are outside his virtual infrastructure, if the attackers are able to leverage virtualization vulnerabilities, allowing to access the tenant's infrastructure from co-located machines.

In a Cloud context in which several actors coexist, an attack graph represents an excellent tool to understand the impact of neighboring tenants on a particular infrastructure. Security-aware provisioning algorithms can be developed in order to prevent the co-location of virtual machines which would increase the attackers' chances of compromising the environment.

1.5 Contributions

In order to enable Cloud administrators to leverage the benefits of attack graphs for securing their infrastructure and evaluate the risks they face, we propose in this thesis a Cloud-aware graph model and generation approach.

Our first contribution is the **retrieval of the topology and connectivity of the Cloud infrastructure.** This data is an input needed for the attack graph generation process. Considering the dynamism of the Cloud, having an updated topology and connectivity is essential for an accurate attack graph. We leverage Cloud technologies such as the Cloud Management Platform (CMP) to that end, in order to retrieve an updated topology and connectivity, while avoiding limitations of physical infrastructure methods. In addition the CMP, and to anticipate later needs for dynamic network configuration, we consider an environment comprising an SDN (Software-Defined Networking) controller interfaced with the CMP, and handle potential conflicts in the data gathered. Events generated allow to track changes in the infrastructure and quickly update our attack graph representation considering only the deltas incurred by the reported modifications.

Our second contribution is the **inclusion of virtualization vulnerabilities in attack graph modeling.** We first inspected virtualization vulnerabilities descriptions present in existing vulnerability databases in order to extract the pre-requisites necessary for an attacker to be able to leverage them. In addition, we also identify the exact scope of their impacts once used successfully in an attack. This information is a crucial step to extend vulnerability models proposed in existing approaches to take into account the impact of virtualization on the attack scenarios generated. This allows to understand precisely how weaknesses in the hypervisor can be used to compromise either the underlying physical machine or the hosted virtual machines. However, not all hypervisors benefit from the same level of meticulousness in the description of vulnerabilities that affect them. As a consequence, pertinent information can be missing, making it more difficult to compare the results across several pieces of software. Nonetheless, the requirements and impact categories extracted from our analysis allowed us to extend the scope of vulnerabilities considered in attack graph construction to the virtualization layer. The study performed is described more extensively in Chapter 3;

Thirdly, we propose **an hybrid attack graph modeling.** In order to generate attack graphs that are not only comprehensive but simplified enough to handle the scale of the Cloud, we leverage advances realized in prior attack graph models. We address this issue by proposing an extension to the host-centric attack graph model, in which the vertices represent the virtual and physical host existing in the Cloud, coupled, when applicable, with the attacker and device states, as presented in Chapter 5. The chosen representation retains the ability to use algorithms based on graph theory for risk evaluation, without requesting an intermediate transformation of the attack graph. By using an event-based attack graph construction algorithm, the attack graph obtained reflects the most accurate security exposure of the infrastructure at any point in time. The data contained within event notification messages are analyzed to determine their impacts on either connectivity or vulnerability change, and as a result the delta incurred on the attack graph. By continuously listening to changes and modifying the attack graph accordingly we are

able to avoid rebuilding it from scratch every time a change occurs. This process is presented extensively in Chapter 6.

We realize **an implementation** and perform **an evaluation of all of these proposals** to determine their efficiency in a realistic Cloud context, in terms of scale and architecture.

1.6 Outline of the Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we present the state of the art relative to the collection of data needed for the attack graph construction, namely the infrastructure topology and connectivity. The state of the art on the attack graph domain is then presented, as well as the Cloud environment and the challenges encountered in this infrastructure, regarding both connectivity retrieval and attack graph generation.

In Chapter 3, we present the virtualization vulnerabilities with the objective of extending existing attack graph generation methods to include the virtualized layer existing in Cloud environments.

Chapter 4 presents the way the topology and connectivity can be retrieved from the Cloud in order to serve as input in the attack graph generation.

The model specifically designed for attack graph representation in the context of the Cloud is presented in Chapter 5.

Based on this model, and an extension of the event-based approach adopted in Chapter 4 we are able to generate up-to-date attack graphs in the context of the Cloud, as presented in Chapter 6.

Chapter 7 presents the experiments realized in order to validate our approach as well as the results obtained.

We finally conclude in Chapter 8, with a presentation of the limitations and perspectives of our work.

CHAPTER 2

STATE OF THE ART

In this chapter, we will start with a state of the art on topology and connectivity retrieval, since they are necessary inputs for an attack graph generation. We will then introduce the state of the art on attack graphs, focusing on the models and the generation algorithms. Then, we present a brief description of the Cloud, which is the environment of interest in this thesis, and outline the challenges posed by Cloud environments for topology and connectivity retrieval, as well as, attack graph generation.

2.1 State of the Art on Connectivity and Topology Discovery

An essential input used on the attack graph generation process is the topology and connectivity of the infrastructure devices. It is thus important to identify how these information can be obtained in the context of the Cloud.

We present the state of the art regarding connectivity and topology discovery techniques, organized into passive and active methods. The active approach comprises agent-based methods, with agents installed in each device, and monitor-based methods, less intrusive than agent-based approaches, due to the use of dedicated servers. The applicability of passive and active methodologies to the Cloud is analyzed to uncover limits and requirements for an efficient topology and connectivity retrieval. We start by defining the terms to better understand what they refer to in the remainder of this section.

Definition 1. *Network topology refers to the way the network is organized. It describes the way in which the constituent parts of the network are interrelated or arranged, how the computers or nodes are linked to each other. The topology can be physical, namely mapping hardware configuration,*

or logical, namely mapping the path that data must follow to travel in the network. The topology can be physically laid out a certain way, but configured logically to operate in a different manner.

Definition 2. *Network connectivity refers to the methods used by the devices to communicate with each other and the direction of the information. It identifies the protocols and ports involved in the communication between machines.*

Being the network of networks, the study of the Internet topology has drawn the most attention from the research community and generated the majority of the work in that domain, as shown in various surveys [35, 36]. Since the Internet topology can be analyzed at different granularity levels, we will focus on the interface level, a granularity in which a node represents a network interface with a designated IP address, namely a host or a router. This is a scope that can be adapted to the context of the Cloud and generates data that are relevant in the attack graph generation process.

2.1.1 Passive Discovery Methods

Passive methods are based on a non-intrusive observation of the network traffic flowing over the wire to detect devices and reconstruct equipment topology. Passive measurements can be carried out by the deployment of specialized hardware such as network taps [37] at strategic locations in the network and binding them to traffic analyzers, however with significant costs. Other methodologies involve port mirroring, incurring an additional workload on the switches involved, as each packet on the monitored ports is copied and sent to a monitoring host [38]. Flow export protocols such as sFlow or NetFlow can also be leveraged to reconstruct the topology. They provide access to information pertaining to layer 2, 3 and 4 of the OSI model. Few methods in the literature rely solely on pure passive methodologies for topology reconstruction. However, Eriksson *et al.* [39] used passive measurements to infer structural properties in the Internet. By observing the hop-count vectors between sources and passive monitors, they are able to cluster sources sharing network paths, according to similarities discovered.

2.1.2 Active Discovery Methods

2.1.2.1 Agent-based Approaches

Agent-based approaches rely on agents deployed in each device, and often use the SNMP protocol (Simple Network Management Protocol) [40], with SNMP agents installed in routers, switches or end-hosts. SNMP is a network management protocol supported by many management devices. An SNMP manager queries the SNMP agents installed in each SNMP-enabled devices. It is then able to identify the pieces of equipment according to the type of replies and information queried from the devices Management Information Bases (MIBs). Network management tools allow the automated discovery of routers, subnets and layer-3 topology.

Breitbart *et al.* [41] propose an algorithm based on standard SNMP information to construct layer-2 topology. They rely on local address forwarding tables collected in the SNMP Management Information Base (MIB) of the equipment.

Lowerkamp *et al.* [42] have extended this work by integrating incomplete database knowledge and non-cooperative (without SNMP) pieces of equipment such as hubs. Even when relying on standard protocols such as SNMP, difficulties arise due to vendors specificities. Indeed, implementation can be extended across platforms, and inconsistencies in table indexing schemes may occur. This leads to additional challenges when processing data originating from multiple sources. Besides, agent-based approaches lead to intrusiveness into infrastructure devices, due to the need for an agent within a tenant equipment.

Nassu *et al.* [43] introduce a strategy for topology discovery in dynamic and decentralized networks which relies on mobile agents and swarm intelligence. Instead of being static and remaining on the node they are originally installed on, the agents are able to randomly migrate and spread through the network to better detect changes occurring in the infrastructure. The number of agents can also be adapted according to the workload, as well as plan their itinerary to reach the highest number of nodes.

2.1.2.2 Monitor-based Approaches

Monitor-based approaches are more flexible than the previous ones: they use a dedicated set of probing hosts, responsible for performing topology acquisition by leveraging protocols and applications already available in the users' devices, such as *ping* and *traceroute* [44, 45]. Generally, monitor-based approaches do not require the use of customer resources, since they are independent from their hardware.

Different implementations of the *traceroute* tool exist, using either *ICMP* [46] or *UDP*[1] and *TCP* packets as probes. *traceroute* is the most widely used tool to map the interface topology level of the Internet [35].

These mechanisms can be combined to perform topology discovery as done by Siamwalla *et al.* [47], who associated ping, traceroute, SNMP, DNS, and ARP.

Bush *et al.* [48] used traceroute to introduce an approach called dual probing, which consists in actively probing endpoints from different parts of the address space in order to prevent biases and errors that might occur using a single location. Beverly *et al.* [49] proposed an approach based on *traceroute* and using external knowledge on the networks for probing target selection, in order to reduce the number of probes generated. Skitter[50], tool developed by the Center for Applied Internet Data Analysis (CAIDA), and the Test Traffic Measurement (TTM) [51] from RIPE Network Coordination Center (NCC) are extensive tracing systems that have been used for Internet topology discovery at the IP level. They leverage *traceroute* mechanisms between

[1]https://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/ios-software-releases-121-mainline/12778-ping-traceroute.html

24 to 200 monitors to reconstruct the topology. The Distributed Internet Measurements and Simulations (DIMES) [52] systems succeed in running an even larger number of monitors (8700 over the 5 continents in 2007) by providing a publicly available route tracing tool and acknowledging users participation to the research efforts. Donnet *et al.* [53] determined that traceroute-based tools for discovery can be inefficient, as they have to deal with the redundancy induced from repetitively probing the same interfaces. Hence, to decrease probing traffic, they opted for Doubletree [53], an algorithm allowing to reduce simultaneously intra- and inter-monitor duplicated data, by starting the probing at an intermediate distance between monitor and destination, and performing backward (destination-rooted tree) and forward (monitor-rooted tree) probing schemes. Monitors share paths they already probed to their destinations, to avoid their peers to take the same ones. In [54], they introduced the use of Bloom filters to reduce communication overhead induced by this path sharing methodology and proposed to limit the number of monitors to a given destination by applying clustering.

In Summary, we presented in the last section passive and active topology discovery methods that can be used to have an insight on the infrastructure in traditional environment. A way to take advantage of this information is by building attack graphs, as we will present in the next section.

2.2 State of the Art on Attack Graphs

Attack graphs are a convenient and compact way of considering the vulnerabilities that may exists on hosts in a network, with regards to their interactions due to the network connectivity, and understand how this affects the global security of the infrastructure. For a simple definition of an attack graph, we can present it as a model presenting all the paths available to an attacker in an information system, to compromise the existing resources. It contains vertices and edges, whose semantics vary from an author to the next. Attack graphs have a rich literature and several models have been proposed over the years. We will present the most prominent ones in the rest of this section. We can identify three main models in the literature: ***state-based***, ***host-based*** and ***exploit- or vulnerability-based*** models.

2.2.1 Attack Graph Models

2.2.1.1 State-based models

A state-based enumeration approach was primarily used to build attack graphs [1–4, 55, 56]. In such methods, a node generally represents the system state after the occurrence of an event, while the edges represent a transition from one state to the next. These events can be an attacker action exclusively [1, 2, 55, 56], or include benign actions [3, 4] occurring in the system. These events are the root cause for the transitions.

In Philips and Swiler's formalism [1, 2], the attack graph is built based on a configuration file (comprising topology information and devices configuration), an attacker profile (i.e. its skill level and capabilities, such as availability of toolkits or financial means), and a database of common attacks templates, broken into atomic steps. An illustration of such an attack graph is represented on Figure 2.1. Each vertex in the graph represents a possible attack state and is the aggregation

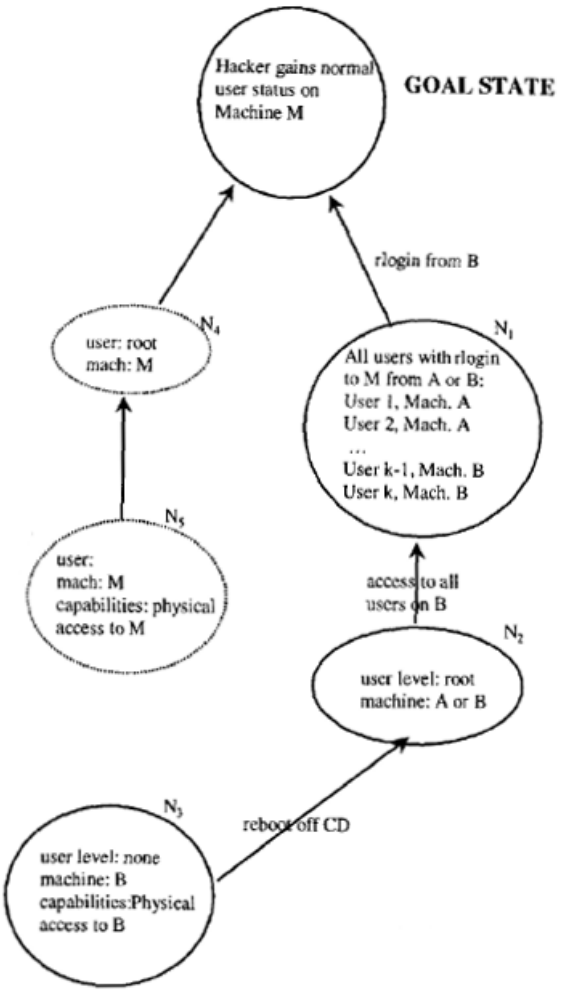


Figure 2.1: Example of Attack Graph obtained following Philips and Swiler's [1, 2] model

of different fields: *User access level* specifying the level of privilege, *Machine(s)* specifying an individual or set of machines, *Vulnerabilities* indicating changes incurred in relevant portions of the configuration file for a given node, *Capabilities* indicating changes incurred on relevant portions of the attacker profile and *State*, which breaks attacks into atomic pieces and indicates progress in the attack. Edges represent either the actions performed by an attacker or an event such as the detection of a particular type of packet on the network. These edges may be weighted. The weight is presented as a function depending on the configuration and the attacker profile, and can be used to express various metrics such as the effort required by an attacker's action

or the probability of attacker success. A difficulty presented by this model is the fact that since the *Machine(s)* field in the vertex can aggregate several devices, a path unique to a particular device cannot be represented by a vertex relative to a set of machines. This would require the construction of intermediary vertices with individual machines represented in the *Machine(s)* field. Additionally, vertices and edges represent explicitly more information than necessary, with states represented either within the vertices, or partially contained in the edges. This results in a more complex analysis.

Sheyner *et al.* [3, 4] present the first formal treatment of attack graphs. They represent the state of the network as a collection of boolean variables, each representing configuration parameters and attacker's privileges, while the state-transition relations represent attacker's actions. The security properties required for this network are then evaluated against the model, using a model checker. While this formal approach might prove to be less error-prone compared with custom-designed algorithms, the efficiency is highly dependent on the semantics chosen and formal tool used for building the graph. With this particular model, every node contains the entire network state at an attack step, leading to an exponential number of possible states. An experiment performed by Ou *et al.* [55] resulted in a graph prohibitively large even for a small network, namely 10 millions edges for 10 hosts with 5 vulnerabilities. Indeed, since all

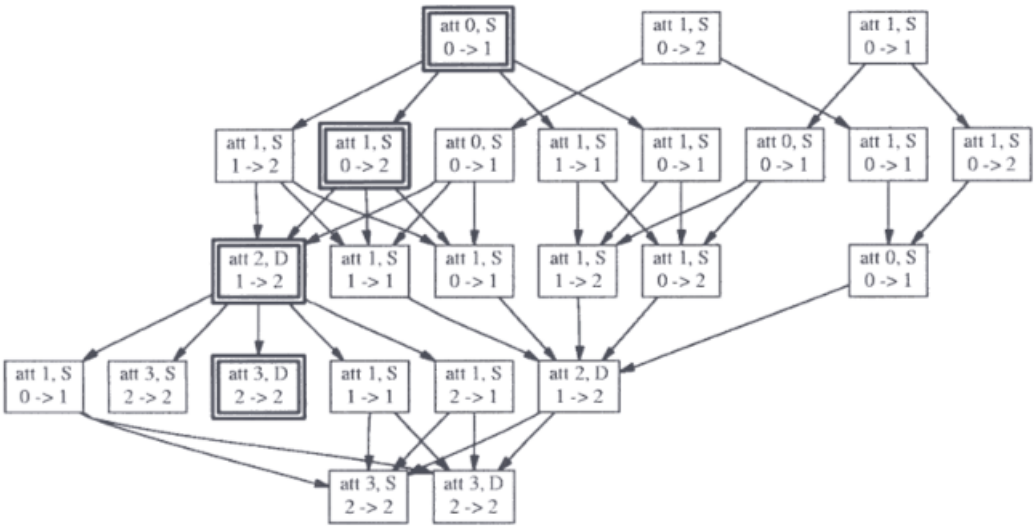


Figure 2.2: Example of Attack Graph obtained following Sheyner *et al.* [3, 4] model

the network state is represented within each node, a small parameter variation results in a completely different state. For instance, given two hosts A and B connected to the same host C, we have the following: attacking Host C from Host A results in a completely different state than attacking Host C from Host B with an identical attack action, despite the similitude of the action effects in both cases. This is essentially because the source and target host are explicitly considered in the state. An illustration of Sheyner *et al.* attack graph is represented on Figure 2.2, where each node is labeled by an attack number, a flag S/D stating whether the attack is stealthy

or detectable by an Intrusion Detection System (IDS), the source and target host of the attack. The path with double-boxed nodes highlights the sequence of attacks leading to the attacker obtaining root privileges on host 2. Ou *et al.* [55] also identified scalability issues caused by a great amount of duplicate paths that only differ in the order in which the steps are performed. In addition, making sense of the boolean variables is a challenge, and the logical link between nodes cannot be evidently deduced.

Moving away from the global approach consisting in depicting the complete network state in a node, Ou *et al.* [55, 56] opted for a partial state description. A node in these graphs represents a logical statement, containing only some aspects of the network state. It is similar to one boolean variable in the nodes of the model of Sheyner *et al.* [3]. Since their formalism is oriented toward presenting the logic behind why an attack can happen, they introduce the term of *logical attack graphs*. The logical attack graph is comprised of two types of nodes. On one hand the *derivation node*, and on the other hand the *fact node*. A fact node represents a true fact about the network, and results from configuration information reported by host and network scanners. It is labeled with the corresponding logical statement. The derivation node is computed from the configuration information by iteration over interaction rules on the input, it is dependent on one or several fact nodes. It is labeled with the interaction rule used for reaching it. Fact nodes can be dependent on derivation nodes with interaction rules yielding the fact. The edges in the graph determine the causal relationships between the different type of nodes. Implicit logical (AND/OR) relationships emerge from this model: since a fact node may have different ways to become true, the edges coming from derivation nodes to a fact node form a disjunction. On the other hand, since derivation nodes depend on multiple fact nodes for the success of the corresponding interaction rule, the edges coming from fact nodes to a derivation node form a conjunction. Compared with the approach in [3], the vertices represent uncombined and specific states, relative to a portion of the network global state. Depending on the granularity of the modeling, this fragmentation can result in lots of intermediary states. The logical links between the nodes are clearer than before, however the vertices and edges semantics add complexity to the analysis, since the interpretation of outgoing and incoming edges is dependent on the type of vertex.

Generally, state enumeration attack graphs tend to explicitly represent more information than necessary, be it the entire network state in each vertex or multiple vertices with partial ad fragmented information.

2.2.1.2 Host-based models

Instead of using a model based on states, authors can chose to focus on the hosts by adopting a *host-centric approach*.

Ammann *et al.* [5] present such an approach, organized around hosts for attack graph representation. It has been originally designed for penetration testers and system administrators, with

the postulate that their analysis is less detail-focused, and more oriented towards maximal levels of penetration obtainable on a host. As a result, to address the scalability issue generally inherent to attack graph generation methods, they present an alternative to complete graphs which contain all the attack paths relative to all the weaknesses found on hosts in the infrastructure. Instead, only the worst case scenario is considered at each host, meaning that only the attack action resulting in the maximal level of attacker privilege on a host is included in the attack graph. Each node in these graphs represents a host in the network, while an edge is the representation of an access level between two hosts. The authors start from an initial access graph, in which directed edges, associated with the highest access level on the destination host, are added to the graph according to intended trust relationships in the network. A trust relationship represents the ability for a user on a machine to access a remote service on a connected machine without providing any credentials. In a second phase, an algorithm determines the maximal level of privilege obtainable by an attacker on each host after performing available attack actions, and potentially modifies existing edges to retain only the highest level of access. An example of attack graph following this representation is presented on Figure 2.4, with Figure 2.3a showing the access graph, i.e., communications enabled, and Figure 2.3b the resulting attack graph.

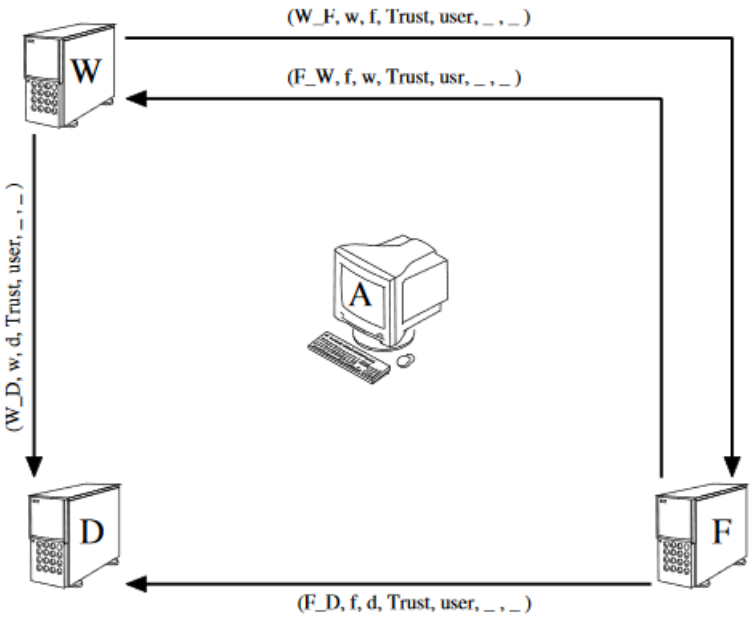
Amman *et al.* present a very operational take on attack graph modeling, with a focus on the hosts that renders it more palatable to system administrators and penetration testers. They trade the exhaustivity of complete attack graphs for the efficiency of sub-optimal attack graph model, with regards to the maximal penetration approach. However, this design choice prevents them from performing global analyses and optimization on the graph, resulting in an incremental, hence potentially costly, security improvement.

Zhong *et al.* [57] also adopt a host-centric approach. However in their case, the edges have a different semantic. In these graphs, they represent attack actions used for compromising the target host independently from any access level, while edges in [5] depict first and foremost access level tagged with either the trust relationship or attack action identifiers.

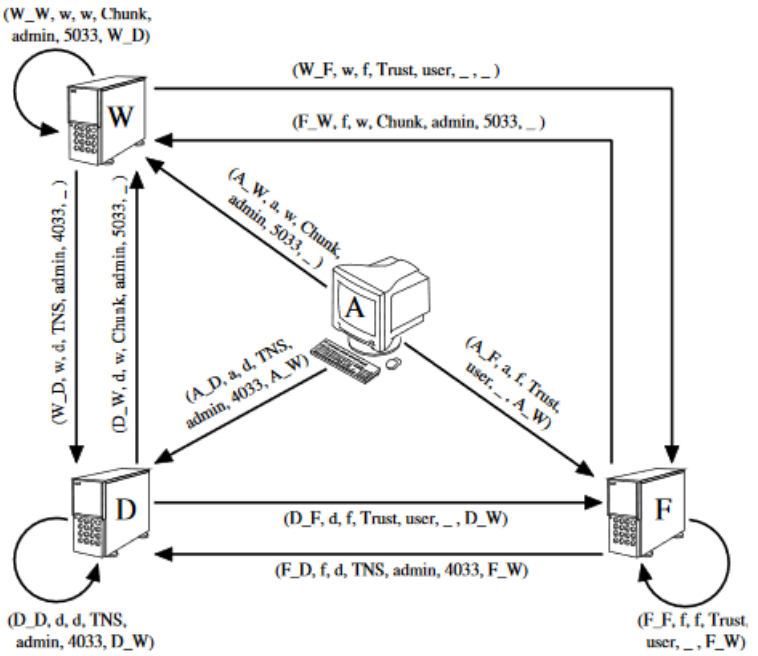
2.2.1.3 Exploit- or Vulnerability-based models

Alternative approaches to the representation of computer attacks are based on *exploit dependency graphs*. The term *exploit* refers to the program used to perpetrate an attack taking advantage of known vulnerabilities existing on the hosts. The mapping between vulnerabilities and their associated exploits is publicly available on dedicated websites. In such graphs, attributes illustrating the conditions and effects of the attacks are represented by graph nodes and edges.

Dacier *et al.* [58, 59] introduce the notion of privilege attack graphs. In this model, the nodes also depicts attacks pre- and post-conditions, with the distinction that only the access level of users or groups of users are considered as conditions. Edges in that context represents the vulnerabilities permitting the transition from one access level to the next.



(a) Access Graph

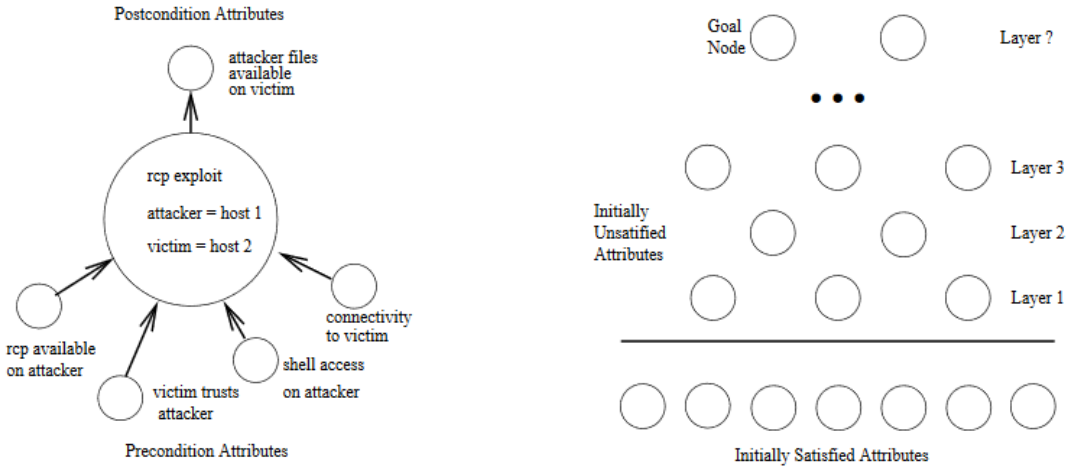


(b) Attack Graph showing maximum access

Figure 2.3: Example of Attack Graph obtained following Amman *et al.* [5] model

In [6], Ammann *et al.* model an exploit as an atomic transformation, that given a set of preconditions, establishes a set of postconditions. This is closely related to the template model of Phillips and Swiler in [1]. These preconditions and postconditions sets represents attributes

and are modeled as vertices in the graph. Edges in the graph are labeled with exploit identifiers. Edges with a specific exploit identifier are drawn from each attribute that is a precondition of this exploit, towards every attributes that is a post-condition of this exploit. The number of edges associated with an exploit is then the product of the number of preconditions and post-conditions. To alleviate the complexity of the representation, the authors opt for an *edge-less graph*. This is realized by an implicit representation of the edge as a label affected to the vertex description. This label contains the array of exploit identifiers responsible for the realization of that attribute. The attributes are organized as hierarchical layers, based on the number of exploits required before their satisfaction. An example of exploit is presented on Figure 2.4a, as well as, the hierachy of attributes on Figure 2.4b according to illustrations in [6].



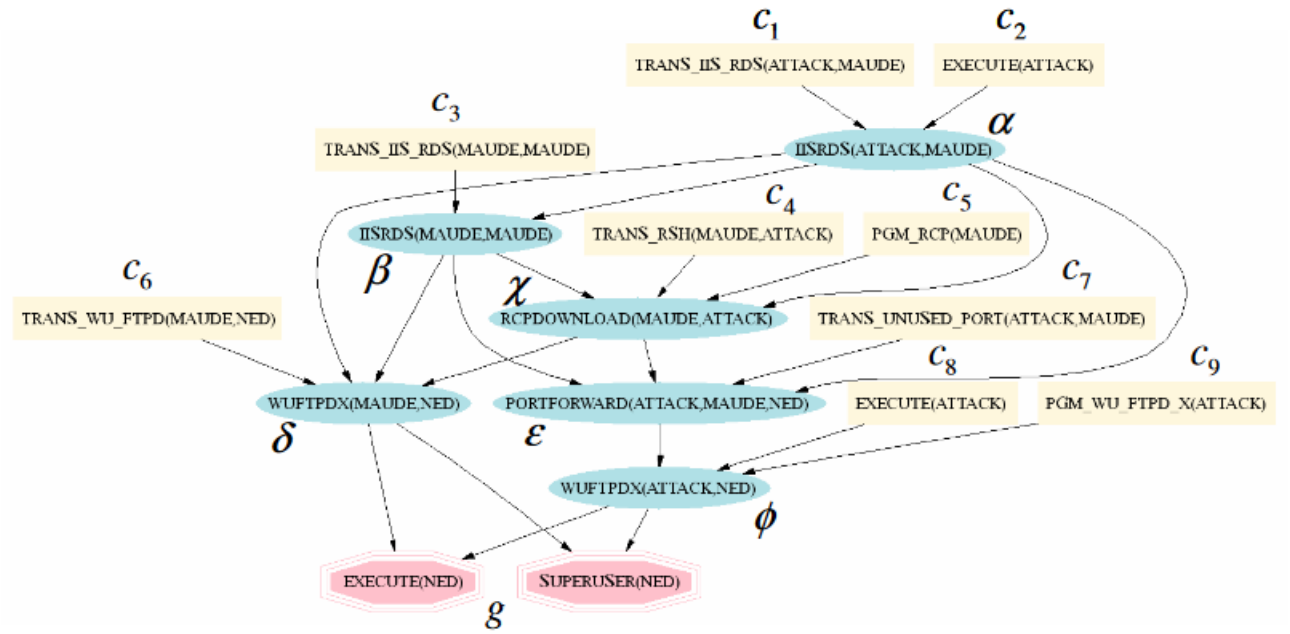
(a) Example of Exploit

(b) Attributes arrangement

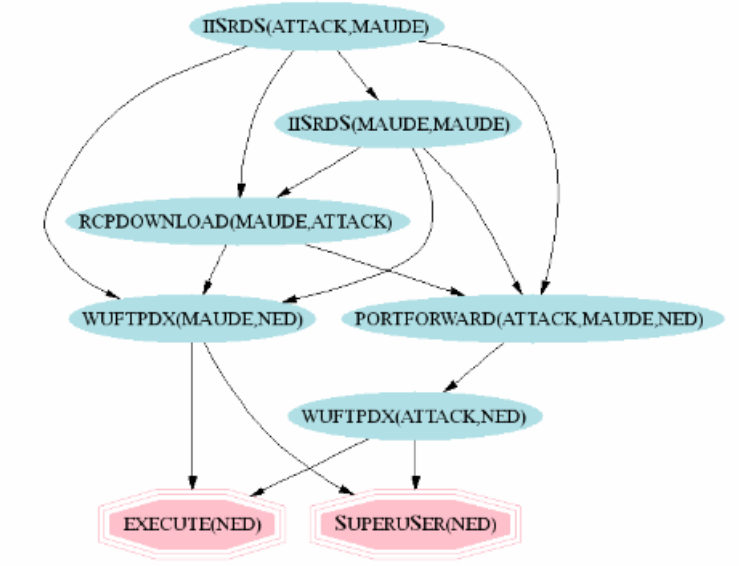
Figure 2.4: Example of Attack Graph obtained following Amman *et al.* [6] model

In their tool TVA (Topological Vulnerability Analysis), Jajodia *et al.* [7, 60, 61] adopt an identical approach to the one of Ammann *et al.* They compute the attack paths based on a directed graph of the dependencies among exploits and conditions, and choose to represent both exploits and conditions as nodes. In that context, dependencies are expressed with unlabeled edges. Directed edges from attacks to condition nodes represent the post-conditions of an attack while the ones from conditions to exploits nodes depict the pre-conditions of an attack. An illustration of such a graph is represented on Figure 2.5, with Figure 2.5a representing the attack graph with the pre- and post-conditions represented, and Figure 2.5b representing the attack graph with no conditions represented.

Ingols *et al.* [8] propose a different approach to full graphs [62]. In full graphs, nodes correspond to states and edges to vulnerability instances. A node is added to the graph if no ancestor node has the same state as the new node and was reached using the same vulnerability as the new node. These graphs illustrate explicitly every order in which an attacker can compromise the hosts. Ingols *et al.* introduce the concept of Multiple Prerequisites graphs (MP-graphs) in NetSPA,



(a) Attack Graph with Conditions represented



(b) Attack Graph with no Conditions represented

Figure 2.5: Example of Attack Graph obtained following Jajodia *et al.* [7] model

their attack graph generation tool. In this formalism, 3 types of nodes are used: *vulnerability instances nodes*, representing a particular vulnerability on a machine, *attack prerequisites* nodes

representing a credential or a groups of connected machines (reachability groups) and *states* nodes, representing the attacker's level of access on a particular host. As a consequence, nodes representing states in MP-graphs represents also vulnerability prerequisites. With the introduction of reachability groups, this modeling allows to represent full graphs in a more compact way, and speed up the graph generation process.

The contrast between full graph and MP graph is presented on Figure 2.7 based on the network example represented on Figure 2.6.

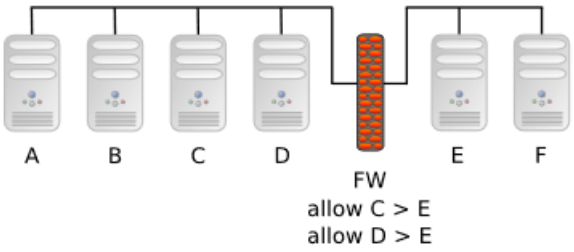
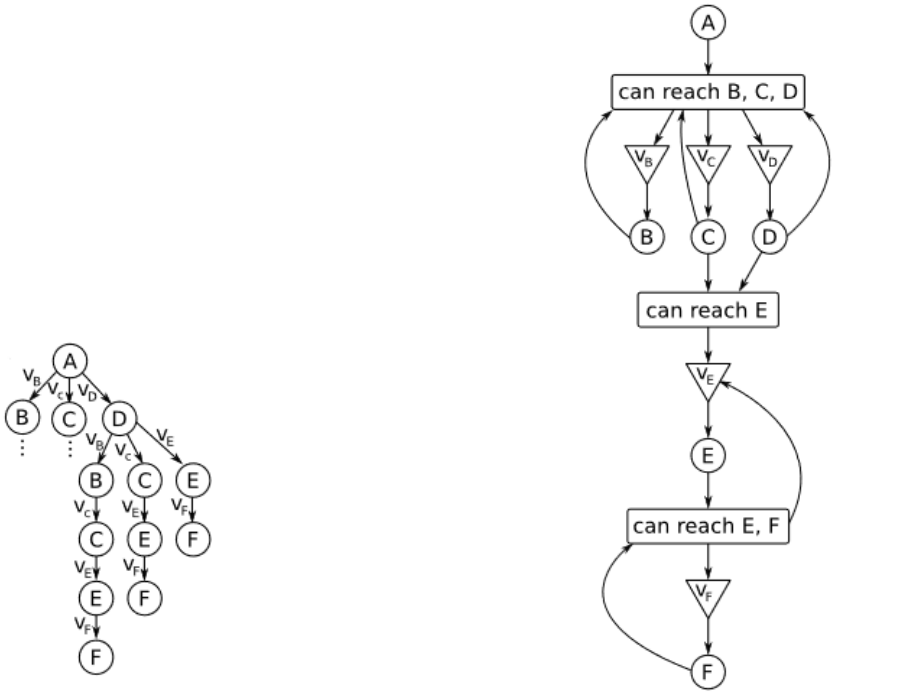


Figure 2.6: Network example



(a) Full Graph

(b) Multiple Prerequisites graph

Figure 2.7: Example of Attack Graph obtained following Ingols *et al.* [8] model

A comparison of the attack graph model previously introduced is presented in Table 2.1.

Attack Graph Models	Limitations
State-based models using model checking	A node is equivalent to a network state; entire network state encoded in each node; exponential size of the graph
State-based models using logical approach	A node is equivalent to a logical statement, i.e., a partial state; use of logical operators preventing the use of graph algorithms without prior graph transformation
Exploit-based models	A node is equivalent to an exploit action; the focus on the hosts involved in the attack graph is lost
Host-based models	A node is equivalent to an infrastructure device; less details are presented, it requires an additional subgraph to get a complete scenario

Table 2.1: Comparison of attack graph models

2.2.2 Attack Graph Generation Algorithms

Several approaches rely on the use of in-house customized algorithms in order to generate attack graphs. The algorithms behind the NetSPA and TVA tools represent some examples. But many other approaches have been proposed in the literature in the past decades. In the next section, we present ***logic-based algorithms based on reasoning engines*** and ***custom algorithms***.

2.2.2.1 Logic-based algorithms based on Reasoning Engines

Algorithms based on Model Checking Engines

Ritchey and Ammann [63] were the first to propose the use of model checkers for attack graph generation.

Model checking consists in automatically verifying that a system satisfies a desirable property or specification and relies on an exhaustive enumeration of all the states reachable by the system. Using a model of the system under analysis and a formulation of the property to verify, a model checker is able to verify whether the property is true or false. In the latter case, it outputs one [63] or several [3, 27] counter-examples, which represent the sequence of states and transitions invalidating the property. This approach requires a complete model of the system as a finite state machine and a specific formalisation of the properties. On a different note, [64] used model checking for configuration vulnerability analysis in single host systems, using an infinite-state model.

By using standard model checking approaches, the authors can directly exploit technological advances in the field without having to design tailored algorithms. As such, the performance of the generation is contingent to the model checkers efficiency, which are designed to handle large state space [65]. However, a known issue of model checking is the combinatorial blow-up of the state-space, i.e., the state explosion problem, making it only suitable for small problems. The number of state variables required to model the system information increases tremendously with the number of hosts and vulnerabilities. As Sheyner *et al.* [3] experienced, an attack graph

relative to a small network of 5 hosts with 8 atomic attacks resulted in over 6000 reachable states. Even when all the states cannot be reached in the search, the potential state explosion makes it impractical except for small networks with a limited number of vulnerabilities.

Algorithms based on other Logic Programming Reasoning Engines

MulVAL [55, 56] is an open source framework which became the building block of several approaches using attack graphs [66]. It contains a reasoning engine depending on XSB [67], a logic-based inference engine. Using a high-level declarative programming language such as Prolog (Programing in Logic) [68] allows to express facts and rules, which are relationships among objects and their properties. This approach enables the use of first-order logic and first-order predicate calculus, permitting a formal support to attack graph generation in MulVAL. It is then possible to state a given problem and query the system for the result, without specifying in details how to solve it. In MulVAL specifically, an initial attack simulation phase results in the derivation of all multi-host, multi-stage attack paths, which serves as inputs to a policy checking phase. The problem stated in that environment, as in model checking approaches, is the detection of a policy violation, namely an unauthorized access to resources in the system.

The rationale behind the use of logic based approaches compared to other ad-hoc methods is that given the complexity of the problem, it is less susceptible to generate errors of reasoning.

2.2.2.2 Custom algorithms

Instead of relying on available reasoning engines, authors can also choose to develop their own algorithms for attack graph generation, hence allowing more flexibility in the design and model of their system. The core building algorithm for lots of approaches is similar to some extent to a breadth-first search [57, 69].

Ammann *et al.* [6] rely on a breadth first approach to determine the hierarchy of the layer in their graph and which nodes can effectively be reachable at each layers, depending on the conditions satisfied in the preceding layer. In a subsequent work [5], they use the same idea to determine obtainable level of access between all the hosts in the network using each host's known vulnerabilities, given that sufficient connectivity, an exploitable vulnerability exists on the destination host and the source host has all the prerequisites for the attack.

In TVA [7], a multi-step process is performed for the computation of the dependency graph. First, a determination of the attacks successfully executable by an attacker is made. Next, a forward traversal starting from the conditions of the initial attack is realized to obtain an initial dependency graph on which a backward traversal is performed starting from the attack goal. On this final dependency graph, a breadth-first traversal algorithm is performed to build a representation of all possible attack paths.

Different approaches [7, 8, 55, 56, 60, 61] adopt the monotonicity property assumption. Ammann *et al.* [6] introduce the notion of monotonicity for the construction of attack graphs and

deem their assumption reasonable in several network analysis situations. Monotonicity means that once the postconditions associated with a given attack are realized, they are never disabled by the successful execution of another attack. This assumption reduces the complexity of the analysis problem from exponential to polynomial, thereby bringing very large networks within reach of analysis.

2.3 Challenges of Attack Graph Generation

2.3.1 Core Components in Attack Graphs Generation

Across all the attack graph generation methods presented, we can identify a recurring set of data serving as input to the generation algorithms, with minor differences introduced to accommodate the particular concepts of the methods presented. In general, attack graph generation approaches are based on both **vulnerability information** and **network connectivity** as input data. The inventory of vulnerabilities existing on all the devices on which the analysis will be performed is generally obtained using a dedicated vulnerability scanner. These vulnerabilities are analyzed with regards to the network connectivity of the devices on which they exists, in order to determine the attack chains possible in the infrastructure. In addition to the vulnerability inventory and network connectivity, attacker profiles may also be taken into account in order to determine which vulnerabilities they will be able to leverage, the type of assets they will be more likely to target or the impact they might generate in the environment. Instead of analyzing concrete vulnerability instances on the machines, generic attack templates may also be used for graph generation.

Traditional approaches generally consider these pieces of information as input data, without considering the implementation of their collection. Indeed, this assumption can be made in these environments because the infrastructure is mostly static and is rarely subject to modifications.

However, in order to design an efficient attack graph generation approach for Cloud environments, it is necessary to consider how these core components are influenced by the novel context considered. For the specific case of network connectivity, we can expect a fast rate of change, in adequacy with the broad flexibility available to users. Indeed, virtual machine creation, migration and deletion have a direct impact on connectivity, which needs to be regularly updated as a result. The notion of co-location can also be introduced, which does not exists in traditional environments. Virtual machines are co-located when they are hosted on the same hypervisor. This can result in a potential communication channel if the isolation property of the hypervisor is broken due to the existence of vulnerabilities in the virtualization software. The existence of this communication channel is affected by virtual machine migration, from a safe to a vulnerable hypervisor, and vice versa.

In Summary, A variety of attack graph models can be found in the literature for an application to traditional environments. The performance of the generation depends highly on the model and assumption chosen. Indeed, initial state-based approaches considering the entire state at each node have an exponential complexity, and have been replaced by methods with partial state nodes, exploit-centric or host-centric.

It emerges from our study that, just like aggregating all the information in a single node, fragmenting the information too much is detrimental to the analysis. Indeed, even considering a fully automated approach providing relevant attack paths and applying countermeasures on the fly, an attack graph user may need to visualize the attack paths generated to verify the mitigation proposals, or understand the anatomy of the attacks for forensics reasons. It is hence necessary to find the right balance in the granularity of the information represented, this is even more crucial considering the Cloud scale. We identify Amman *et al.* [5] host-centric proposal, specifically tailored for human operators, as a nice approach to make attack graphs more user-friendly.

We present in the following sections the specific environment considered in this thesis, based on Software-Defined Networking and the Cloud.

2.4 Software Defined-Networking: A Way to Perform Networking in the Cloud

The advent of Software Defined-Networking (SDN) is the promise of more flexibility in network infrastructures. SDN is a rising network paradigm in which the control and data planes of the network are decoupled from one another allowing the abstraction of the underlying infrastructure to applications and network services, as well as the direct programmability of the network control. In such architecture, the brain controlling all the pieces of equipment of the network is centralized on a single device: the controller. We can separate the SDN architecture into three layers:

- The control layer (control plane): it refers to the device that controls the behavior of the network: network paths, forwarding behavior, etc. It is instantiated as a single, high-level software controller,
- The infrastructure layer (data plane): it refers on the other hand to the devices that are responsible for forwarding the packets: routers, switches, firewalls, and other middleboxes,
- The application layer: it is added on top of the control plane and allows developers to extend the controller functionalities.

Figure 2.8 is an illustration of this novel architecture. The forwarding instructions are based on the notion of flows, a flow being a set of packets sharing the same characteristics: ingress/egress ports, Ethernet addresses, IP addresses, VLAN tags, protocols, etc. According to

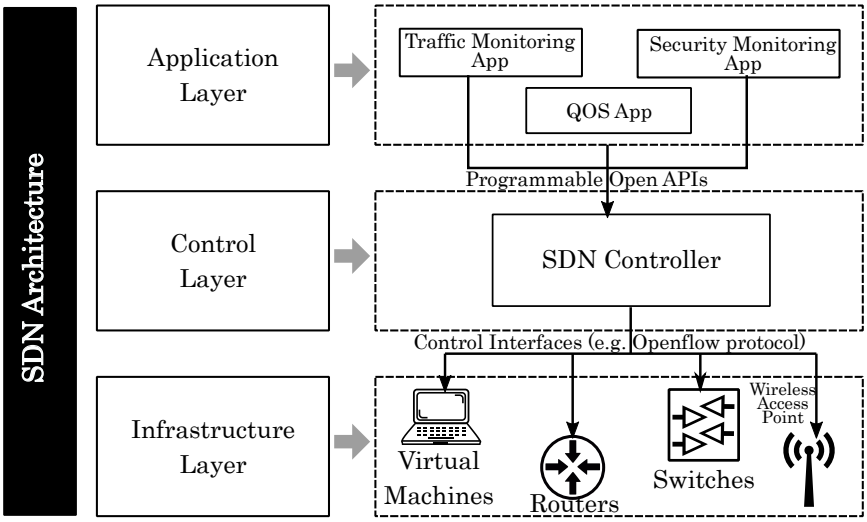


Figure 2.8: SDN Architecture [9]

those characteristics, the controller defines each flow as well as the way it should be handled by the switch. The switch contains flow tables, with a list of flow entries. Each flow entry contains a match field characterizing the flow, a counter and a set of instructions to apply to the flow. When a packet matching no entry in the flow tables arrives to the switch, a new entry must be created: the packet will be sent to the controller, which will then define a new flow and send the entry to be added to the flow tables of the switch.

It follows from what we described that SDN controllers can be connected to virtual switches existing in internal network virtualization, in order to bring the benefits of SDN to the Cloud: among other things, adaptability by dynamically changing the networking rules according to the traffic, ease of configuration and management using SDN applications, enhanced flexibility and extensibility by facilitating the integration of new applications, centralized management through the controller which concentrates the whole knowledge about the network. The dynamic nature of the SDN paradigm represents an opportunity for security improvements by designing security-specific applications on top of the control plane, such as intrusion detection or DOS prevention systems based on flow analysis.

2.5 Cloud Computing and Virtualization

2.5.1 Cloud computing

Due to the tremendous growth of Cloud Computing usage over the years, it has now become a pervasive IT delivery service. The National Institute of Standards and Technology (NIST) defines Cloud Computing as follows [70]:

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

2.5.1.1 Cloud computing deployment models

Given the openness of the infrastructure to the general public and the number of organizations whose infrastructures and services are hosted in a given Cloud, the NIST determined four deployment models for the Cloud.

Private Cloud

In that deployment model, a single organization with multiple consumers benefits from the Cloud infrastructure. That infrastructure may be owned, managed, and operated by the organization, a third-party, or a combination of both, and it may exist on or off premises.

Community Cloud

In that deployment model, a specific community of consumers, from organizations with shared concerns, benefits from the Cloud infrastructure. That infrastructure may be owned, managed, and operated by one of the community organizations, a third-party, or a combination of them, and it may exist on or off premises.

Public Cloud

In that deployment model, the Cloud infrastructure is opened to the general public. That infrastructure may be owned, managed, and operated by a business, academic, or government organization, or a combination of them. It exists on the Cloud provider premises.

Hybrid Cloud

In that deployment model, the Cloud infrastructure is comprised of two or more distinct Cloud infrastructures belonging to the previous deployment models. They each remain a unique entity, but are bound together by standardized or proprietary technology that enables data and application portability.

2.5.1.2 Cloud computing service models

Consumers of a Cloud Computing platform are provided with various capabilities allowing them to implement their services or architectures. The NIST introduces three abstraction layers to define the service models. The responsibility to manage, monitor and secure the infrastructure hardware and software components is split between the Cloud platform administrator and the consumer in different degrees, based on the choice of service model.

Infrastructure as a Service (IaaS)

In that service model, the consumer can provision essential resources such as networks, storage and computing, in which he can deploy arbitrary operating systems (OSes) and applications. While he controls the operating systems, storage and deployed applications, and possibly

some networking components, the underlying hardware infrastructure is managed by the Cloud administrator.

Regarding the specific aspect of security, he is responsible for data classification and accountability, client endpoint protection, identity and access management, application controls and shares the responsibility of network controls with the Cloud provider.

Platform as a Service (PaaS)

In that service model, the consumer has no control over the underlying hardware infrastructure, as well as networks, servers, operating systems or storage. He can deploy paid or consumer-created applications designed using programming languages, libraries, services and tools supported by the provider, and as such, can manage the deployed applications and possibly configuration settings for the application-hosting environment.

Identity and access management as well as application controls are shared between the Cloud provider and the customers, while data and client and endpoint protection are the responsibility of the customer.

Software as a Service (SaaS)

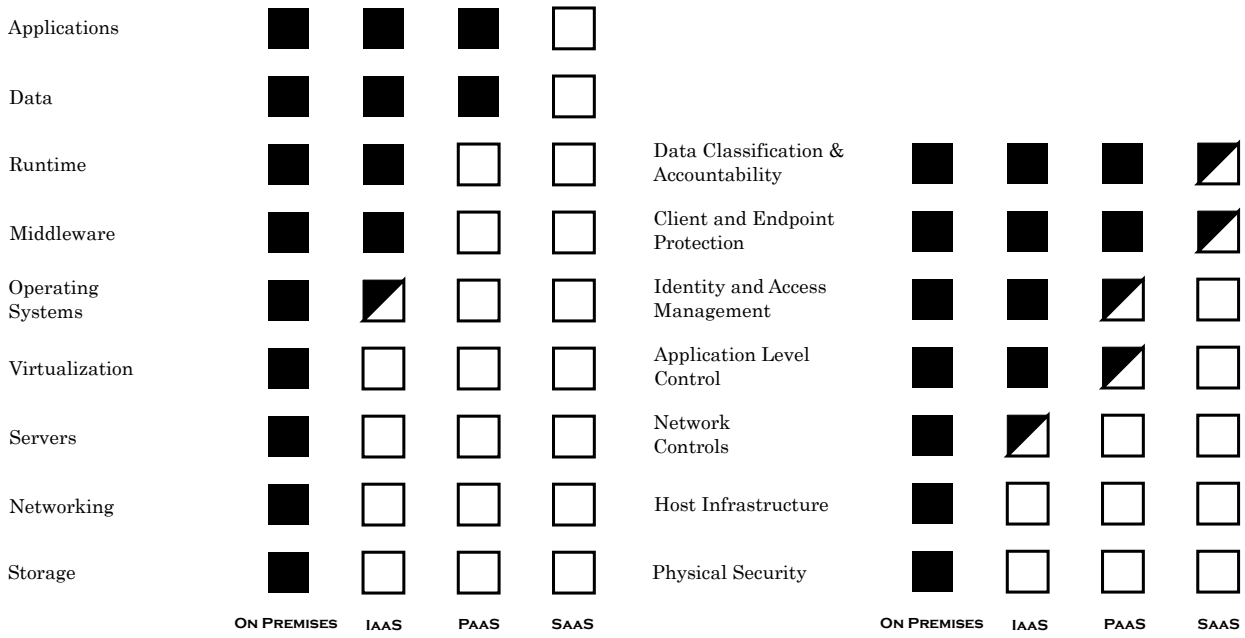
In that service model, the consumer can only use the applications made available by the Cloud provider with no control over the underlying hardware infrastructure, the networks, servers, operating systems, storage or individual application capabilities, except possibly limited user-specific application configuration settings. These applications are accessible through client devices using a thin client interface or a program interface.

In that configuration, the customer shares the responsibility of data and client and endpoint protection with the Cloud provider.

The scope of the responsibilities assigned to the Cloud provider and its customers are summarized on Figure 2.9. The black-filled boxes represent elements managed by the customer, while the white-filled ones are managed by the Cloud provider. Black and white filled boxes represent a shared responsibility between the Cloud provider and the customer.

2.5.2 Virtualization as a Cloud Computing Enabler

Virtualization is a technology referring to the abstraction of physical IT resources. It allows to conceal the specificities of those resources from the users. It enables either a single physical resource to appear as multiple virtual resources, or multiple physical resources to appear as a single virtual resource. This process is permitted by a layer acting as an intermediary between the resources to abstract and the users. This resource sharing, introduced by the use of logical entities on top of the physical ones, permits not only the optimization of resource utilization by affecting unused resources to requesting users, but also flexibility and rapid provisioning due to the software nature of the resource orchestration.



(a) Division of responsibilities regarding the components of the Cloud infrastructure

(b) Division of responsibilities regarding the security services proposed in the Cloud [71]

Figure 2.9: Division of responsibilities between the Cloud provider and the end-user according to the Cloud service model

Virtualization emerged in the 1960s out of the necessity for the scientific community to dispose of time sharing systems allowing user interactions [72]. At that time, single-user batch processing systems were the most prominent type of computing platforms, greatly impacting the development and debugging time of software programs. The goal was then, while allowing a steady stream of batch work to be processed in the background, to enable as small amount of computer time to be redirected to the online users for a more interactive use, e.g., commands to prepare, execute and terminate their programs. Additionally, every new computer system produced by International Business Machines (IBM) came at that time with new designs and technical specifications. This compelled the users to continuously spend a considerable amount of time to learn the system once again, port software programs and resolve incompatibilities with the existing hardware. Tractions in the research community lead IBM to develop the CP-40 system used only in lab environments, which evolved into the CP-67 system, first commercial mainframe to support virtualization [73]. Their designs benefited strongly from technical advances developed in building time sharing systems. The systems developed by IBM consisted in a Control Program (CP) deployed on the Mainframe, and responsible for creating virtual machines running a Console Monitor System (CMS) with which the users could interact. The CP tackled the issue of multiple use by providing separate computing environments at the machine instruction level for each user, while the CMS provided single user service freed from the issue of sharing, allocation and protection. This approach allowed to share the overall resources of the mainframe with the users,

in addition to ensuring the users' workload isolation.

Over the years, this pioneer approach evolved into virtualization as we know it, and that we present in the rest of this section. First of all, it is essential to distinguish between emulation and virtualization.

The purpose of *emulation* is to imitate a hardware in software, on a completely different platform. For instance, emulation can be used to enable an operating system, or any kind of software, to run on any physical machine, even if it was not originally designed for it. Indeed, hardware components can be completely replaced by a software construct with the same behavior allowing the OS to run smoothly on the platform, as if it was running on the real hardware. For instance, emulation can provide an environment to run a video game designed for a specific console, within a computer. In that context, the game is executed through software as the hardware architectures are different between the console and the computer. It usually incurs severe performance costs due either to the limitations of the destination platform, or the difficulty in reproducing the intricacies of the original hardware or software.

Emulation can however be associated with virtualization. In that context, emulation is constrained to the simulation of a reduced set of components, generally computer devices, for a better balance between functionality and performance cost. The Quick EMUlator (QEMU) is an example of emulator commonly used. It provides emulation of a number of different devices such as graphics cards, sound cards, network devices, storage devices and controllers, serial/parallel/USB devices and memory devices.

Virtualization consists in a partial simulation of a computer hardware enabling a guest to run, with most operations occurring on the real hardware. With that approach, the real system and the guest system have an identical architecture.

Different types of resources may be considered in a virtualization scenario, however, the most common ones in the context of the Cloud are ***compute, networking and storage resources***.

2.5.2.1 Compute Virtualization

Compute virtualization consists in running several virtual machines (VMs) on top of a single physical server, also called host, by enabling the partitioning and sharing of its available resources: CPU, memory and devices. Such functionality is provided by an hypervisor, also called Virtual Machine Monitor (VMM). It is a highly privileged piece of software running on bare metal or alongside an Operating System (OS), and able to allocate hardware resources.

Two different categories of hypervisors exist, depending on their closeness with the hardware.

On one hand, we have the ***Type I*** or ***native hypervisor***. It is also qualified of bare-metal hypervisor since the VMM runs directly on the host hardware without an underlying operating system. In that case, due to the absence of an OS, the VMM is responsible for controlling the hardware, scheduling, monitoring and allocating system resources to the virtual machines. These virtual machines run above the hypervisor, on a separate layer.

On the other hand, we have the ***Type II*** or ***hosted hypervisor***, which runs as a regular application within the traditional operating system of the host. In that scenario, the virtual machines run as a third software layer above the hardware. The I/O requests sent by the VMs are trapped by the host OS to be interpreted, incurring an overhead for the guest OS.

Due to performance concerns, Type I hypervisors are the ones most deployed on physical servers in the Cloud infrastructure, since they are the closest to the hardware. Hence, in the rest of this thesis, any reference to a virtual machine monitor or hypervisor will concern bare-metal hypervisors. The main computer components involved in a virtualization scenario are the processor (CPU), the memory, and the devices.

We present next methods used to create virtual machines, namely full virtualization using binary translation, hardware assisted virtualization and OS-assisted virtualization.

Full or native virtualization using Binary Translation

Full virtualization allows unmodified operating systems to be hosted, as they function totally unaware of running in a virtual environment. It provides the VM with virtual hardware equivalent to the physical computer, namely the CPU, memory and I/O devices. In order to address the issue of non-virtualizable x86 instructions, binary translation was introduced as presented on Figure 2.10. With binary translation, all non-virtualizable or "unsafe" instructions generated by the virtual machine are translated by the hypervisor into new sequences of instructions, i.e. safe equivalents, that have the exact same effect on the virtual hardware. It does so on the fly, and caches the results for future use. Binary translation is generally combined with direct execution methods. It means that user level instructions are directly executed on the processor at native speed, for high performance, leaving the hypervisor to translated only the problematic instructions.

Hardware Assisted Virtualization: Full Virtualization using Hardware Support

Hardware vendors developed new features to simplify virtualization techniques by targeting privileged instructions specifically. They provide a way to privilege the sensitive instructions previously mentioned. With the added extensions in the processor, a novel CPU execution mode allows the hypervisor to run in a new root mode, under Ring 0, as represented on Figure 2.10. Privileged and sensitive calls are set to automatically trap to the hypervisor, removing the need for binary translation.

OS-Assisted Virtualization or Paravirtualization

The paravirtualization mechanism was introduced by Xen in 2003 [74]. In that model, the virtual machine is *aware* of running on an hypervisor, it is thus able to communicate with it for improved performance and efficiency. To that end, the host kernel has to be modified to replace non-virtualizable instructions with *hypercalls* interacting directly with the virtualization layer. Hypercalls can also be provided for memory management, interrupt handling and time keeping, which are additional critical kernel operations. We represent the functioning of this virtualization

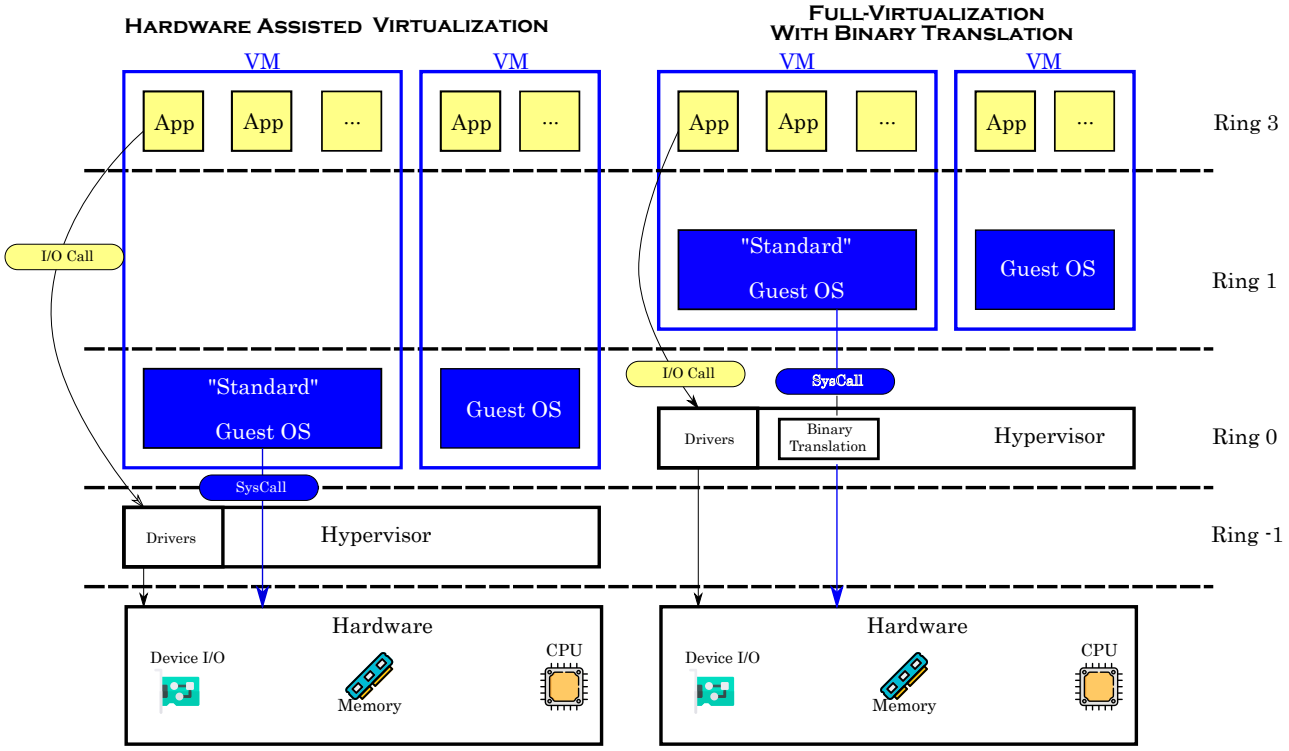


Figure 2.10: Hardware-Assisted Virtualization and Full-Virtualization with Binary Translation

method on Figure 2.11.

2.5.2.2 Networking Virtualization

Network virtualization consists in abstracting the underlying hardware and software networking resources (i.e, switches, routers, etc.) in order to provide decoupled logical or virtual network entities. These entities are combined in such a way that individual users have a dedicated view of the network. In the Cloud context, we will focus on *virtual sharing networks*, which allow the splitting of physical resources among multiple network instances, while providing clear delineation between them [75]. Network virtualization can be categorized into ***internal*** and ***external network virtualization***.

Internal Network Virtualization

Also known as *network in a box*, internal network virtualization occurs within a single physical server to create synthetic networks between virtual machines. In that context, pieces of software on the physical server (generally the hypervisor) emulate network connectivity inside the host and allows the guest virtual machines to communicate between each other, as presented on Figure 2.12.

The network interfaces within the virtual machines are called virtual network interface cards or virtual NICs (vNICs) and are all linked to a virtual switch, itself connected to the physical network card of the underlying host. As a result, the vNICS can communicate with each other

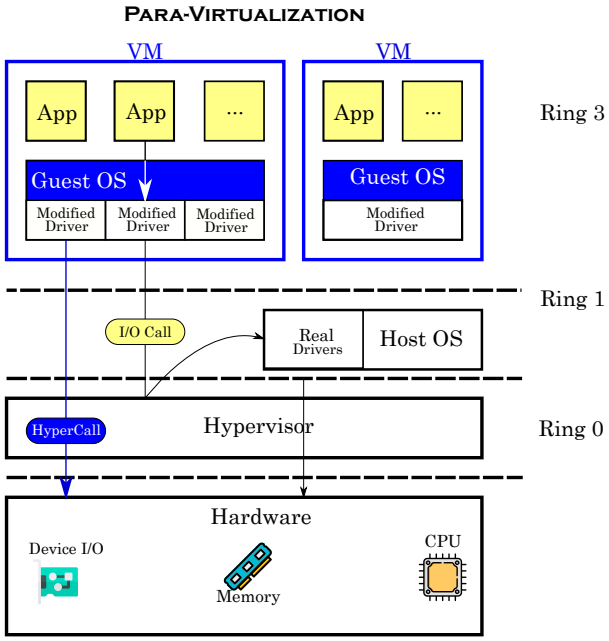


Figure 2.11: OS-Assisted or Para- Virtualization

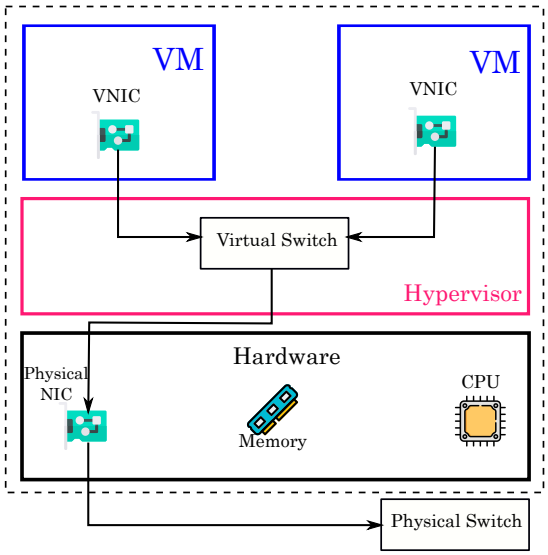


Figure 2.12: Internal network virtualization

as though they were on the same local network, effectively constituting a single-host virtual network. The internal virtual networks can be configured to be part of larger external virtual networks.

External Network Virtualization

This terminology applies to any type of network virtualization occurring outside of a virtual server, and directly involving the physical devices (i.e, switches, routers, etc.). For instance, the

use of Virtual Local Area Network (VLAN) technology allows to partition a single switched network into logical groups by introducing a 12-bit VLAN identifier or tag in the Ethernet frame. Each VLAN provides data link access to all hosts connected to switch ports configured with the same VLAN ID. The restriction to only 4096 distinct VLANs per switching domain, led to the development of additional protocols such as VXLAN (Virtual Extensible LAN) [76], NVGRE (Network Virtualization using Generic Routing Encapsulation) [77] and GENEVE (Generic Network Virtualization Encapsulation) [78], which support larger ranges of tags to identify virtual networks, and are able to accommodate the requirements of large hosting providers.

2.5.2.3 Storage Virtualization

The Storage Network Industry Association (SNIA) describes storage virtualization as follows [79]:

> " 1. The act of abstracting, hiding or isolating the internal functions of a storage (sub)system or service from applications, host, computers or general network resources, for the purpose of enabling application and network-independent management of storage or data.
> 2. The application of virtualization to storage services or devices for the purpose of aggregating functions or devices, hiding complexity, or adding new capabilities to lower storage resources. "

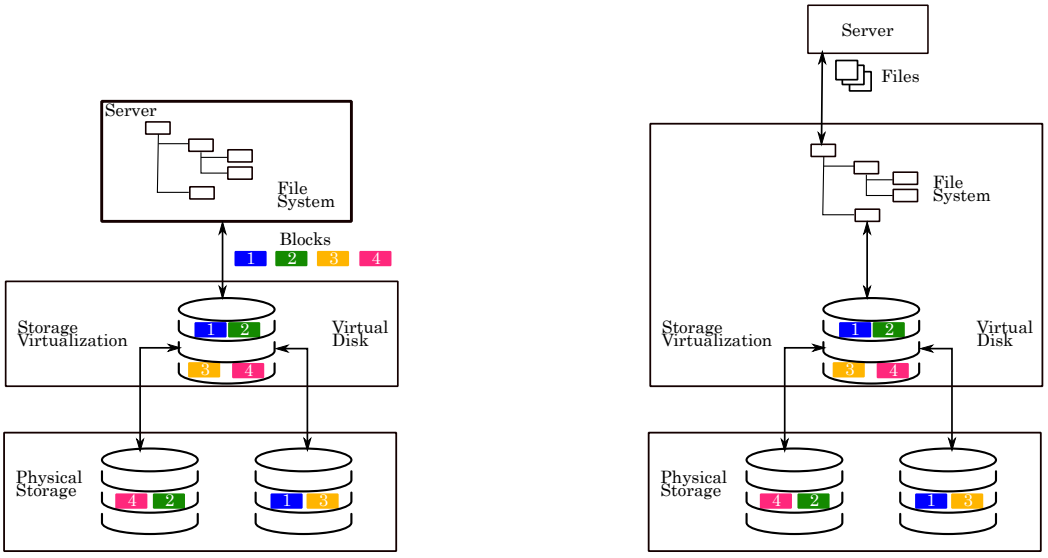
Different types of storage virtualization are described in the SNIA taxonomy. However, in the specific context of Cloud computing, we can distinguish between ***file system*** and ***block virtualization***.

File System Virtualization

With file system virtualization, location transparency can be achieved, since dedicated file servers manage shared network access to files in the file system. It removes dependencies between the data accessed at the file level and the location where the files are physically stored. Several hosts may access the files, independently of their operating systems, with the same interfaces they would use to access files locally hosted. That storage method is implemented at the Network Attached Storage (NAS).

Block Virtualization

Block virtualization consists in aggregating several physical disks to present a unique logical device. While hiding the low-level details, it aims to meet the user requirements in terms of capacity, performance and reliability by providing the suitable logical volumes. That storage virtualization method is implemented in a storage area network (SAN) and provides a translation layer between the hosts and storage arrays. Servers are redirected on virtualized Logical Unit Numbers (LUNs) on the virtualized storage devices, instead of the ones on the individual storage array.



(a) Block-level storage virtualization

(b) File-level storage virtualization

Figure 2.13: Storage virtualization methods considered in Cloud environments

In Summary, we presented in Section 2.5 the different types of virtualized components constituting the foundation of Cloud environments. Indeed, a Cloud environment relies on compute, network and storage virtualization to provide services to its customers. From this background, we observe that Cloud environments are highly complex since they are at the crossroads of several technologies and incorporate many layers of abstraction. With the experience gained from traditional environments, we can reasonably assume that the intricacies of the Cloud environment add complexity to resources monitoring management configuration, and hence are detrimental to security.

The approach we propose to regain control and insight into the virtualized infrastructure is the use of attack graphs, which have been used before in traditional environments. In the next sections, we will highlight the potential limitations in the context of the Cloud, hence uncovering challenges that need to be addressed.

2.5.3 Limitations of Passive and Active Connectivity and Topology Discovery Methods in the Cloud

Methods generally used to probe the network (i.e., traceroute, ping, ICMP) are not initially designed to gather topology information but rather to provide diagnosis capabilities. Network operators are responsible for implementing the default behavior of their devices upon receiving such probes. This behavior can greatly vary from a network to the next, thus affecting the accuracy of the information obtained. This is the case for example when some types of packets required for topology construction are filtered by firewalls. The ability to detect a particular network segment depends on the position of the listening device or the probing agents as presented in [48], which

can lead to a knowledge gap in the reconstructed topology. This approach is hardly adaptable to the Cloud, in which communication between virtual machines located on the same hypervisor occurs, without ever reaching the physical network.

On the other hand, active discovery methods tend to impose a heavy load on the network, incurring non-billable bandwidth consumption, and tend to produce traffic potentially flagged as malicious which brought down to the Cloud scale is not desirable. Indeed, this probing traffic can be identified as Denial of Service (DoS) attempts by Intrusion Detection Sytems [80] and generate unnecessary alerts in the network. With more monitors in the network comes more visibility into the infrastructure, but it also creates data redundancy, since the same interfaces are probed several times. According to Donnet *et al.* [81], the amount of redundant data generated can reach 86% on a given monitor.

Agent-based methods are not suitable given the need to install an agent on each device, especially when this virtual machine is not directly under the control of the Cloud provider.

Traditional methods can only detect whether a machine is active or not, however, in the Cloud, virtual machines can be in several states: paused, shutdown, in migration, hence impacting the topology representation. Additionally, with the multi-tenant nature of the Cloud, they are not able to attribute each machine to their owner, which is a crucial information required in a security context.

2.5.4 Discussion on the Availability of Cloud Connectivity Reconstruction Tools in the Cloud

To the best of our knowledge, researches dealing with connectivity in Cloud environments are not necessarily oriented towards a clear representation of how machines communicate with each other, on which protocols and ports.

Fabian *et al.* [82] focus on the network reachability of Cloud services. They analyze Internet Autonomous Systems connection to identify the way an outage can impact the services offered by Cloud providers. Graph models are constructed from the collected data to analyze the connectivity of the services based on important graph-based measures. It is an approach designed to help customers in determining highly available Cloud service providers. Paredes *et al.* [83] present a patent describing a system to configure connectivity in a virtualized environment, while Velasco *et al.* [84] propose an architecture based on Software-Defined Networking for the interconnection of datacenters. Chen *et al.* [85] are interested in the security implications of intra- and inter-Cloud communications and propose the use of a virtual extranet for businesses interested in collaborating with each other. The collaboration requirements and resources agreement are specified using a specific language. The methods presented do not allows users to gain insights into the internal connectivity of the Cloud service provider finally chosen. Mundt and Vetterick [86] demonstrate how Cloud resources can be used for scientific calculation. They present the analysis of a topology dataset coming from a mesh network, using Cloud resources. This work

leverages the virtualized platform resources but for the analysis of an external and independent network.

We conclude from these methods that the Cloud connectivity configuration and setup is more studied than the obtention of the Cloud connectivity configured. However, the availability of resource management technologies in the Cloud represents an opportunity to address limitations from traditional methodologies. These technologies allow to leverage a centralized store containing all the needed information for the topology and connectivity construction [87–89]. In addition, since they are designed to react to changes in the Cloud infrastructure, they can be used to identify any modification occurring and update the connectivity.

2.5.5 Attack Graphs in the Cloud

Several surveys analyze the current state of Cloud security and solutions proposed to secure this environment. If some researchers focus on specific issues such as data security [90] or achieving confidentiality from the Cloud provider [91], others [92–94] adopted a broader view to the challenges and mitigations proposed in the Cloud. However, these surveys do not reference the use of attack graphs as a way to address security in the Cloud. In their survey of information security incident handling in the Cloud, Rahman *et al.* [95] introduce solutions for incident response. One of these solutions presents a cost-sensitive assessment approach to ensure a trade-off between damage and response costs by leveraging a dependency graph that propagates the security incidents impacts [96, 97].

2.5.5.1 Approaches Based on Existing Tools

Chung *et al.* [98] proposed NICE (Network Intrusion detection and Countermeasure sElection), a Cloud security framework based on an attack graph built using MulVAL [56]. A network-based intrusion detection system is installed on each physical server in order to generate alerts upon the detection of malicious traffic. These alerts are correlated based on the attack graph established, in order to determine the probable scenario currently used by the attacker and select the most appropriate countermeasures. The authors investigate the programmability of software switches-based solutions to improve security, but fail to consider how the context of the Cloud might affect attack graph generation. Indeed, each event in the environment, such as a vulnerability discovery, the deployment of a countermeasures or connectivity changes, triggers the reconstruction of the entire attack graph. On the other hand, no extension of MulVAL is proposed to include virtualization vulnerabilities in the graph generation.

On a similar note, Mjihil *et al.* [99] propose a Cloud security assessment tool which reuses MulVAL for attack graph generation. They consider the issue of nested virtualization, by which each virtual machine hosted in the Cloud infrastructure can itself have a virtualization software installed to provide additional virtual servers within the tenants infrastructure. The proposed framework consists in the deployment of a security agent within each virtual layer. This agent

is responsible for building the attack graph associated with that layer and passing the results to the agent in the parent layer. Agents allow a generation by parts of the graph, but authors do not indicate how the resulting pieces are merged to obtain a global view or what happens when virtual machines are able to cross physical server boundaries due to co-location and virtualization vulnerabilities. The dynamic dimension of the Cloud is also not considered, as well as the vulnerabilities relative to virtualization.

Alhebaishi *et al.* [100] applied various popular threat modeling approaches, among which attack graphs, to Cloud environments. Regarding the attack graph approach, the presented graphs seem similar to TVA, however no explicit mention of the methodology used for the generation is made. As the resulting graphs are specifically tailored to their environments, it is difficult to decide whether the graph is built manually or not. On the other hand, they do not consider explicitly the subtleties of virtualization vulnerabilities, or the dynamic aspect of Cloud environments.

2.5.5.2 Relevant Cloud Security Approaches not Relying on Attack Graphs

Alternative works, not focused on attack graphs, include in a better way the Cloud dimension and particularities in their approach.

Madi *et al.* [87] focus on virtualized infrastructures and tackle the verification of compliance properties such as the co-residency, co-ownership or virtualized ports consistency. They analyze data sources coming straight from the virtualized environment, and compare them with data from the Cloud management system and an SDN controller to check the proper deployment of the infrastructure.

Bleikertz *et al.* [88, 89, 101] aim to validate the correctness of instance configuration(s) from an isolation perspective (in the context of the Cloud). A flow analysis tool based on the extraction of virtual system configurations via a number of probes, and its transformation into a graph model is introduced. Based on generic or user-specific trust assumptions, the model is augmented with traversal rules, resulting in a representation allowing to identify unwanted information flows, i.e., isolation breaches, in the infrastructure. A differential analysis is performed when a change occurs, by comparing the newly obtained model and the policy to detect potential failures.

Both these works have a purpose that is different from our goal, however they have the advantage to properly integrate the Cloud challenges in their problem resolution. Madi *et al.* remain at the level of the topology, when for an efficient Cloud attack graph generation, we need both the topology and the connectivity in real time and represented in an exploitable format. Besides, their choice of processing the data retrieved in batch mode distances us from the real time property we expect from attack graph generation in such an environment. On the other hand, Bleikertz *et al.* consider the information flow in the infrastructure and address the dynamic evolution in their analysis, but their approach is narrowed in its application, as the probes introduced for data retrieval are hypervisor-specific.

2.6 Discussion on Attack Graphs in the Cloud

From the best of our knowledge, there is a lack of proper and comprehensive attack graphs usage is the context of the Cloud as presented in Section 2.5.5. The use of the Cloud implies the implementation of virtualization technologies to meet users expectations regarding this type of infrastructure. The promise of self-provisioning, elasticity and dynamism that accompanies the deployment of the Cloud is a real security concern. Even if multiple attack graph approaches have been proposed in traditional environments, they suffer from scalability issues and representation complexity and cannot be directly transposed to a Cloud context. The limited amount of works implementing attack graphs in the context of the Cloud reuse existing methods as-is, without considering the dynamic nature of the infrastructure or the additional attack surface that the hypervisor represents.

In addition, the attack graph generated must be able to include all dimensions of this novel environment. It should take into account virtualization attack scenarios, especially when they have the potential of impacting the assets of several companies at once. Understanding the impacts and conditions of these vulnerabilities represent the first roadblock tackled in our work.

We then aim to present the attack graph generated in the Cloud, in a clear and concise manner for an ease of analysis. To that end, we favor the advances made by the MulVAL tool and its partial state nodes, as well as the host-centric approach advocated by Amman *et al.* for system administrators and penetration testers.

We identify the fact of obtaining an updated view of the network connectivity as one of the roadblocks to the adoption of attack graph generation in Cloud environments. With the frequency of changes considered in this environment, an attack graph approach based on non-updated inputs would be useless. There is a need to design a way to obtain the current and most accurate network configuration, and this represents the third lock that we address in this thesis.

2.7 Conclusion

As we presented in Sections 2.6 and 2.5.4, existing attack graph solutions cannot be applied as-is in Cloud environments. Building an efficient attack graph solution in the Cloud context requires to address several challenges.

Existing models should be extended to include the virtualized dimension of the infrastructure. Besides, the right balance should be found in the model in order to avoid fragmenting excessively the information represented and add complexity to the graph. Finally, the necessity to have the most updated graph motivates the implementation of an efficient and online data gathering process, able to track the changes occurring in the infrastructure and adapt the inputs provided to the attack graph generation process.

In the rest of this thesis, we address the technological locks identified by the the state of the art analysis, and preventing the use of attack graphs in the Cloud. In Chapter 3, we present

the study performed to extend existing models and include virtualization vulnerabilities in the graph generation process. Chapter 4 describes the methods to retrieve an up-to-date network connectivity in the Cloud and ensure that this data remains relevant given the current network configuration. Chapter 5 presents the attack graph model chosen in the Cloud context. The experiments performed as well as the results obtained to validate our proposal are presented in 7.

CHAPTER 3

VIRTUALIZATION VULNERABILITIES

In this chapter, we will focus on the modeling of virtualization vulnerabilities, with the purpose of extending existing vulnerability classification in the context of virtualization. Virtualization technologies are at the core of the Cloud. Modern environments are still affected by traditional environment vulnerabilities but in addition, they have an increased attack surface due to the novel technologies used, and particularly the virtualization layer. The key enabler of Cloud computing being the hypervisor software, it is necessary to understand the conditions under which a vulnerability existing in the software can be triggered by an adversary in order to compromise the instantiated virtual machines. Indeed, previous studies uncovering such vulnerabilities showed that they require very strict prerequisites in order to be leveraged in an attack, be it, for example, a specific configuration the physical and virtual machines considered. This insight represents a necessity, especially when resources are shared and accessible to a diverse range of users in the Cloud, some with potentially malicious motivations. The absence of such understanding is clearly a drawback of existing attack graph models since they ignore the specific attack surface that the hypervisor represents. As a result, attack paths generated in that context miss the virtualization layer, which is critical in the Cloud.

Perez-Botero and al. [102] performed a work close to ours. However, their classification strongly focus on 11 functionalities that traditional hypervisors provide, and that they mapped to the identified vulnerabilities. These functionalities include *Soft Memory Management Unit*, *Interrupt and Timer Mechanisms* and *Hypercalls*. This constitutes a representation that is designed to pinpoint which functionalities of the hypervisors are leveraged in a vulnerability, and not under which conditions the vulnerability can be exploited. With this classification, two vulnerabilities relying on different functionalities would still be identified as potentially exploitable by an attacker, since the hypervisor would by default expose these functionalities.

What would allow to distinguish between exploitable and non-exploitable vulnerabilities is really the pre-conditions required to exploit the vulnerabilities, not the hypervisors functionalities. We propose a higher level classification suitable for an exploitation in the context of attack graph construction, even if we find to some extent some of Perez-Botero *et al.* categories in ours.

We conduct an extensive analysis of real-world virtualization vulnerabilities, reported in common vulnerability databases and security advisories of the hypervisors considered. By studying every report, we isolated categories of conditions which are either required to exploit these vulnerabilities, or which are fulfilled upon exploit success. In our environment, we consider the SDN controller and the Cloud Management System as being trustworthy. This hypothesis is required to be able to properly analyze attacks existing in the infrastructure and provide reliable countermeasures, since by compromising these orchestration layers, attackers can tamper with their outputs and hence the security mechanisms.

We first present deployed hypervisors used during our study, then the threats incurred by virtualization vulnerabilities. In another section, we present the categories extracted from the vulnerability description analysis, and finally the results obtained for each hypervisor.

3.1 Hypervisor Background

Several key players exist now in the public Cloud landscape: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform or VMWare vCloud. Their choices regarding the hypervisors powering their Cloud infrastructures contribute to the widespread adoption of some hypervisors technologies over the others. While some public Clouds developed proprietary blends of hypervisors to power their infrastructures, they still use existing open-source software as a foundation for their home-brewed versions: AWS with Xen and now KVM, Google Cloud Platform with KVM. On the other hand, closed source hypervisors such as Hyper-V and VMware ESX also have a large footprint in the virtualization landscape by providing their own platforms, respectively Microsoft Azure and VMware vCloud. As a consequence, understanding their associated vulnerabilities has the potential to benefit to millions of users worldwide. In this section, we will feature bare-metal hypervisors most commonly found on these public Clouds and present their architecture.

3.1.1 Xen Hypervisor

Xen is an open source bare metal hypervisor that follows the least privilege principle, by providing multiple purpose-built components.

Every Xen deployment creates a *Control Domain*, also called *Dom0*. It is a privileged virtual machine able to directly access the hardware, handle accesses to the I/O functions and interact with other VMs. It includes a toolstack, responsible for performing settings and virtual machines management operations. Additionally, native device drivers interfaces can be found in Dom0. Vir-

tual device drivers, also called backends, are also contained in Dom0, their frontend counterparts being implemented in the virtual machine requesting a particular driver.

Xen was the first to foster a strong cooperation between virtual machines and hypervisor through ParaVirtualization (PV) as presented in Section 2.5.2.1.

It also offers Hardware-assisted Virtual Machines (HVMs) that use the virtualization extensions from the CPU to boost emulation performances and QEMU device models to emulate the devices. Upon the success of the paravirtualization approach, Xen made improvements to HVMs to be able to use PV drivers and increase the performances for older operating systems.

In an effort to combine the advantages of both PV and HVM modes, Xen developed a novel type of virtual machine called PVH. It consists in a lightweight HVM using hardware virtualization support for memory and privileged instructions, while using paravirtualized drivers for I/O, native operating system interfaces and everything else.

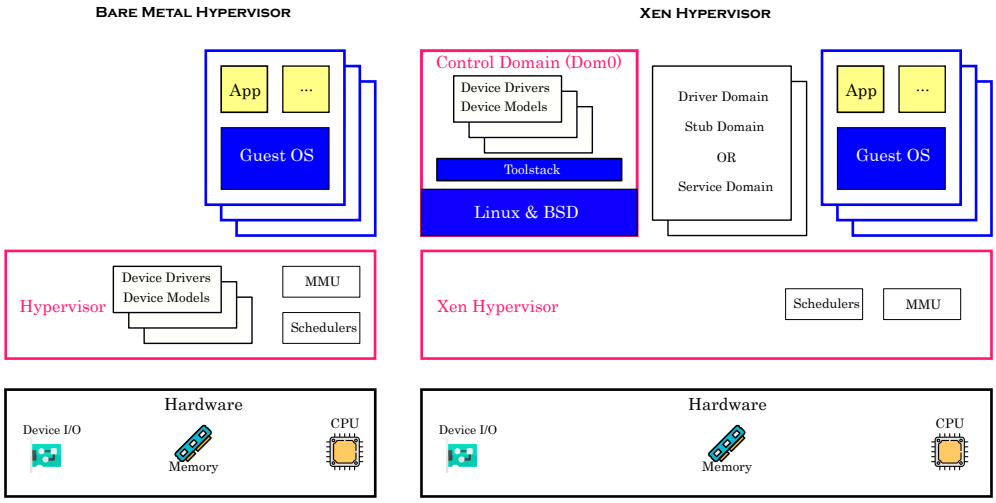


Figure 3.1: Comparison between bare-metal and Xen architecture

3.1.2 Kernel-based Virtual Machine (KVM) Hypervisor

KVM is an open source virtualization technology deployed on Linux hosts. It presents itself as a loadable kernel module which provides the core virtualization infrastructure and processor specific modules for the use of hardware support virtualization. The intrusiveness is minimized by turning the Linux kernel itself into a hypervisor through a loadable component. In this context, each virtual machine created is a regular Linux process, and thus benefits from the host kernel isolation mechanisms and scheduling optimization. KVM extends the traditional kernel versus user execution modes to introduce a guest mode for applications ran from within a virtual machine.

In addition to the kernel module, every KVM deployment includes by default a slightly modified QEMU component running in user space for I/O hardware emulation.

KVM does not support the execution of paravirtualized virtual machines (with modified kernels), however it leverages paravirtualized drivers in the guests just like Xen, via a native integration of these drivers in the Linux kernel.

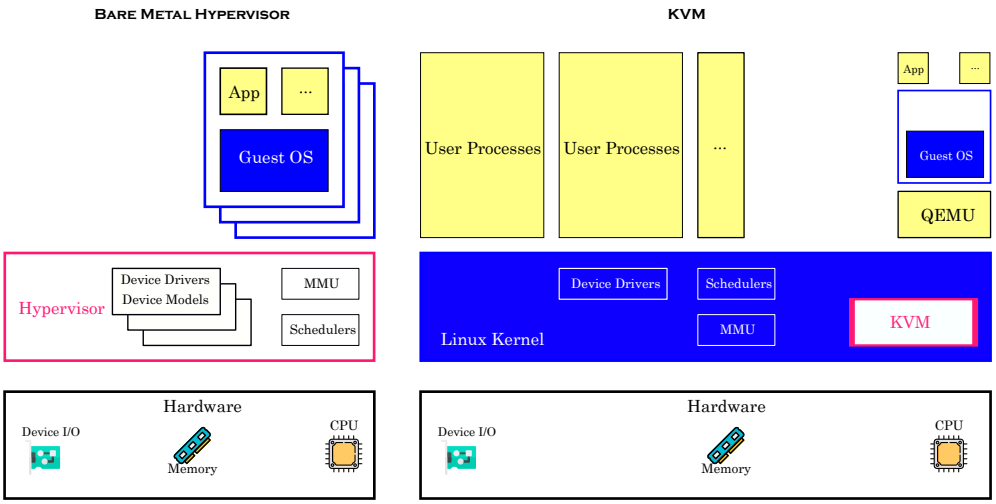


Figure 3.2: Comparison between bare-metal and KVM architecture

3.1.3 Hyper-V Hypervisor

Hyper-V is the proprietary Type I hypervisor solution proposed by Microsoft. A Hyper-V deployment creates a *root partition* also called *parent partition* which has a similar mandate as Xen's Dom0: direct access to the hardware and virtual machine management. It is also responsible for exposing I/O devices to the virtual machines, in addition to implementing either Hyper-V specific devices or emulated devices.

Virtual machines are created through the parent partition and are identified as child partitions. Hyper-V can host two types of virtual machines: enlightened, i.e. hypervisor-aware, and unenlightened virtual machines. In an enlightened machine, a Virtual Service Client (VSC) communicates with a virtual service provider (VSP) in the root partition, in order to provide device access to the virtual machine. Unenlightened virtual machines on the other hand only rely on hardware emulation.

3.1.4 VMware ESX Hypervisor

VMware ESX hypervisor is a proprietary hypervisor featured by VMware.

It implements an underlying operating system called *VMKernel* [1] and able to run additional processes, namely management applications, remote management agents and virtual machines. It has control over all hardware devices and manages the resources. This kernel also contains device driver modules adapted to the hardware.

[1] https://microage.com/wp-content/uploads/2016/02/ESXi_architecture.pdf

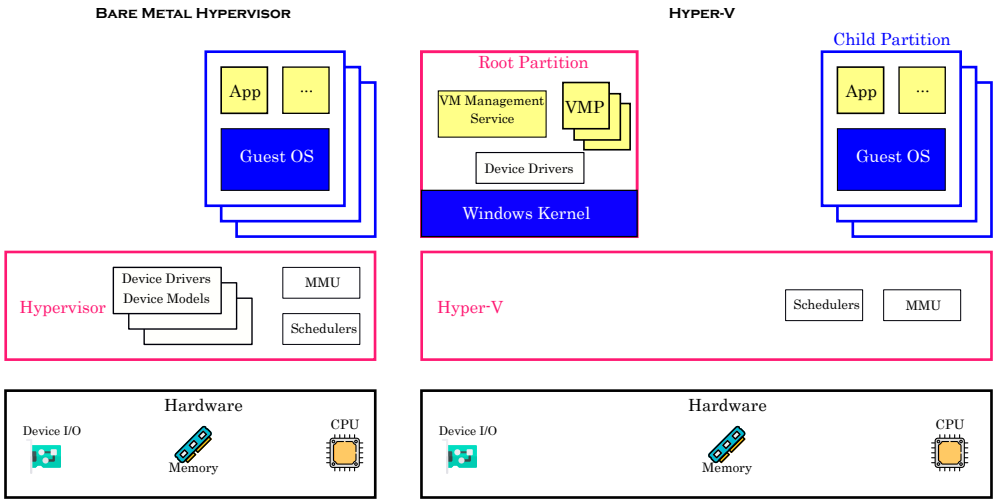


Figure 3.3: Comparison between bare-metal and Hyper-V architecture

Contrarily to Xen and Hyper-V philosophy, in which the drivers are decoupled from the hypervisor code, each virtual machine sees its own set of driver devices within the kernel. Every machine is paired with a Virtual Machine Monitor, responsible for implementing its virtual CPUs. Similarly to KVM, VMware does not support running paravirtualized guests, but can benefit from paravirtualized drivers used in the context of full virtualization.

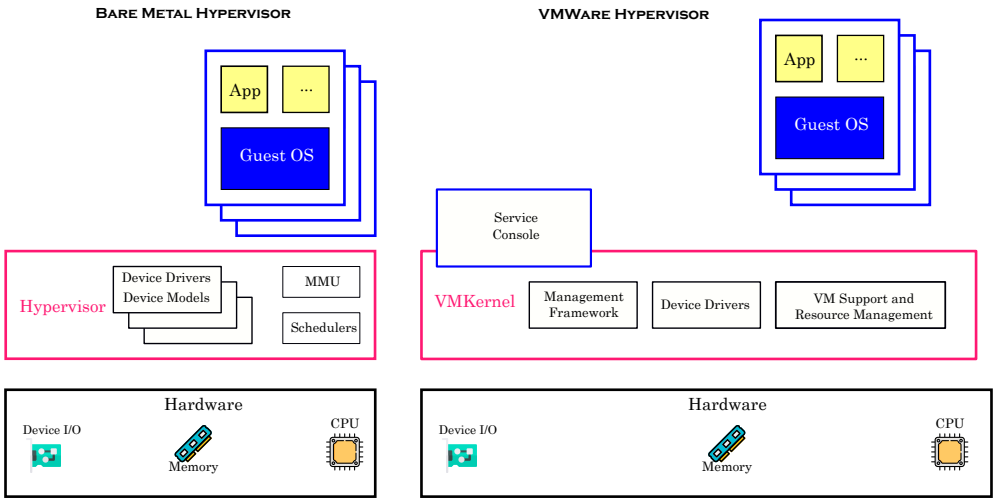


Figure 3.4: Comparison between bare-metal and VMWare ESX architecture

3.1.5 Conclusion

A diversity of hypervisors can be found in datacenters, each with their own architecture and means to realize virtualization. Nonetheless, recurring conditions enabling the exploit of their vulnerabilities emerged from the thorough analysis of vulnerability databases and security

reports associated with these software. In the next section, we will present the threats in the context of virtualization vulnerabilities.

3.2 Threats Against a Virtualized Cloud Environment

In addition to the ones existing in traditional networks, an attacker targeting a virtualized environment benefits from a larger range of options due to the emergence of novel attack scenarios. Moreover, by identifying the environment as virtualized, the attacker can craft specialized malware and attacks to take advantage of this particular environment. VM-Based Rootkits (VMBRs) are one instance of such malware [103], as well as the *Crisis* malware [104], that attempts to spread specifically to virtual machines installed on an infected host.

In this section, we regroup virtualization-based vulnerabilities specifically, as discussed in surveys such as [18, 105–107] and pertaining to security issues related to virtualization.

3.2.1 Fingerprinting the environment

By identifying the running environment, an attacker is able to focus his effort in the most effective scenario available for its target. Since in practice, real and virtualized environments are not exactly identical despite Popek and Goldberg requirements presented in Section 2.5.2.1, execution time measurements for some instructions or access to the Translation Lookaside Buffer (TLB) can be used to distinguish between virtualized and non-virtualized machines, as proposed by Ferrie [108]. Knowledge gathering can be as precise as detecting the specific hypervisor software deployed on the physical machine by discriminating between the way each of them handles particular instructions.

3.2.2 Subverting the Hypervisor

Being the control point of the infrastructure, the Virtual Machine Management (VMM) is a sensitive item in a virtualized environment, as it is responsible for VM management and monitoring. By gaining access to this component, an attacker can not only access the VM network traffic passing through that host, but also their memory, and even modify VM resources or states, i.e, active, paused, shutdown.

3.2.3 Virtual Machine Escape

This implies a breach of the isolation property enforced by the hypervisor. This possibility is of great concern given the shared nature of the infrastructure, especially in public Clouds. Several scenarios are feasible in this case:

- Guest to guest escape: A malicious VM is able to leverage a vulnerability to access neighboring virtual machines. In doing so, an attacker can impact a wide range of companies by leveraging a single exploit;
- Guest to host escape: In that case, a corrupted VM is able to break the boundaries enforced by the hypervisor to access the underlying physical machine. Since the hypervisor is the entity controlling all the hosted VMs, it is possible for an attacker to completely take over these virtual machines.

These two items are feared scenarios presented in several surveys. However, depending on the characteristics of the hypervisors deployed, the scope of the escape can be narrowed down. Indeed, some hypervisors implement the least privilege principle by decoupling the management functions and assigning them partially to specialized controlling domains which are isolated from the main hypervisor layer. Hence a VM escape results in less privileges obtained compared to a monolithic hypervisor.

3.2.4 Attacks on Virtual Machines during Migration

A novelty of virtualization is the ability to transfer an active or paused VM from one host to another for maintenance or security purposes, as well as resource optimization. Since the migrated VM has to transit through the network, if no security measures are taken, its data can be intercepted through the network by malicious users, resulting in a data breach and potential compromise of the credentials it contains.

3.2.5 Virtual machine data extraction

Ristenpart *et al.* [109] proposed approaches to trigger virtual machines co-residency, enabling an attacker VM to compromise a VM on the same hypervisor as its source. By enforcing this co-residency, an attacker is able to leverage cross-VM information flows to steal the victim data, including authentication data, encryption keys or sensitive information. Inactive VMs also have to be considered, since their memory content is often stored on disk and potentially accessible if no encryption measures are taken.

3.2.6 Denial of Service by Resources Starvation

Since the infrastructure is shared, virtual machines may see a depletion in the amount of resources assigned to them, depending on the behavior of neighboring virtual machines. A malicious VM can generate a large workload consuming CPU, memory and network bandwidth on the underlying host, with the intention to perform a Denial of Service attack on the other virtual machines sharing the same physical resource. The intent can also be to trigger a resource re-optimization to ensure co-residency with specific virtual machines as explained by [109].

3.2.7 Patching Weaknesses

A stricter vulnerability patching policy is required in virtualized environment. Indeed, VMs can exist at different locations. Even if administrators succeed in applying security patches to all of the vulnerable ones at a given point in time, their states can be saved and rolled back to insecure snapshots.

3.2.8 Conclusion

We presented here the threats to virtualization at a macro-level, focusing on giving the big picture of their impacts on the infrastructure. However, the majority of these threats are enabled by factual low-level bugs existing in the virtualization layer. Hence, we adopt an empirical approach based on real vulnerabilities reported, with the objective to extract the requirements specific to the exploit of a virtualization vulnerabilities by an attacker, and extend the vulnerability modeling used in the context of attack graphs to virtualization vulnerabilities.

Other research analyzed the databases of vulnerabilities reported for virtualization in an exhaustive fashion, but with different goals. In an effort to better understand security characteristics relative to hypervisors and container technologies, Gkortzis *et al.* [110] are interested in classifying them according to categories used in the Common Weakness Enumeration (CWE) database, and not an analysis of their prerequisites and effects.

Similarly, Perez-Botero *et al.* [102] performed a work close to ours, however, their classification strongly focus on mapping the common hypervisors functionalities to the vulnerabilities, and not determining their requirements.

We propose a higher level classification suitable for an exploitation in the context of attack graph construction, by not diving into the virtualization software to uncover low-level attack vectors. However, by breaking down the vulnerabilities into trigger sources and attack targets instead, Perez-Botero *et al.* exhibit categories of prerequisites and effects that we indeed find in our approach. In the next section, we will describe in which categories these conditions fall, as the classification of these conditions accompanies the effort of generating virtualization vulnerability templates.

3.3 Categorizing the Conditions for Exploitation of Virtualization Attacks

We searched the National Vulnerability Database (NVD) from the NIST to gather and analyze vulnerability reports for VMWare ESX hypervisor, Hyper-V and KVM. For Xen, we used the security advisories released by the software manufacturers directly. Our analysis revealed three primary components whose characteristics permit an attacker to exploit a vulnerability, and which may be impacted by this exploit: the physical server, the hypervisor software and the

virtual machine. These categories can be classified into pre-conditions required to successfully exploit the vulnerability and post-conditions that result from a successful exploit.

3.3.1 Pre-conditions of a Virtualized Vulnerability Exploit

3.3.1.1 Hypervisor Version

Despite some vulnerabilities being identified as affecting all versions of the hypervisor software, the majority of them is associated with a particular version of the hypervisor considered and the level of patches already applied to mitigate the weaknesses previously found.

3.3.1.2 Hypervisor Toolstack

In the specific case of the Xen hypervisor, management capabilities can be provided using a diverse range of toolstacks. Each one of them exposes an API, with specific tools in charge of handling the most common low-level operations for the users. As the choice of a toolstack is particular to each Xen deployment, the various bugs found in the toolstack code base only affect the hypervisor environment in which they are deployed.

3.3.1.3 Hypervisor and Guest Options

Hypervisors and guest virtual machines can be run with a wide range of different parameters which allow to extend their functionalities. A vulnerability found in the code related to a particular parameter is only enabled when the hypervisor or the virtual machine is run with that parameter set up.

3.3.1.4 Quotas

The hypervisor is in charge of managing the resources provided to the virtual machines running on the physical host. Some vulnerabilities can only be triggered when a specific threshold of resource allocated to a VM is exceeded, causing either memory leaks or Denial of Service.

3.3.1.5 Domains Enabled

Xen is an hypervisor assigning different roles to the virtual machines (domains) running on the system. It first implements a privileged domain called Dom0, and able to access the hardware, run hardware drivers and manage other domains. These other domains are the users virtual machines (DomU) and are unprivileged. Instead of aggregating all the drivers within the Dom0, specific driver domains can be created. They represent unprivileged Xen domains, responsible for a particular piece of hardware. This prevents the control domain from being not only a bottleneck, but also a single point of failure for the virtualized environment. Besides, since there is a higher risk of having a vulnerability in the driver code base, there are much less benefits gained for the

attacker by subverting a domain via driver code, when this code is decoupled from the control domain.

3.3.1.6 CPU Architecture

Two main categories of processor architectures exist, namely the 32-bit and 64-bit architectures. Their denomination indicates memory words size, as well as data buses, instruction and memory size. The functioning of the applications is also affected by the processor's nature, as with a 32-bit architecture, the OS and applications work with 32-bits-wide data units, compared with 64-bits-wide data units for a 64-bit architecture. The type of processor impacts not only on the performances, but also the type of software able to run. Indeed, a computer with a 32-bit processor is unable to run a 64-bit OS or software version, even if the reverse is true to some extent. However, running a 32-bit software version on a 64-bit processor would under-utilize the CPU and not allow it to function at full capacity.

The host CPU architecture or the guest CPU architecture chosen at the creation of the virtual machine can be an essential criterion in determining whether a vulnerability is accessible to an attacker or not.

3.3.1.7 Processor Manufacturer

The processors found in computers are designed and manufactured by several companies. The most prominent ones are Intel and AMD for the x86 architecture, and Arm Holding for the ARM architecture. Depending on the processors installed on the physical server running the hypervisor, some vulnerabilities in the virtualization software can be exploited.

3.3.1.8 Operating System

The Operating System (OS) is the primary software on a computer, in charge of interfacing the computer hardware with the computer software run by the users. It manages hardware resources such as network and storage devices. It generally features three essential components. On one hand, the kernel in control of the hardware devices. It is responsible for reading and writing data to and from memory, processing operations and handling the computer inputs and outputs. On the other hand, a user interface allows the user to interact with the OS through command lines or a graphical user interface. Lastly, to enable additional applications to be run on the hardware, the OS provides a set of Application Programming Interfaces (APIs) used by developers to write modular code.

In the context of virtualization, even if the hypervisors deployed within datacenters are qualified of bare-metal, i.e., installed directly on the hardware, they are often used in conjunction with an operating system. This is either to benefit from existing OS functions such as the native Linux scheduler or memory management services with KVM, or to implement modularity with Xen.

While examining hypervisor-related vulnerability descriptions, we isolated reports in which the Operating System chosen for the host, or at the creation of a virtual machine is a pre-condition to the attacker's ability to leverage the vulnerability identified. As a result, the choice of OS has an impact on an attacker's ability to exploit a vulnerability.

3.3.1.9 Virtual Machine Types

As presented in Section 2.5.2.1 introducing the different virtualization methods, virtual machines can be created based on various methods: full-virtualization, hardware-assisted virtualization or paravirtualization. From the type chosen for the VM creation can depend the ability of an attacker to exploit a particular vulnerability. Indeed some are only accessible to modified or enlightened VMs (paravirtualized), others to fully virtualized ones, or VMs leveraging virtualization facilities provided by the processor manufacturers.

3.3.1.10 Devices and Drivers Enabled

For a virtual machine to function similarly to a physical machine, it has to provide I/O devices to the users for sending information to the system for processing, and displaying the result of that processing. Bugs can be found in the code designed to provide the virtual devices to the virtual machines, but can only be exploited if the concerned devices are effectively requested by the VMs.

3.3.1.11 Level of Privileges

All users actions on a computer are affected with privileges, which determine the actions they are allowed to perform. This can include reading from or writing to a file, accessing a device, or executing a program. As presented in Section 2.5.2.1, processors have several modes allowing Operating Systems to run at different privilege levels. The execution of system calls at a lower privilege than the one required is prevented. Hence the primary goal of an attacker getting a foothold on a machine is to gain an elevated access to protected resources by the use of exploits.

Privilege levels identified in the context of virtualization vulnerabilities are related to privileges granted on either the physical machine, the virtual machine or the emulation software, such as QEMU, used to provide drivers to the VMs.

3.3.2 Post-conditions of a Virtualized Vulnerability Exploit

3.3.2.1 Vulnerability Effects

The successful realization of an exploit targeting a virtualization vulnerabilty can cause one or several issues including Denial of Service (DoS), memory leaks, data corruptions, privilege escalations or arbitrary code execution.

3.3.2.2 Vulnerability Targets

Contrarily to traditional environments in which vulnerabilities only impact the physical machine on which the weaknesses exist, in the Cloud context, the vulnerability scope is much wider. It can impact not only the physical host, but also the virtual ones. Depending on the hypervisor chosen, it can impact some of the components used to realize the virtualization such as the driver domains for Xen, the emulation software or the management consoles.

3.3.2.3 Level of Privileges

The privilege level represents not only a requirement for the use of a vulnerability but also a consequence of the realization of a successful exploit. Indeed, a first access on the system, however low, must first be gained for an attacker to use it as a basis for an exploit, which could result in higher privileges granted.

3.4 Hypervisor Vulnerability Study

We present the results of the study performed on over four hundreds vulnerabilities linked to virtualization, with pre-conditions represented in Table 3.1 and post-conditions in 3.2. The number of occurrences per category of requirements identified for each hypervisor software is reported in the tables, with the percentage relative to the total number of vulnerabilities analyzed for that particular hypervisor.

The most detailed vulnerabilities descriptions were obtained for the Xen hypervisor. In addition to entries in the National Vulnerability Database (NVD), security advisories are even available, with a thorough presentation of the bugs encountered and means to resolve them. We took advantage of this detailed information to extract the categories of requirements presented earlier. Over the course of our study, we noticed that the content of NVD entries for KVM, Hyper-V and VMWare ESX hypervisors could be mapped to these categories. The lack of details in the descriptions, as well as the dissimilarities in hypervisor architectures explains the null occurrence of some requirements identified for KVM, Hyper-V and VMWare ESX hypervisors.

These requirements are relative to four major components: exploits are enabled by characteristics pertaining to the physical machine, the virtual machine, the hypervisor and the attacker. Physical machine properties are the host architecture and operating system as well as the processor architecture and manufacturer. Virtual machine properties are the VM type, architecture and operating system, as well as the quotas, device drivers and options affected to them by the hypervisor. Hypervisor properties are relative to the available management toolstacks, activated options and the existence of a decoupled architecture enabling specific control domains. Attacker properties are the privileges required, the desired effect of the vulnerability and the targeted component.

		Xen Nb occurences and Percentage	VMWare ESX Nb occurences and Percentage	Hyper-V Nb occurences and Percentage	KVM Nb occurences and Percentage
Hypervisor Version		240 (100%)	87 (100%)	54 (100%)	96 (100%)
Vulnerable VM Types	HVM	95 (39.58%)	0	0	0
	PV	54 (22.5%)	0	0	1 (1.04%)
	PVH	6 (2.5%)	0	0	0
	PVHVM	1 (0.42%)	0	0	0
Guest Architecture	32 bits	9 (3.75%)	2 (2.30%)	0	0
	64 bits	14 (5.83%)	2 (2.30%)	0	0
Host Architecture	32 bits	5 (2.08%)	0	0	0
	64 bits	10 (4.16%)	0	3 (5.56%)	0
VM OS		16 (6.67%)	8 (9.20%)	0	0
Hypervisor OS		0	0	0	0
Toolstack Used		38 (15.83%)	8 (9.20%)	0	0
Privileges Required	User on VM	129 (53.75%)	44 (50.57%)	38 (70.37%)	64 (66.67%)
	Root on VM	85 (35.42%)	0	11 (20.37%)	1 (1.04%)
	Kernel on VM	6 (2.5%)	0	0	0
	Remote	2 (0.83%)	28 (32.18%)	1 (1.85%)	0
	Driver	3 (1.25%)	0	0	0
	User on VMM	0	13 (14.94%)	5 (5.75%)	31 (32.29%)
	Root on VMM	20 (8.33%)	0	0	0
Processor Manufacturer		22 (9.17%)	0	0	15 (15.63%)
Processor Architecture	x86	116 (48.33%)	0	0	47 (48.97%)
	ARM	35 (14.58%)	0	1 (1.85%)	2 (2.08%)
Quotas		12 (5%)	0	0	0
Device Drivers Enabled		42 (17.5%)	12 (13.79%)	0	15 (15.63%)
Guest Options		34 (14.17%)	2 (2.30%)	0	0
Hypervisor Options		21 (8.75%)	2 (2.30%)	0	0
Specific Domains	Control Domain	12 (5%)	0	0	0
	Stub-domains	30 (12.5%)	0	0	0
	Driver Domains	6 (2.5%)	0	0	0
Number of vulnerabilities for each hypervisor		**240**	**87**	**54**	**96**

Table 3.1: Summary of the pre-conditions obtained during the analysis performed on 477 virtualization vulnerabilities

It emerges from this study that an exploit can have a more or less restricted scope of impact on the infrastructure. The attacker can affect either the virtual machine, the hypervisor, the management tools, specific domains (in case of decoupled architecture) or facilities providing the virtualized devices. Of course, depending on the vulnerabilities, several or all of these components can be affected.

A summary of the different vulnerabilities targets and attack vectors can be found on Figure 3.5, which is an extension of Figure 1 of Zhang *et al.* [111].

It appears difficult to oppose the hypervisors to one another, since the proportion of vulnera-

		Xen Nb occurences and Percentage	VMWare ESX Nb occurences and Percentage	Hyper-V Nb occurences and Percentage	KVM Nb occurences and Percentage
Vulnerability Effects	DoS	167 (69.58%)	36 (41.38%)	20 (37.04%)	81 (84.38%)
	Code Execution	9 (3.75%)	28 (32.18%)	17 (31.48%)	10 (10.42%)
	Privilege Escalation	91 (37.92%)	23 (26.44%)	5 (9.26%)	22 (22.92%)
	Memory Corruption	15 (6.25%)	6 (6.90%)	0	6 (6.25%)
	Data Leak	67 (27.92%)	11 (12.64%)	13 (24.07%)	11 (11.46%)
Vulnerabilities Target	VM	72 (30%)	43 (49.43%)	46 (85.19%)	47 (48.96%)
	VMM	184 (76.67%)	43 (49.43%)	12 (22.22%)	51 (53.13%)
	QEMU	8 (3.33%)	0	0	1 (1.04%)
	on Management Tools	4 (1.67%)	5 (5.75%)	0	0
	on Specific Domains	10 (4.17%)	0	0	0
Number of vulnerabilities for each hypervisor		**240**	**87**	**54**	**96**

Table 3.2: Summary of the post-conditions obtained during the analysis performed on 477 virtualization vulnerabilities

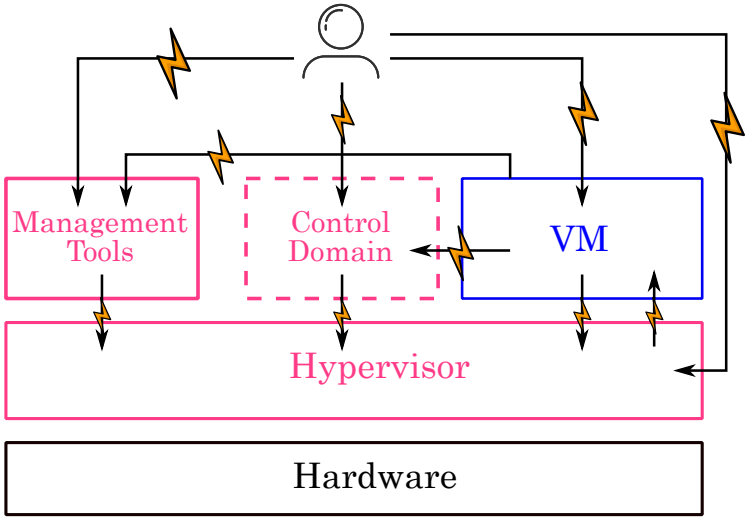


Figure 3.5: Virtualization vulnerabilities targets and attack vectors

bilities disclosed for each of them is not comparable, a fact that can be explained by the open or closed nature of their source code, as well as the traction of the open source community to use one over the other. This latter fact indeed leads to more ease of analysis and hence scrutiny of one technology over the other.

With this observation in mind, we can however derive from the study of Table 3.2 that for all hypervisors considered, the risk of Denial of Service is the most prevalent consequence an exploit can have, hence endangering the availability of services provided by the Cloud infrastructure. With Xen hypervisor appearing weaker than other hypervisors against privilege escalation (37.92%) and data leaks (27.92%) and VMWare seeming more vulnerable to code executions (32.18%) and memory corruption (6.90%), attackers can implement stealthier attacks in an

attempt to compromise the infrastructure over a longer period of time and get higher value from their network intrusion, by getting access to or stealing data from neighboring virtual machines. With a single vulnerability, an attacker can have several effects on the infrastructure, as well as disrupt several targets, reason why the sections about vulnerabilities effects and targets amount to more than 100%.

We learn from Table 3.1 that more than 10% of the vulnerabilities reported for all hypervisors, except Hyper-V, are related to specific device drivers enabled within the virtual machine. Depending on the vulnerabilities, only some type of virtual machines can be successfully exploited by an attacker. Regarding the Xen hypervisor, for which we have more details about this property, 39.58% of the reported vulnerabilities are only available to HVM virtual machines. Since there is a strong dependency between the physical machines and the virtual machines, via the hypervisor, we notice that over 48% of vulnerabilities reported for Xen and KVM require the presence of an x86 processor, while this number is of 14.58% for the presence of an ARM processor.

Figures 3.6, 3.7, 3.8 and 3.9 are a visual representations of the results presented in Tables 3.1 and 3.2, with the requirements in ordinates and the vulnerability number in abscissa. A dot represents the presence of this requirement for the vulnerability concerned. Some category of requirements having several options, the radius of the dot is adapted according to the number of options that the vulnerability validates.

3.5 Conclusion

From the thorough analysis of over four hundred vulnerabilities pertaining to virtualization, we can conclude that very specific requirements need to be met before an attacker is able to efficiently leverage an exploit and compromise the infrastructure. A clear understanding of these requirements is valuable since it allows a more accurate modeling of these vulnerabilities. It is worth spending the extra effort to model them accurately, because considering vulnerabilities that cannot be triggered by an attacker has the potential of creating a larger amount of links than in traditional environment, since the impact can be immediate on all the machines hosted on the vulnerable hypervisor.

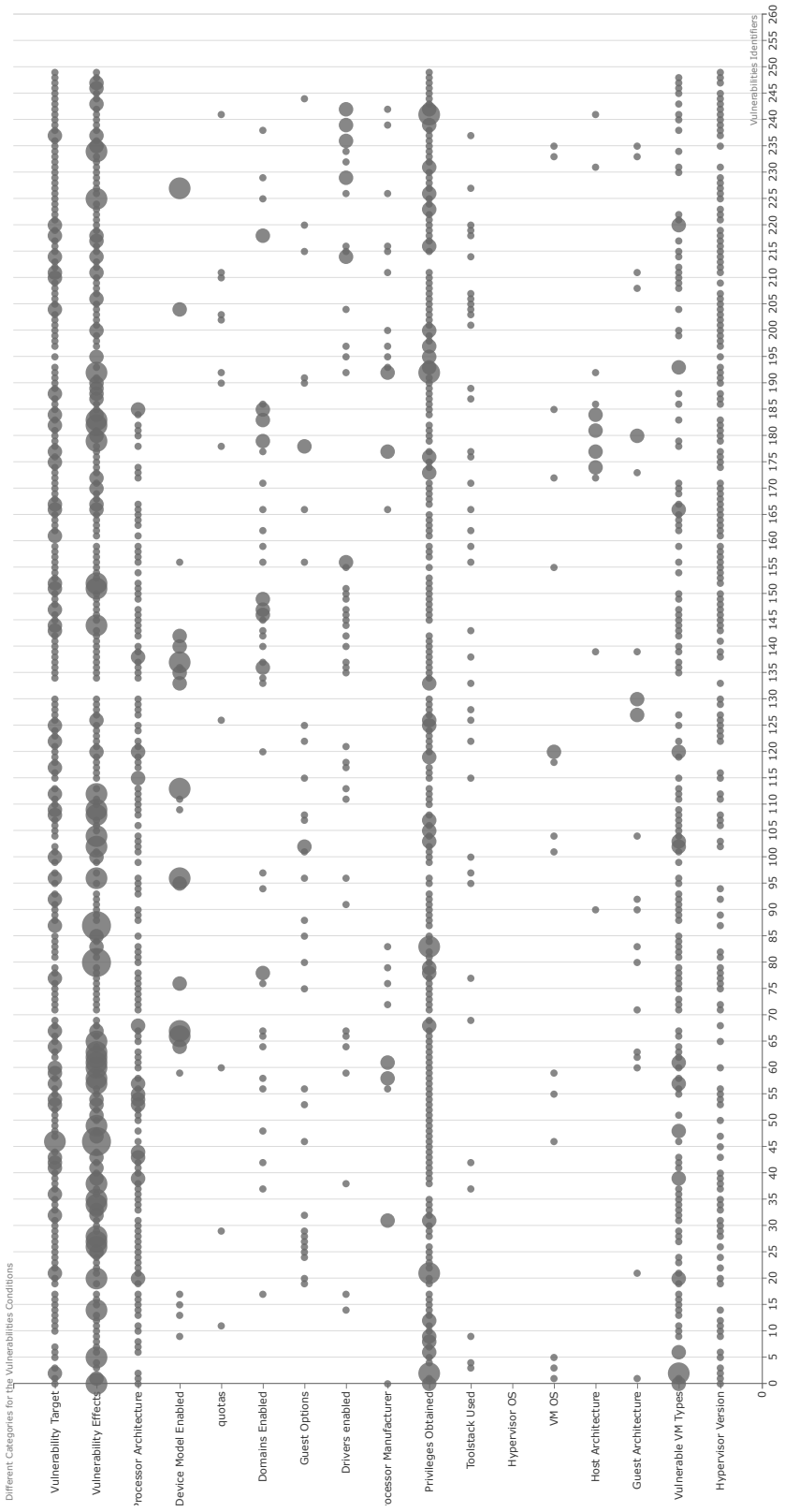


Figure 3.6: Categories of Conditions for Virtualization Vulnerabilities on Xen

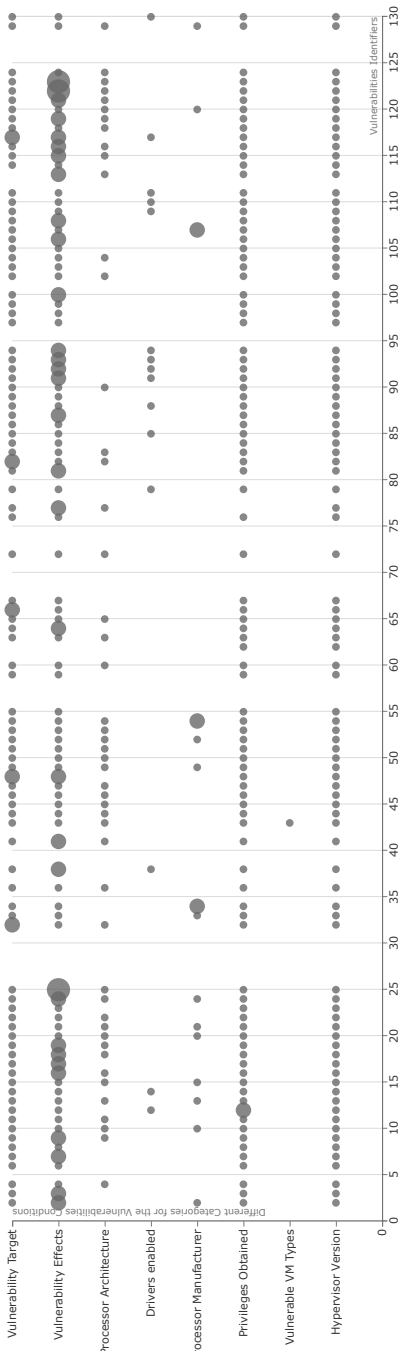


Figure 3.7: Categories of Conditions for Virtualization Vulnerabilities on KVM

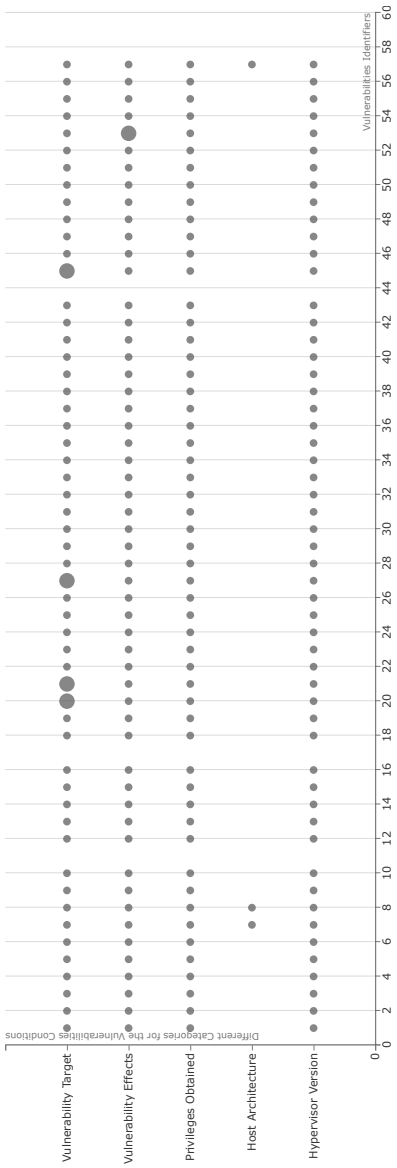


Figure 3.8: Categories of Conditions for Virtualization Vulnerabilities on Hyper-V

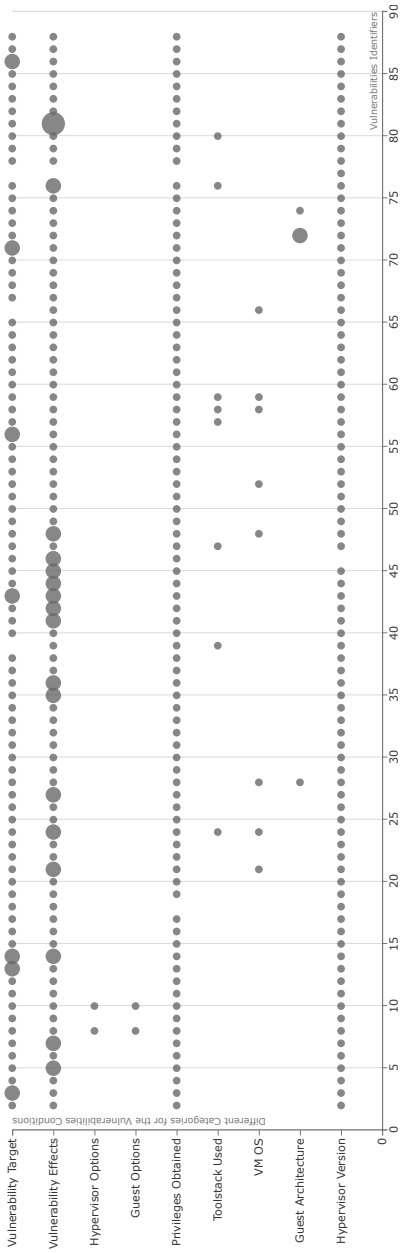


Figure 3.9: Categories of Conditions for Virtualization Vulnerabilities on VMWare ESX

CHAPTER 4

TOPOLOGY AND CONNECTIVITY CONSTRUCTION IN THE CLOUD

The connectivity of the devices existing in the infrastructure is a crucial input required for the construction of an up-to-date attack graph in the context of the Cloud. This requires building a connectivity graph between the virtual machines, that we can extract with the knowledge of the overall topology and the deployed network security policy. Existing methodologies for building such models for physical networks can produce incomplete results. Moreover, they are not suitable for Cloud infrastructures due to either their intrusiveness or lack of connectivity discovery, as presented in Section 2.1. We propose a method to compute the connectivity graph, relying on information provided by both the Cloud Management Platform (CMP) and the SDN controller. Connectivity can first be extracted from knowledge databases, then dynamically updated on the occurrence of Cloud-related events.

4.1 Connectivity Reconstruction in Cloud Environments

In this section, we will present the different steps used to reconstruct the connectivity of the Cloud infrastructure devices, based on information gathered from the Cloud Management Platform and the SDN controller.

4.1.1 Presentation of the Data Sources

The information needed for connectivity reconstruction in the Cloud can be found in the Cloud Management Platform, but also in the SDN controller, if one is deployed in the infrastructure. These data sources are presented in the rest of this section, as well as the challenges caused by their integration.

4.1.1.1 The Cloud Management Platform

A Cloud Management Platform is a suite of integrated tools used to monitor and control Cloud computing resources. Open source platforms as well as paying ones can be found, while some providers would rather develop a tailored management system. As such, CMPs greatly vary in terms of features offered to administrators and users. However, some common services are found in all of these toolsuites.

CRM Trilogix made a comprehensive presentation of the Cloud Management Platform [10] and the interactions of the different components with each other. It generally follows a modular design, allowing to activate or deactivate services according to the needs. CMP modules can be organized according to three functional layers:

- The Cloud portal and self-service portal: it is the graphical interface on which resources can be requested, manage and track their Cloud service subscription. Self-service capabilities are provided to the users via this portal, in addition to analytic features exposing consumption patterns in the environment;

- The automated provisioning, orchestration and service design layer is responsible for the resource allocation and service provisioning, the metering and billing of the resources consumed by the tenant, the service brokering, aggregation and arbitration in case of collaboration between multiple Clouds. The service management allows to monitor Cloud-based services to help with capacity planning, workload deployment and ensuring the fulfillment of availability and performance requirements;

- The network operations and management layer is responsible for resource management and asset configuration, network and security monitoring, in addition to service provider performance and status monitoring. Governance and security capabilities are implemented to allow Cloud administrators to enforce policy-based control of Cloud resources, and offer security features, namely identity and access management or encryption.

A detailed functional architecture of a Cloud management system proposed by CRM Trilogix is depicted on Figure 4.1, representing the common CMP functions. It is only for illustration purposes, as the CMP architecture is dependent on the software vendor.

As orchestration activities occur both before and after automation activities, to ensure for instance resources availability and proper virtual architecture deployment, two orchestration layers are represented on Figure 4.1, framing the automation layer. The CMP is designed to integrate seamlessly with third-party Cloud services if necessary.

Depending on the CMP vendors, more or less features can be available to the administrators, but we understand that generally, the CMP is truly the control center of the whole Cloud infrastructure. As such it is a centralized data source in which we can tap in order to gain valuable insights in the tenants' and Cloud's infrastructure.

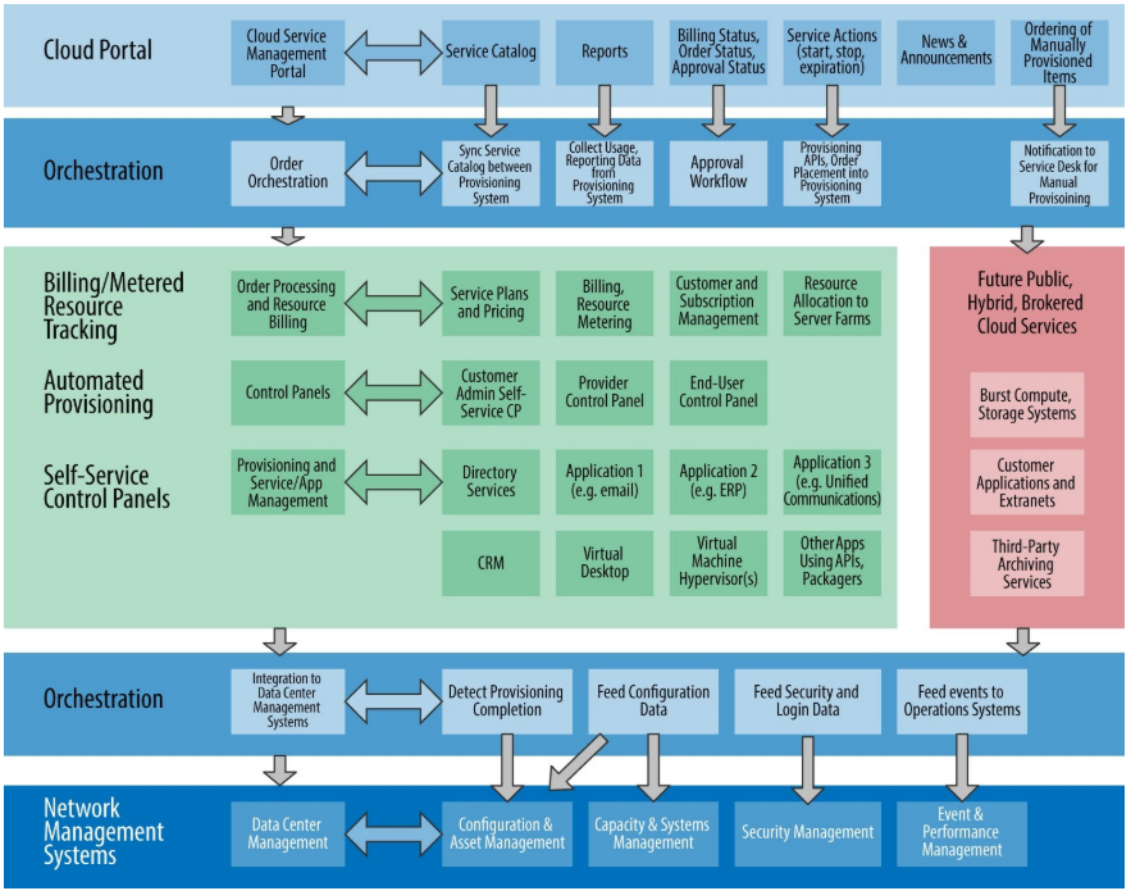


Figure 4.1: An example of Cloud Management Platform architecture [10]

4.1.1.2 The SDN Controller

In the Cloud infrastructure we consider, networking is handled by an SDN controller for dynamic network configuration. Since we already presented the SDN paradigm in Section 2.4, we will focus here on its interaction with the Cloud Management Platform to provide the network provisioning capabilities in a Cloud context.

Networking applications can be installed on top of the SDN controller in order to extend its features. This is realized using Northbound Application Programming Interfaces (APIs), which are offered by the controller. Their goal is to abstract the inner-workings of the network, allowing application developers to make changes to accommodate the needs of the application without having to understand exactly what that means for the network. The Northbound APIs are one of the critical and innovative part of the SDN paradigm since the added value of SDN depends on the cutting-edge features brought by the applications it can support and enable.

The Cloud Management Platform is integrated with the SDN controller using an application based on its northbound APIs. This application, called *Cloud-SDN* application in the rest of this thesis, is responsible for configuring the network based on the users' requests through the CMP

self-service portal, as presented on Figure 4.2.

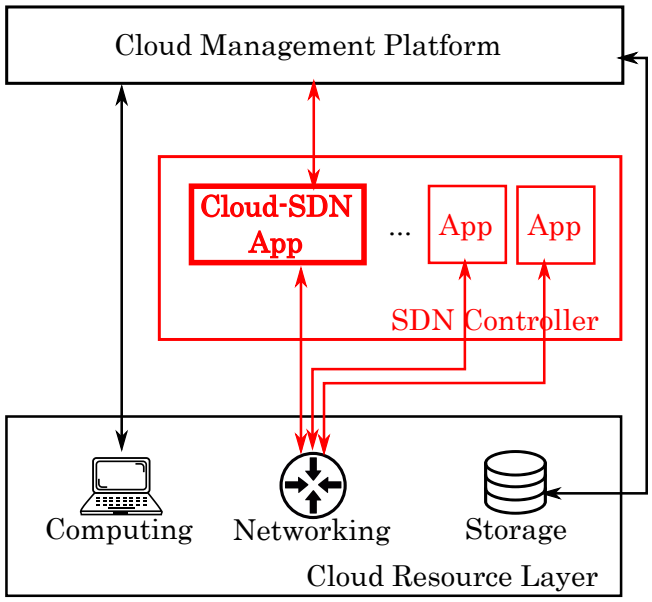


Figure 4.2: Interactions between the Cloud Management Platform Architecture, the SDN Controller and the Cloud Resources

As a result, the CMP network configuration view is contained in the SDN controller. While the CMP presents management interfaces to both the Cloud provider and the tenants, the SDN controller is exclusively managed by the Cloud provider. The administration of the virtual infrastructures is delegated to the CMP, which interacts with the SDN controller to provision the tenants' networks. Multi-tenancy allows each tenant to have its own virtual infrastructure made up of a set of VMs interconnected by virtual networks.

4.1.1.3 Challenges Introduced by Integrating the SDN controller and the Cloud Management Platform

Beyond the scalability and volatile nature of the tenants' infrastructure which are characteristics of the Cloud, and need to be considered in the solution, the combination of the CMP and the SDN controller poses an additional challenge. Indeed, Software-Defined Networking enables administrators to deploy applications in the SDN controller. Those applications can then, according to the programmed logic, reactively modify the flow rules on the virtual switches and directly affect the topology, without providing any feedback to the CMP. It results in inconsistent topology views between the Cloud Management System and the SDN Controller, since the CMP network configuration view included in the SDN controller is dynamically modified without the CMP being aware of these changes. If left unresolved, these discrepancies would result in the generation of incorrect attack graphs, considering the current state of the infrastructure. In order

to avoid the inaccuracy issue in this specific context, it is very important to rely on a view of the topology and connectivity that merges information from both the CMP and the SDN controller.

In summary, we presented in this section the Cloud technologies that can be leveraged in order to obtain a comprehension vision of the tenants topology and device connectivity in the context of the Cloud. We also presented the challenges incurred by the association of various technologies, namely the Cloud Management Platform and the SDN controller. In the following section, we will present the way the different data gathered from these different sources can be organized and processed for our goal.

4.1.2 Topology and Connectivity Graph Model

In a Cloud environment, computing, networking and storage resources are provided to the customers of the infrastructure, by sharing the capabilities of the physical devices. Each tenant can customize the interconnection of these resources to support its business needs. The isolation between the workloads of distinct tenants is enforced by virtualization technologies. Hypervisors instantiate several virtual machines according to the provisioning requests emitted by the tenants. These virtual machines are connected to virtual networks through virtual ports.

A Cloud Management Platform deployed in such environments allows to keep track of the Cloud administrator's and tenants' specific resources. By targeting specific data recorded within the CMP centralized database, we are able to reconstruct the topology of the Cloud network, i.e., the way in which its constituent parts are interrelated.

Indeed, from any Cloud Management Platform used for virtual infrastructures management, the same building blocks can be extracted to set up the topology: the *Hypervisor*, the *Virtual Machine*, the *Tenant*, the *Virtual Router*, the *Virtual Port*, the *Virtual Subnet*, the *Virtual Network*, the *Security Rule* and the *Security Group*,.

Virtual routers interconnect virtual machines belonging to distinct virtual subnets. Security groups represent a collection of security rules, that are applicable to virtual machines. A security rule is comparable to an *allow*-type firewall rule. Indeed, all types of communication are *denied* by default. A security rule can contain the IP range for source and destination hosts, port range, protocol (TCP, UDP or ICMP) and direction of the traffic authorized on the virtual machines. An additional option indicating the remote security group traffic allowed can also be provided. This means that only virtual machines belonging to the identified security group would be able to communicate with the designated machine, according to the rule configured by this security rule.

By relying on the topology retrieved using the CMP, we are able to reconstruct the intended connectivity in each tenants virtual infrastructures, based on the configured security groups and security rules. The purpose of this phase is to determine the network accesses existing in the infrastructure, in order to identify the machines which are effectively able to communicate with each other. This information is relevant for the attack graph generation, since communication

links potentially allow attackers to access and exploit vulnerabilities residing on remote devices connected to their current location.

We will use the definition of network access as proposed by Thibault Probst in [112].

In that work, Probst defines a network access as an authorized access to a service from a source to a destination, where the service is considered as the association of a protocol and a port. All these network accesses can be modeled using a reachability matrix which contains all the communication allowed within the infrastructure. This matrix is often specified in security policies to indicate the authorized network accesses.

We define the reachability matrix as a two-dimensional matrix where the first dimension represents the sources IP addresses and the second dimension the destination IP addresses. Our definition from a reachability matrix differs slightly from the one from Probst, whose first dimension is relative to the virtual machine as a whole and not specific IP addresses within that machine. He argues that from a user standpoint, we don't necessarily know in advance the IP address that will be used in a communication. However in our case, since all communications are denied by default, the users need to explicitly configure the security rules before any access can be authorized. In doing so, the source IP address used in a given communication can be extracted from the security rule. In addition, since we have direct access to the database maintained by the CMP, we can also determine the IP address affected to the virtual port of the corresponding virtual machine.

Eventually, in the attack graph, the distinction in IP addresses for a machine is not represented explicitly in the graph. The relevant information is the attack action used to reach a particular state on the destination machine, and not the IP address used in the communication link that enabled this attack action to happen, even if this information can be found. Hence, a single vertex is created to represent the state of the targeted destination machine, if attack actions using two distinct IP addresses of the target result in the same state on the destination machine.

We represent in Table 4.1 the amended definition of the reachability matrix.

			Destinations					
			VM_C		VM_D	VM_E		
			IP_{C1}	IP_{C2}	IP_{D1}	IP_{E1}	IP_{E2}	IP_{E3}
Sources	VM_A	IP_{A1}	$Service_y$					
		IP_{A2}				$Service_x$		
	VM_B	IP_{B1}						$Service_z$

Table 4.1: Reachability matrix

In order to build this reachability matrix, we need to focus on the configuration of all the components in the topology which have the potential to impact the connectivity. Based on the topology building blocks presented on Figure 4.4, we can introduce the topology- and connectivity-related predicates as presented in Table 4.2.

Let H, VM, SR, SG, T, VP, VR, S and N be the sets representing the collection of hypervisors (physical nodes), virtual machines, security rules, security groups, tenants, virtual ports, virtual routers, subnets and networks respectively.

In their expression, $h \in H$, $\{vm, vm_1, vm_2\} \in VM$, $t \in T$, $secr \in SR$, $secg \in SG$, $vp \in VP$, $vr \in VR$, $\{s, s_1, s_2\} \in S$, $n \in N$.

Common actions	Effects
instantiate(h,vm)	the hypervisor h is the host of the virtual machine vm
areColocated(vm_1, vm_2)	the virtual machines vm_1 and vm_2 are instantiated on the same hypervisor
own(t,X) where $X \in VM$ *or* $X \in S$ *or* $X \in SG$	the tenant t is the owner of the element X
isAttachedTo(vp,X) where $X \in VR$ *or* $X \in VM$	the element X is attached to the virtual port vp
belongTo(s,vp)	the virtual port vp belongs to the subnet s
isLinkedTo(s,n)	the network n is linked to the subnet s
contains(secg,secr)	the security group *secg* contains the security rule *secr*
isEnforcedOn(secg, vm)	the security group *secg* is enforced on the virtual machine
isInSubnet(s, X) where $X \in VM$ or X is an IP address	the element X is in the subnet s
hasIP(IP, X) where $X \in VM$ or $X \in H$	the element X has the IP address IP
areRouted(r, s_1, s_2)	the subnets s_1 and s_2 are connected via the router r
areConnected(x,y)	the communication is possible between x and y for at least one combination of protocol, addresses and ports

Table 4.2: Topology and connectivity predicates

areColocated(vm_1, vm_2) is a predicate expressing the fact that vm_1 and vm_2 belong to the same hypervisor. It is a *derived* predicate using the predicate *instantiate(h, vm)* which expresses the fact that the virtual machine vm is located on the hypervisor h. The resulting expression is the following:

$$areColocated(vm_1, vm_2) \text{ :- } instantiate(h, vm_1) \wedge instantiate(h, vm_2)$$

isInSubnet, *areRouted* and *areConnected* are predicates deduced from the topology and connectivity predicates. Two virtual machines X and Y *areConnected* if they are either on the same subnet or on subnets linked by routers, and their security rules allow communication for at least one combination of addresses, ports and protocol. Subnet information is provided by *belongTo* and *isLinkedTo*, while reasoning on security rule applicability is permitted by the

predicates *isEnforcedOn* and *contains*. The content of the rules themselves is then interpreted in order to derive connectivity among virtual machines as illustrated on Figure 4.3.

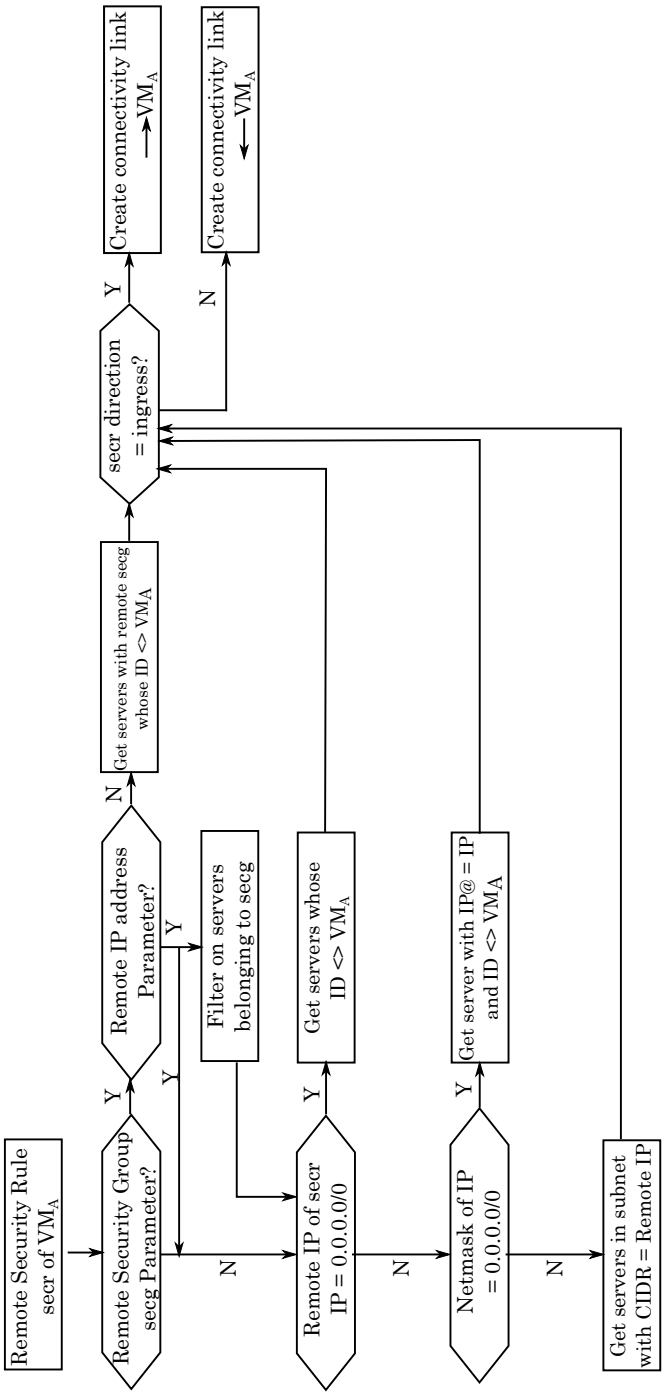


Figure 4.3: Interpretation of the Security Rules

We present in the following, an instance of how this can be deduced from the predicates introduced. In these expressions, *isAttachedTo(vp, vm)* means that the virtual machine *vm* is

attached to the virtual port vp, *isAttachedTo(vp, r)* means that the virtual router r is attached to the virtual port vp and *isAttachedTo(vp, s)* means that the virtual subnet s is attached to the virtual port vp.

$$isInSubnet(s, vm) \text{ :- } isAttachedTo(vp_{1vm}, vm) \wedge isAttachedTo(vp_{1vm}, s)$$

$$areRouted(r, s_1, s_2) \text{ :- } isAttachedTo(vp_{1r}, r) \wedge isAttachedTo(vp_{1r}, s_1) \wedge isAttachedTo(vp_{2r}, r) \wedge isAttachedTo(vp_{2r}, s_2)$$

Considering a security rule defined as:
$secrule(id_{secr}, IP_{src}, IP_{dest}, protocol, port_{src}, port_{dest}, direction_{egress}, id_{remoteSecg})$, we can deduce the connectivity between vm_1 and vm_2 using one of the two following expressions.

In these expressions, *contains(secg,*id_{secr}*)* means that the security group $secg$ contains the security rule $secr$, *isEnforcedOn(secg, vm)* means that the security group $secg$ is enforced on the virtual machine vm, *areRouted(r,* s_1*,* s_2*)* means that the virtual subnets s_1 and s_2 are routed using the virtual router r, *isInSubnet(s, vm)* means that the virtual machine vm belongs to the virtual subnet s, *isInSubnet(s,* IP_{dst}*)* means that the IP address IP_{dst} belongs to the virtual subnet s, and finally *hasIP(*IP_{dst}*, vm)* means that the virtual machine vm has the IP address IP_{dst}.

$$areConnected(vm_1, vm_2) \text{ :- } contains(secg, id_{secr}) \wedge isEnforcedOn(secg, vm_2) \wedge areRouted(r, s_1, s_2) \wedge isInSubnet(s_1, vm_1) \wedge hasIP(IP_{src}, vm_1) \wedge isInSubnet(s_2, vm_2) \wedge hasIP(IP_{dst}, vm_2) \wedge isInSubnet(s_1, IP_{src}) \wedge isInSubnet(s_2, IP_{dst})$$

$$areConnected(vm_1, vm_2) \text{ :- } contains(secg, id_{secr}) \wedge isEnforcedOn(secg, vm_2) \wedge isInSubnet(s, vm_1) \wedge hasIP(IP_{src}, vm_1) \wedge isInSubnet(s, vm_2) \wedge hasIP(IP_{dst}, vm_2) \wedge isInSubnet(s_1, IP_{src}) \wedge isInSubnet(s_2, IP_{dst})$$

We can merge the topology and connectivity models and represent them as a graph, by assimilating the building blocks to typed nodes and the predicates to typed relationships as represented on Figure 4.4. For simplicity, we only represent the most important ones.

4.1.2.1 Addressing the Connectivity Discrepancies caused by the Integration with an SDN Controller

As introduced in Section 4.1.1.3, provider's applications deployed on the SDN controller are able to modify the configuration of virtual switches and implement flow rules, independently from the configuration requested using the CMP.

Flow rules are equivalent to routing rules authorizing or forbidding connections between virtual machines based on traffic patterns and SDN applications' logic. From a network connectivity standpoint, they are the concrete realization of security policies. They are ordered according to

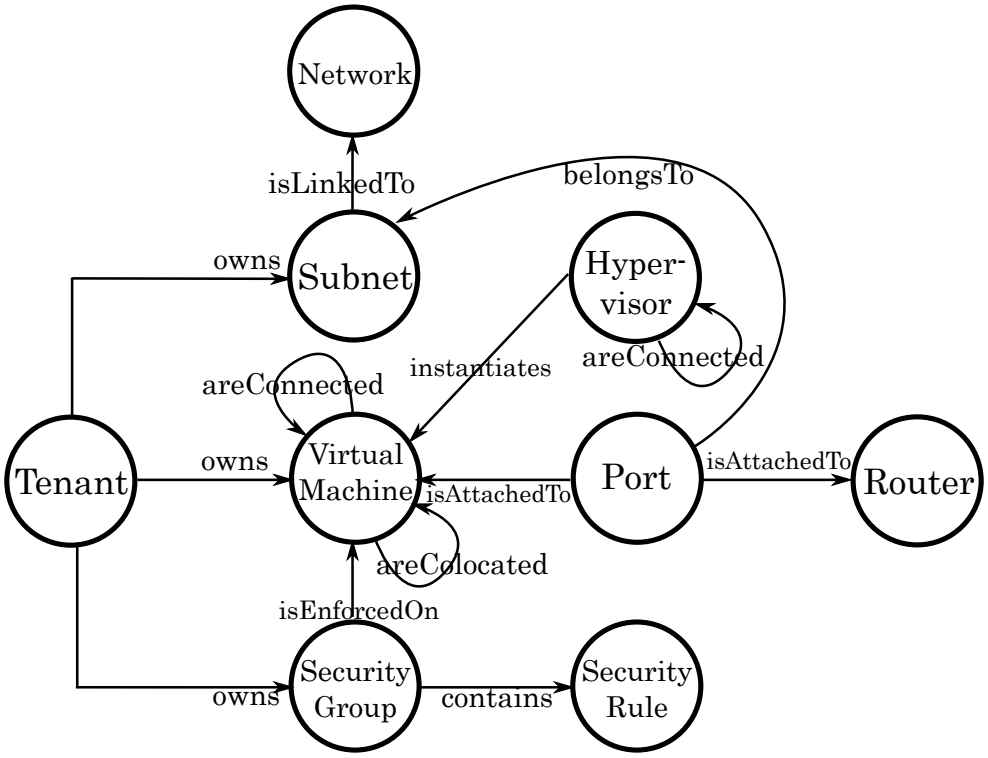


Figure 4.4: Graph Model

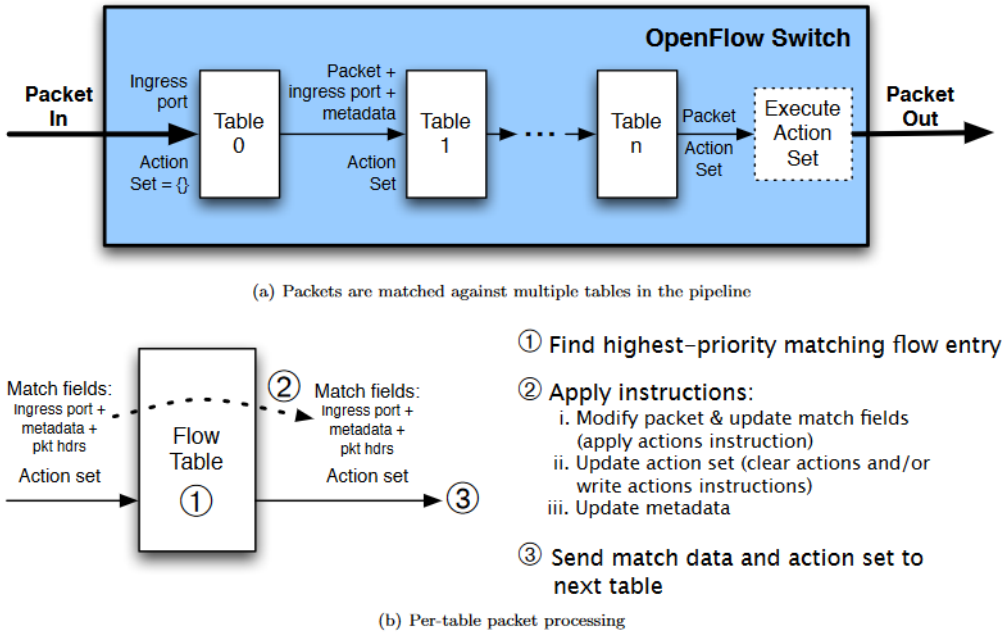


Figure 4.5: Packet flow through the processing pipeline [11]

arbitrary priorities given by the developer of the applications running on the SDN controller, resulting in a hierarchical ordering of the flow rules installed on the virtual switches. These

flow rules are organized in several flow tables. Once a packet is received, the switch starts by performing a table lookup in the first flow table, and based on pipeline processing, may perform table lookups in other flow tables, as illustrated on Figure 4.6. Figure 4.6 represents the pipeline processing of a flow rule as specified by the Open Networking Foundation (ONF) in the version 1.3 of the Openflow protocol [11], which is a protocol used by the SDN controller to communicate with the switches.

Starting from the first table, packets are matched against flow rules in decreasing order of priority and only the flow rule instructions corresponding to the first match are applied. The matching is typically performed on the packet type, the protocol, the IP source and destination addresses, the ports, the metadata fields, and other pipeline fields. Examples of actions can be to *drop* the packet, to *forward* the packet to an egress port, or to *jump to table* in order to continue the pipeline processing. Figure 4.6 represents instructions that are executed when a packet matches an entry in the flow table. These instructions result in changes to the packet, action set and/or pipeline processing [12].

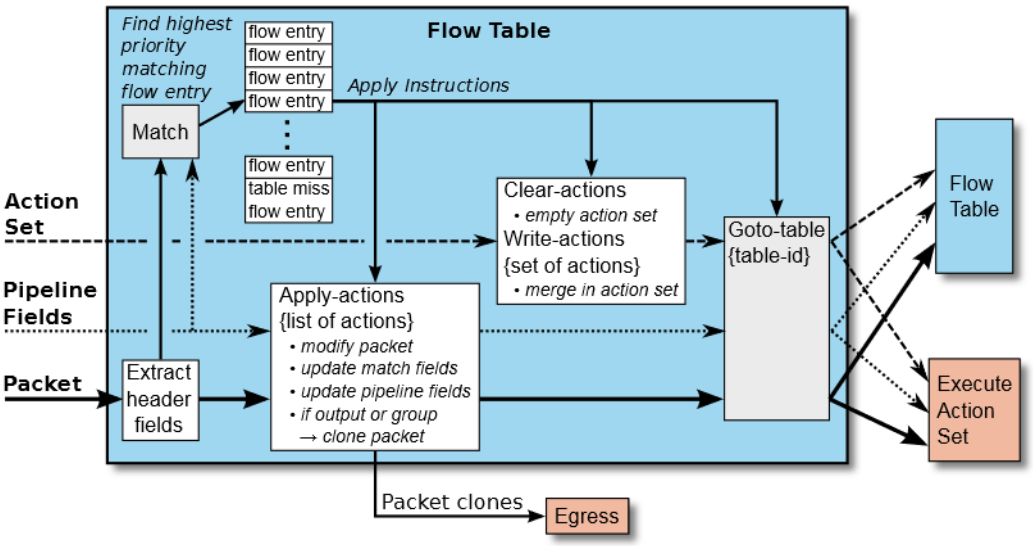


Figure 4.6: Matching and instruction execution in a flow table [12]

In our context, the Cloud-SDN application is interfaced with the Cloud Management Platform, and implements the necessary flow rules with an arbitrarily defined priority. A similar behavior is observed for all the other applications installed on the SDN controller. Only the flow rules installed with a priority higher than the ones installed via the Cloud-SDN application are unknown from the CMP and impact the resulting connectivity. Indeed, in case of positive match with an incoming packet, the actions requested in those rules will superseed lower priorities ones (in particular, the ones from the CMP).

In order to track the occurrence of the changes not requested by the Cloud-SDN application, we implement an additional application on the SDN controller, whose role is to listen to events

generated when a flow rule modification occurs. Indeed, every flow rule adjustment or creation triggered by an application generates a notification in the SDN controller. This concurrent SDN application monitors and intercepts the messages generated, interprets their contents and deduces the way in which the connectivity, as viewed from the CMP, should be amended.

4.1.3 Topology and connectivity graph construction

We favor a passive approach for the reconstruction of the connectivity based on the observed configurations retrieved from the Cloud Management Platform Database. This allows to incur the least amount of disruption on the customers virtual infrastructures, while gathering relevant data for the attack graph construction.

In this section, we introduce our architecture, and describe the different phases of the algorithm. Figure 4.7 illustrates our architecture: a Cloud infrastructure administered by a CMP and a SDN controller whose data are processed by the topology and connectivity builder. The

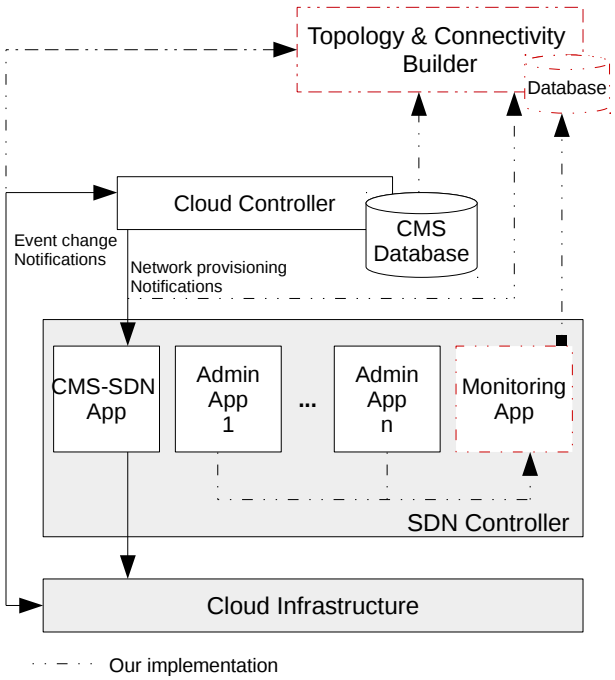


Figure 4.7: Architecture

SDN controller comprises an SDN monitoring application, listening to flow rule events generated by other applications. It is one benefit of the SDN paradigm to enable the administrator to deploy additional applications on their controllers in production. With this SDN monitoring application, rule characteristics are then stored in the SDN rule database. In parallel, the topology and connectivity builder runs in a dedicated server. It first creates a static topology and connectivity by processing the CMP database and stores the obtained representation into its own graph

database. Secondly, it listens to CMP-generated events to update its view. The events tracked are the creation, the deletion, the update and the status change of virtual elements.

As an example, the impact on the topology and connectivity, considering changes occurring on virtual machines are presented in Table 4.3. In this table, arrows represent the creation of an edge (predicate) whose type is given by the label written above. A crossed arrow represents the deletion of the predicate.

4.1.3.1 Static Phase

During the static phase, the aim is to establish a baseline topology and connectivity. We begin with the static topology, built with data from the Cloud Management Database.

Common actions	Effects
Create	Node creation: vm Node attribute modification: vm.state=active vm $\xleftarrow{owns}$ tenant vm $\xleftarrow{isEnforcedOn}$ secgroup vm $\xleftarrow{instantiate}$ hypervisor Generation of a port create event Node creation: port subnet $\xleftarrow{belongsTo}$ port port $\xleftarrow{isAttachedTo}$ vm vm $\xleftarrow{areConnected}$ vm_x, where vm_x is an existing machine able to communicate with vm
Delete	vm $\not\xleftarrow{owns}$ tenant vm $\not\xleftarrow{isEnforcedOn}$ secgroup vm $\not\xleftarrow{instantiates}$ hypervisor Generation of a port delete event subnet $\not\xleftarrow{belongsTo}$ port port $\not\xleftarrow{isAttachedTo}$ vm vm $\not\xleftarrow{areConnected}$ vm_x, where vm_x is an existing machine able to communicate with vm Node deletion: port Node deletion: vm
Deactivate: Pause, Shutdown, Shelve, etc.	Node attribute modification: vm.state=inactive
Activate: Unpause, Start, Unshelve, etc.	Node attribute modification: vm.state=active
Update(params)	vm.params=params

Table 4.3: Example of elementary actions performed on virtual machines in Cloud environments and their effects on the topology

The network topology is built first, it is comprised of the interactions between virtual machines, virtual ports, subnets, networks, virtual routers, security groups, security rules and tenants.

In order to optimize its generation, we leverage the multi-tenant nature of the Cloud: the network topology of each tenant is built independently by concurrent threads, allowing to parallelize the task. This method allows to speed up the construction, especially when a lot of tenants are considered. Once the network topology of each tenant is obtained, the relationship is established with the Cloud provider's physical infrastructure by creating an *instantiates* predicate between the tenants' virtual machines and their corresponding hypervisors.

Algorithm 1: Building static connectivity using the CMS

```
   Data: tenants_list, global_topology
   Result: CMS Connectivity in virtual infrastructure
 1 foreach tenant in tenants_list do
 2     routers_list = getRoutersList(tenant);
 3     standaloneSubnets_list = getSubnetsWithNoRouter(routers_list) ;
 4     foreach router in router_list do
 5         virtualMachines_list = getVmsConnectedViaRouter(router);
 6         foreach machine in virtualMachines_list do
 7             securityRules_list = getSecurityRules(vm);
 8             foreach sr in securityRules_list do
 9                 relatedVms_list = identifyRelatedVms(virtualMachines_list, sr.protocol,
                    sr.ip_dst, sr.ip_src, sr.ip_prefix, sr.sec_group_id, sr.direction);
10                 createAreConnected(vm, relatedVms, global_topology);
11     foreach subnet in standaloneSubnets_list do
12         virtualMachines_list = getVmsSubnet(subnet) /* repeat line 6 to 10     */
```

After generating the static topology (global_topology), static connectivity is obtained according to Algorithm 1. It is built by identifying groups of machines able to communicate, due to their interconnection with routers or their belonging to a same subnet (lines 2-3). Each machine cluster is then processed to determine the effective possible communication as stated by security rules contained in the security groups they depend on (lines 8-10). This phase generates *areConnected* links as viewed by the CMS (line 10). Additionally, since flow rules provisioned by SDN applications are hierarchical, we identify the ones with a higher priority than the rules provisioned by the CMS application in the SDN controller, by querying the Flow Rules registry in the SDN controller, as shown on lines 1-4 in Algorithm 2. Indeed, these are the rules yielding discrepancies between the views from the CMS and the SDN controller. We develop a Monitoring Application installed on the SDN controller, responsible for identifying and registering these rules in a separate SDN rule database. Parameters contained in each rule allows to match the related CMS *areConnected* links and modify them according to the SDN view of the connectivity as shown on lines 6-8 in Algorithm 2.

Algorithm 2: Updating the static connectivity using the SDN Controller

```
Data: CMSAppsIds_list, virtualSwitches_list, connectivityGraphDB
Result: Merged Connectivity using CMS and SDN controller
1 CMSFlowrulePriorities_set = getFlowrulePrioritiesByAppsIds(CMSAppsIds_list);
2 foreach switch in virtualSwitches_list do
3     installedRules_list = getSwitchFlowrules(switch);
4     superseedingRules_list = getSuperseedingRules(installedRules_list, CMSAppsIds_list,
        CMSFlowrulePriorities_set);
5     foreach rule in superseedingRules_list do
6         ruleCharacteristics = getRulesCharacteristics(rule);
7         connectivityLinks = findMatchingPatterns(connectivityGraphDB,
            ruleCharacteristics);
8         updateConnectivityGraphDB(connectivityGraphDB, connectivityLinks,
            getRuleAction(rule));
```

4.1.3.2 Dynamic phase

In the Cloud, infrastructure elements are subject to faster modifications than in conventional architectures. Rebuilding the attack graph upon each modification is an approach used in traditional environments, which evolve slowly. However, the computational cost in a highly dynamic environment such as the Cloud would incur a toll on performance. We argue that, by monitoring modification events occurring in the infrastructure, updates can be computed faster than a complete graph regeneration. In Figure 4.8, we present a UML state diagram for the components in the virtualized infrastructure and the reachability links created by the builder in [113]. In that diagram, each transition is triggered by a change event generated either by the Cloud Management Platform or the topology and connectivity builder. For simplification, we only considered in the figure the events and the resulting states that are relevant to our attack graph update process. Events with a white background are the ones intercepted from the CMP, while events with a grey background are generated by the topology and connectivity builder. The latter can be considered as the aggregated processing of elementary events generated by the CMP, in order to build a comprehensive and self-contained notification message exploitable by the attack graph generation algorithm. Each connectivity update occurring in the infrastructure now generates a connectivity modification event that will be consumed by our attack graph generation algorithm. Every connectivity link created or deleted between two devices triggers a notification message containing the details of the link: source, destination, protocol and port. In addition to network connectivity, we consider virtual machine co-location, as it can lead to a communication channel due to a vulnerability on the hosting hypervisor.

This UML diagram shows that the creation or deletion of connectivity links might be caused by the creation of a virtual machine or a change in its state, depending on the firewall rules applied to it. Indeed, states causing the machine to go offline such as pausing, shutting down or

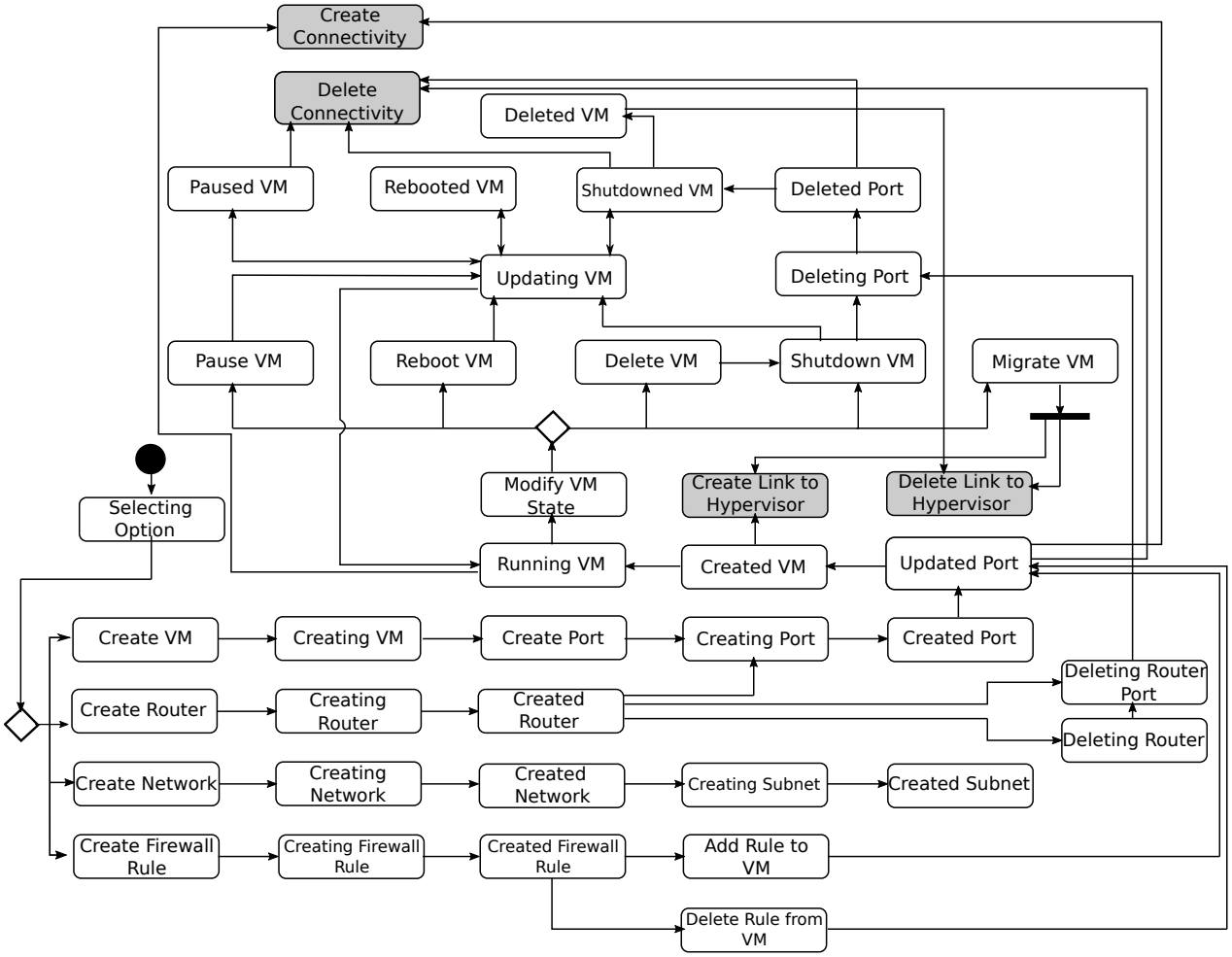


Figure 4.8: Events generated in the infrastructure by the Cloud Management Platform

deleting, could lead to the deletion of connectivity links. Returning to a running state or creating a new virtual machine could create connectivity links, while a reboot is without effect on the connectivity, as it represents a transient state. Since the migration of a virtual machine from one hypervisor to another only impacts the device location, it is also without impact on virtual machines connectivity. However, the original co-location links relative to this virtual machine, will be first deleted, then recreated to reflect its potential interactions with its new neighboring virtual machines on the destination hypervisor. By adding or removing a firewall rule to or from a virtual machine, we trigger an update of its virtual port, leading to the creation or deletion of a connectivity link according to the rule considered. Additionally, connecting or disconnecting a router to or from a subnet also leads to a creation or deletion of connectivity links due to a virtual port creation or deletion. On the other hand, the creation, resp. deletion of a virtual machine generates a message for an instantiation, resp. a deletion link to the hosting hypervisor. The virtual machine migration generates both types of links as it moves the virtual machine from one hypervisor to another. These events are useful in identifying the co-located virtual machines

on the same hypervisor, susceptible to be used in an attack step involving a vulnerability on the hypervisor.

During the dynamic phase, an event listener intercepts topology-related notifications generated by the Cloud Management Platform to store them in a queue. Events are then processed by queue consumers to update the topology and connectivity graph with the changes induced by those notifications as defined in Table 4.3. The events are processed to modify the topology, as well as the connectivity reported in the graph. The SDN applications are also continuously monitored, in order to register any impactful modification caused by a rule creation, update or removal. The information gathered by this monitoring application allows to update an SDN rules database (see Figure 4.7) and modify the *areConnected* links accordingly.

CHAPTER 5

ATTACK GRAPH MODEL

Attack graphs are a modeling tool used in the assessment of the security of an information system. Based on this representation, network administrators are able to understand interactions between vulnerabilities existing on distinct devices and leveraged by attackers to compromise the infrastructure. Beyond the visualization aspect, attack graphs can be used to analyze and quantify the security risks faced by the machines identified as assets in the networks. Such quantification can result in a prioritization of the most critical attack paths encountered and a proposal of the most suitable countermeasures to prevent them, countermeasures that can be coupled with automated deployment techniques.

5.1 Objectives and Requirements of the Proposed Model

The end goal of our attack graph model is to be used for the proactive and automated generation of a set of applicable countermeasures in adequacy with security concerns expressed by the administrators. Indeed, several security questions can be formulated regarding the state of an information system, each resulting in analyzing the attack graph under different lenses: for instance, identifying the most critical attack path globally or for a specific resource, or identifying the minimum number of paths to delete in order to prevent all attack paths from reaching a given resource or set of resources. These questions are often expressible using typical graph problems such as the shortest path or minimum cut problems.

From this observation, follows the first requirement of our attack graph model: its compatibility with the use of algorithms coming from graph theory, in order to optimally address the preoccupations formulated by the administrators. This requirement imposes some constraints on the final representation used for the attack graph and prevents us from directly using models such

as MulVAL [56] and TVA [7]. Indeed, these representations consider conjunctive and disjunctive paths in the attack graph, by using either explicit or implicit logical nodes, namely AND/OR nodes. In our context, the AND nodes are the most problematic, since applying graph algorithms on these types of graphs would require an additional processing, to be able to reconstruct an ordered sequence out of the conjunctive paths. Indeed, algorithms in graph theory are designed for graphs formally defined as follows [114, 115]:

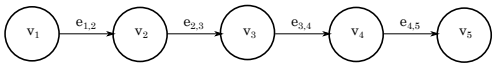
Definition 3. ***Definition of an undirected graph:*** *An undirected graph G is an ordered pair of sets (V,E) where V is a finite set called the set of vertices and E is a set of unordered 2-element subsets of V, called the set of edges. If $e_{u,v}$ is an edge and u and v are vertices such that $e_{u,v} = \{u,v\}$, then $e_{u,v}$ is said to connect u and v, and the vertices u and v are called the endpoints of $e_{u,v}$. $\{u,v\}$ and $\{v,u\}$ represent the same edge in the undirected graph G.*

Definition 4. ***Definition of a directed graph or digraph:*** *A directed graph G is an ordered pair of sets (V,E) where V is a finite set called the set of vertices and E is a set of ordered pairs of V elements, called the set of edges. If $e_{u,v}$ is an edge and u and v are vertices such that $e_{u,v} = (u,v)$, then $e_{u,v}$ is said to connect u and v, the vertex v is called the head of $e_{u,v}$, while u is called the tail of $e_{u,v}$. (u,v) and (v,u) represent two distinct edges in the digraph G.*

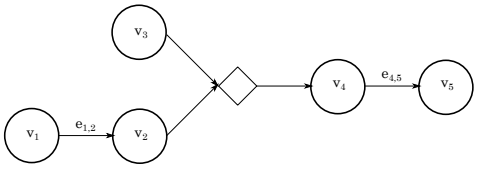
Definition 5. ***Definition of a Walk in a graph G:*** *A walk in a graph $G = (V,E)$ is a sequence of the form $(v_1, e_{1,2}, v_2, e_{2,3}, v_3, ..., v_k, e_{k,k+1}, v_{k+1})$ where $k \geq 0$ such that $v_i \in E$ and $v_{i+1} \in E$ for $1 \leq i \leq k$, with k being the length of the walk.*

Definition 6. ***Definition of a Path in a graph G:*** *A path in a graph $G = (V,E)$ is a walk of length $k \geq 0$ in which the vertices $v_1, ..., v_{k+1}$ are all distinct.*

Based on these formal definitions, attack graphs suitable for the use of graph theory algorithms should be assimilated to directed graphs in which paths represent a continuous sequence of vertices and edges, and are not merged via the use of logical AND nodes. We illustrate a path in a directed graph according to its definition in graph theory on Figure 5.1a. It differs from a similar path considered in attack graph models presented previously, and presented on Figure 5.1b, by having an explicit and unbroken sequence of vertices.



(a) Representation of a path in a directed graph



(b) Representation of a path in attack graphs models with logical nodes

Figure 5.1: Comparing directed graph and logical attack graph paths representation

On the other hand, since such algorithms have been thoroughly optimized in the literature, we can benefit from the improvements realized on their performances to provide a rapid analysis of the attack graph. This is a necessity in the Cloud context, in which we consider a large number of physical and virtual machines.

An additional objective of our attack graph model is to allow the administrators to rapidly identify the machines involved in an attack scenario. According to the Oracle and KPMG Cloud Threat Report [116] released in 2018, only 35% of the respondents declare their company committed to security automation, when automation is essential in efficiently protecting Cloud environments. One of the barriers slowing down the adoption of automation is the fear of losing control, which is exacerbated by opaque decision making processes. We argue that understanding the graph representation gives a clearer picture of the scope of the countermeasures proposed, and will inspire an increased confidence in the efficiency of options automatically chosen for risk reduction. From the state of the art presented, we can identify models with non-uniform semantics, in which nodes can represent either a vulnerability, a location or an attacker action. That makes the model difficult to understand, especially considering a larger scale environment such as the Cloud. The interactions represented in the attack graph should be viewed through the lenses of the most understandable unit, namely, the physical or virtual host. Indeed, as presented in the state of the art, an attack graph user may need to visualize the attack paths generated to verify the mitigation proposals, or understand the anatomy of the attacks for forensics reasons. Amman and al. [5] host-centric proposal, specifically tailored for human operators, represents a nice approach to that end.

Besides, the granularity of attack graphs representation can be so fine that it explicitly represent too much details, such as the preparatory steps needed for a vulnerability exploit: fingerprinting, port scanning, file listing, etc. While valuable to understand the anatomy of an attack scenario or useful when the graph is used for reactive countermeasures, this level of detail can only be understood in a small size environment, as such approaches become rapidly intractable in the context of the Cloud. As a consequence, we consider that the purpose of our model is not to represent these reconnaissance steps, this model considers that the attacker already achieved these steps and has a complete understanding of the infrastructure under attack. They then leverage their knowledge to chain the actions leading them to assets compromise. Hence, the vulnerabilities and actions chosen for the construction of attack paths in our attack graph model refer directly to the attacker's progression, excluding reconnaissance steps, for more readability.

5.2 Brief Presentation of the Model

As presented in Section 2.2.1.2 (page 19), researchers explored a host-centric view of the attack graph which immediately conveys more meaning as it has directly the granularity that a security

administrator can rapidly understand. A host-centric attack graph is by essence a propagation model where the attacker moves from machine to machine. We intend to use this host-centric model for our attack graph representation. In host-centric models, a node represents a host, and an edge is labeled with the vulnerability used to reach the following host. We extend this representation in order to include more information relevant to our attack graph model in the nodes, while keeping the focus on the host-centric view.

5.2.1 Similarities with Existing Methods

Our model being an extension of existing proposals, we introduce here the resemblances that can be found first in the inputs and then in the modeling.

At the basis of any attack graph construction model is the network connectivity, namely which machines can communicate with each other depending on the protocols and the ports.

In addition to the network configuration, trust relationships can also be modeled to determine the remote authentications permitted between infrastructure devices. Trust relationships represent a constraint on the connectivity links. In the absence of a trust relationship, the connectivity is always satisfied. However, when a trust relationship is provided for a given link, the conditions must be met before the connectivity can be realized. This is similar for instance to the need of providing a password or cryptographic key as a specific user, before being able to access a machine using the SSH protocol.

The vulnerability inventory of each network host is the next source of data required for an attack graph construction. The vulnerability modeling is generally based on a description of its pre-conditions and post-conditions. The pre-conditions specify the conditions that must be met for an attack to succeed, while the post-conditions indicate the effects of the attack upon successful realization. The next step is then to determine vulnerability chains based on the logical links existing between them, i.e., post-conditions of a vulnerability A found in the pre-conditions of a vulnerability B.

An algorithm then takes as input the network connectivity, trust relationships, vulnerability inventory and vulnerabilities modeling to generate the attack graph containing all the paths an attacker might use to compromise the network.

5.2.2 Specificities of our Attack Graph Model

The first specificity of our attack graph model is relative to the inputs considered for its construction. In traditional environments, the only communication channel existing between the hosts in the infrastructure is the network connectivity. This connectivity is more often than not a pre-condition to the remote exploit of a given vulnerability, as its existence conditions the vulnerability reachability and hence the existence of the related attack path. An additional communication channel can however be considered in the Cloud. Indeed, when the isolation provided to virtual machines by the hypervisor is breached due to the existence of a vulnerability,

it might be possible for virtual machines to access either neighboring machines hosted on the same hypervisor or the hypervisor itself, depending on the vulnerabilities considered as presented in Chapter 3. As a result, we consider an extended range of vulnerabilities in our context. These vulnerabilities are analyzed in combination with the co-location property of the virtual machines, to determine whether it is possible for a malicious virtual machine to compromise its neighbors or the hosting hypervisor.

The second particularity lies in the modeling adopted for the attack graph. The study performed on the virtualization vulnerabilities allowed to isolate common conditions found in the pre-condition and the post-condition sets, and relative to the source and destination hosts involved in the vulnerability: these conditions are the attacker location and its permission level at the given location. Beyond the specific case of virtualization vulnerabilities, this observation can be verified on a large number of vulnerabilities. To exploit them successfully, an attacker needs not only to be on a specific machine, i.e, at a specific location, but he also needs to have some type of privilege on this machine, be it the one of an anonymous user. We use this knowledge on the vulnerability modeling to define the vertices considered in our attack graph. Instead of considering an attack chain as a succession of vertices corresponding to vulnerabilities (like exploit-centric approaches in Section 2.2.1.3), we transform a vertex associated with a vulnerability in the attack chain into two vertices and an edge: the vertices represent the source or destination host, associated with their corresponding permission levels, while the edge represents the action used to pass from a vertex to the next. This action can either be an exploit or a legitimate action.

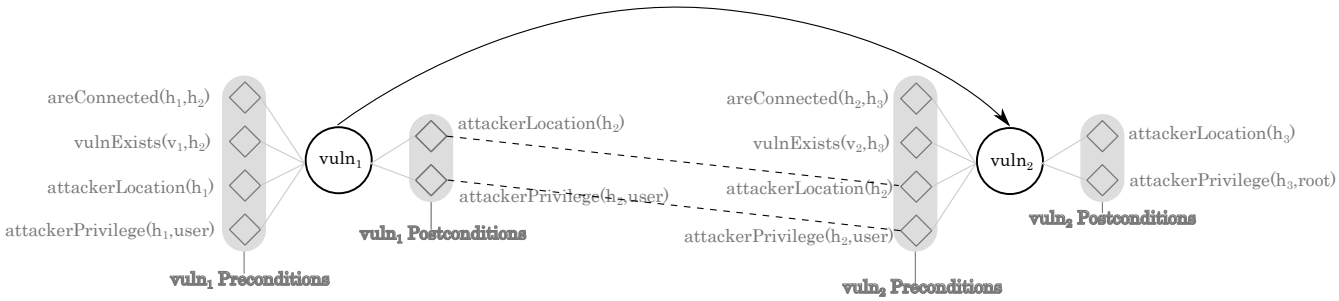


Figure 5.2: Exploit-centric representation of a transition path two vulnerabilities

In practice, an attacker can have access to several compromised machines at the same time. However, since we represent (non-concurrent) attack paths, the attacker location considered in the graph at a given time is a location belonging to one or several attack paths.

We show a transition between two vulnerabilities $vuln_1$ and $vuln_2$ on Figure 5.2. We explicitly represent, with the grey color, the preconditions and postconditions associated with each one of them for illustrative purposes. Diamonds with incoming grey links to a vulnerability represent its preconditions, while the ones with outgoing grey links are its postconditions. Dashed lines are drawn between similar postconditions of the first vulnerability and preconditions of the second

vulnerability. The grey elements can normally be removed from the resulting attack graph to simplify the visual and leave only the effective attack path which is a connection between $vuln_1$ and $vuln_2$, here represented by an arrow. We represent the exact same scenario as Figure 5.2 on Figure 5.3 which is an illustration of our approach and has the immediate benefit to focus on the hosts involved in a sequence of vulnerability exploits. Our approach is not fully host-centric,

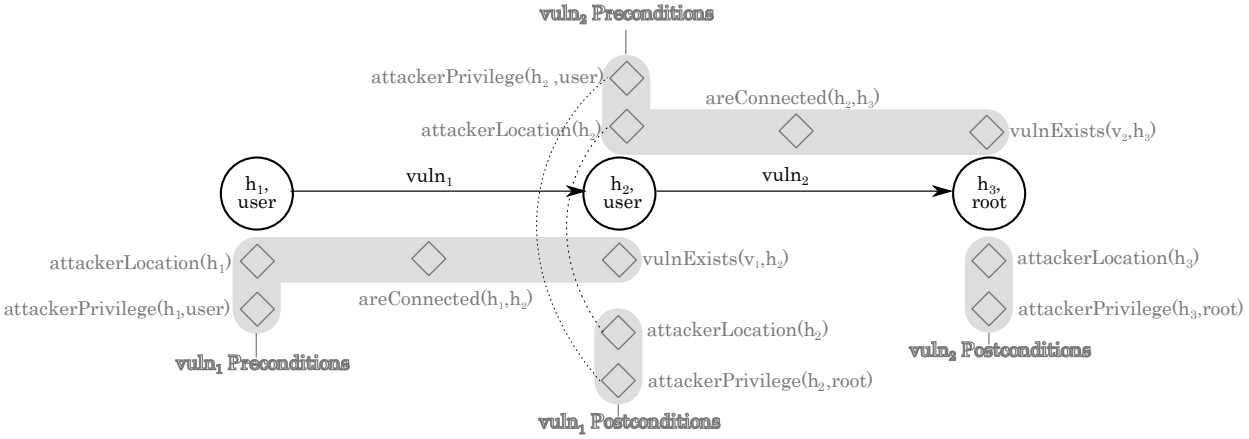


Figure 5.3: Host-centric representation of a path between two hosts

since in addition to the location, the nodes are associated with the current attacker's privilege level on the host. As a result, there are as many nodes for a given host, as there are possible permissions reachable to the attacker by leveraging exploits.

To go further, we represent on Figure 5.4 the exact equivalent of the scenario described by Ou *et al.* in [117], using our modeling approach. The resulting attack graph is presented on Figure 5.5. The scenario is described as follows:

> In a small enterprise network shown, there are three subnets mediated by an external and an internal firewall. The web server is in the DMZ subnet and is directly accessible from the Internet through the external firewall. The database server is located in the Internal subnet and holds sensitive information. It is only accessible from the web server and the User subnet. The User subnet contains the user workstations used by the employees of the company. The firewalls allow all out-bound traffic from the User subnet. The web server contains the vulnerability CVE-2006-37471 in the Apache HTTP service which can result in a remote attacker possibly executing arbitrary code on the machine. The database server contains the vulnerability CVE-2009-2446 in the MySQL database service which could allow administrator access. The user workstations contain the vulnerability CVE-2009-1918 in the Internet Explorer. If a user accesses malicious content using the vulnerable IE browser the machine may be compromised.

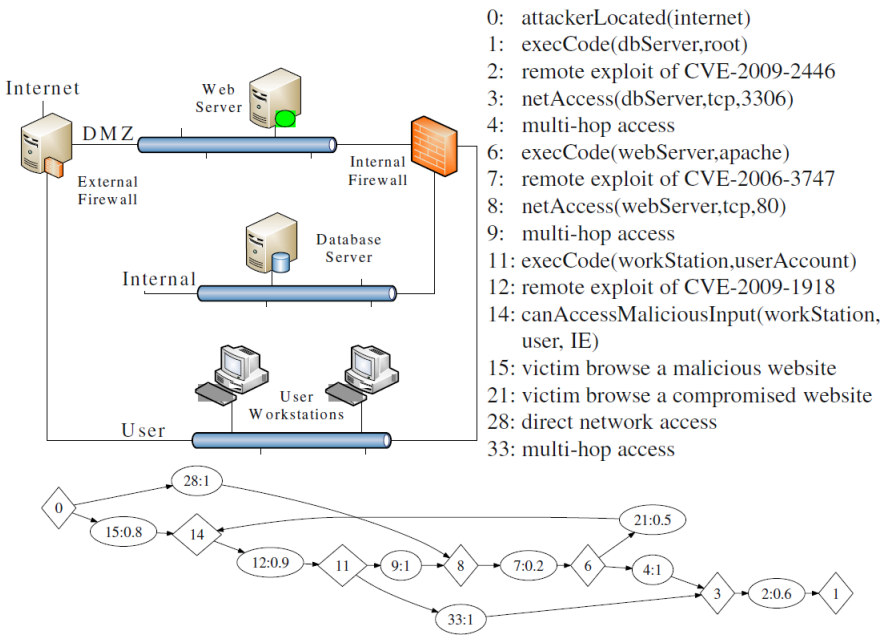


Figure 5.4: MulVAL example scenario

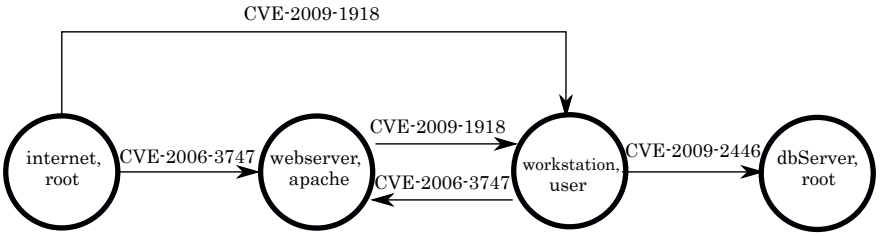


Figure 5.5: MulVAL example scenario using our host-centric approach

In addition to these common conditions related to the attacker location and its privilege level on a given host, exploiting some vulnerabilities can necessitate additional requirements, dependent on the path previously taken by the attacker or on previously performed actions. Our approach towards a host-centric attack graph representation is then based on a separation of the vulnerability pre-conditions and post-conditions into 2 clusters each: a cluster A of conditions containing solely the attacker's location and its permission level at that location, and a cluster B containing all the other conditions.

The preconditions located in cluster B are required so an attacker can perform an exploit. To account for them in our model, we extend our representation to label the edges with the conditions contained in the cluster for each vulnerability considered: the constraints are transferred to the corresponding vulnerability edge and must be satisfied before the transition can be fired.

Additional nodes, fulfilling the conditions in cluster B, are created if they are relative to a location where the vulnerabilities possess these conditions in the post-conditions of their cluster B. Each node is associated with a single condition in cluster B, since a particular vulnerability

might only fulfill one of the conditions. Several nodes potentially exist for a given condition in cluster B, since the constraint may be realized at various location in the network. The nodes have the same semantics as before, only with an additional field expressing the constraint being satisfied.

With this approach, we observe on one hand a constrained edge representing a transition that can only be fired when the constraint is satisfied, and on the other hand, a node representing the fulfillment of that constraint. It emerges from this observation that there is a need to link the node satisfying the constraint to the edge that requires it. We present in Section 5.3.4 the approach used to that end.

For illustrative purposes, we slightly complexify the MulVAL example. Let us consider hypothetically that CVE-2009-2446 requires an additional condition c_1 to be successful and an additional host h_1 is accessible from the internet with a vulnerability v_1 having the condition c_1 in the post-conditions of its cluster B. The graph illustrated on Figure 5.6 can then be obtained: to satisfy condition c_1, and compromise *dbServer*, the attacker should first reach host h_1. We can

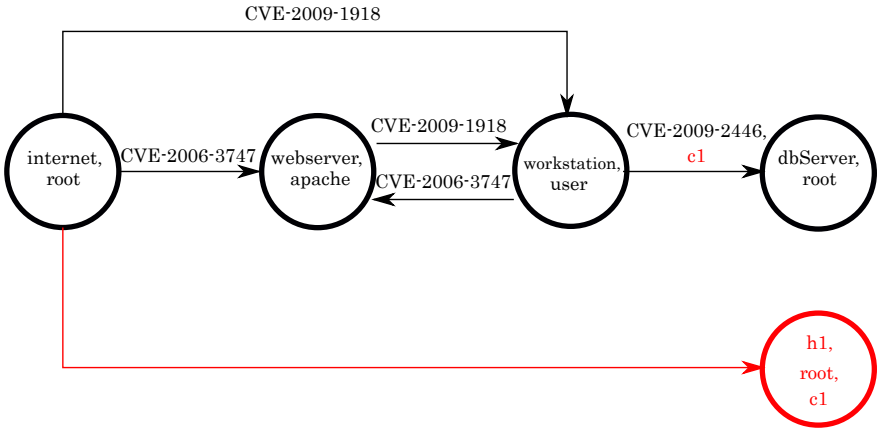


Figure 5.6: MulVAL example scenario using our host-centric approach

note the needed relationship between the location where the condition c_1 is fulfilled, and the location where it is required. Using this approach, we succeed in keeping a host-centric vision because the additional states are directly considered in combination with the attacker location and are not represented individually.

In the next section, we will describe more formally the model used for the attack graph representation in this thesis.

5.3 Formal Model Representation

5.3.1 Environment Model

In the Cloud, the virtual infrastructures of several tenants coexist and are provisioned by leveraging Cloud devices and software: on one hand, we have physical or virtual switches, and

potentially SDN controllers, for network provisioning, and on the other hand, hypervisors for the provisioning of tenants' virtual machines. These virtual infrastructures comprise virtual machines hosted on hypervisors installed on physical servers, virtual subnets, virtual routers and virtual firewall rules organized to create a communication network able to support business needs. The proposed model aims to describe an attacker progression towards the compromise of a set of assets in the context of the Cloud.

Definition 7. *We define an asset as a high value element that needs to be protected from intruders on a physical or virtual machine. An asset can then be the device as a whole, whose compromise will be signified by reaching the highest level of privilege on the equipment, namely the root access, or a particular piece of information on a device. This identification is provided by either the Cloud infrastructure administrator or the tenant.*

The Cloud environment comprises a number of Cloud infrastructure equipment. Among those devices, we consider in our model that a physical server runs an hypervisor, which supports a number of virtual machines.

Definition 8. *Let us call H the set of hypervisors, M the set of virtual machines and I the set of all these infrastructure devices:*

$$I = H \cup M$$

The infrastructure devices mostly interact with each other by leveraging network connectivity. However, since there is a strong dependency between the physical machine and the hosted virtual machines, it is necessary to express the VMs' co-location property, specific to Cloud environments, which can results in additional communication channels in the presence of hypervisor vulnerabilities.

Definition 9. *The connectivity is defined as the following tuple:*

$$C = (i_{src}, i_{dst}, protocol, port),$$

where $i_{src} \in I$ is the source host, $i_{dst} \in I$ the destination host, $protocol$ the communication protocol and $port$ the communication port.

This definition is similar to the one of Ritchey and Ammann [63]. We only represent the communication links that are effectively allowed after taking into account components restricting the communication, such as firewalls.

Definition 10. *We define the co-location property of two virtual machines by the triple :*

$$Coloc = (vm_1, vm_2, hyp),$$

where the virtual machines $vm_1 \in M$ and $vm_2 \in M$ are both located on the hypervisor $hyp \in H$.

5.3.2 Attacker and System States Definitions

Definition 11. *Given an infrastructure device $i \in I$, we define its system state as a list of predicates relative to the device i, in which each element represents an assertion on a system attribute at a given moment, for instance mounted folders or existing files. Let $state_{sys_i}$ be the system state list of device i:*

$$state_{sys_i} = \{s_{sys\_i_1}, ..., s_{sys\_i_n}\}, n \in \mathbb{N}$$

Definition 12. *The state of an attacker can be defined as a list in which each element represents a predicate relative to an attacker attribute at a given moment, for instance his location, meaning the infrastructure device he is active on, or his level of privilege at that location. Let $state_A$ be the attacker state list of attacker A:*

$$state_A = \{s_{A_1}, ..., s_{A_m}\}, m \in \mathbb{N}$$

5.3.3 Actions Model

An attacker is able to perform a certain number of actions either at his current location or on a remote machine. However, not all actions are vulnerability exploits. Indeed, he can take advantage of services and facilities provided to regular users in order to progress in the network. This includes log-in actions or direct network access to machines. We define the set of attacker's actions as follows:

Definition 13. *Let Act be the set of actions available to the attacker. Act is the union of two sets of actions.*

$$Act = Act_{exploit} \cup Act_{feature}$$

On one hand, we have a set of exploit actions $Act_{exploit}$ that are a direct exploitation of devices vulnerabilities. On the other hand, we have a set of benign actions $Act_{feature}$ that any user could perform because they are features provided by the infrastructure, and are only threatening because they are performed by an attacker, such as logging into a machine.

Similarly to Sheyner *et al.* [4],

$$\forall \alpha \in Act, \alpha = (i_{src}, i_{dst}, r),$$

where $i_{src} \in I$ is the source from where the action is performed and $i_{dst} \in I$ the target of the action. We redefine r as a pair of elements relative to the realization conditions and effects of the action α, such that

$$r = (\text{Conditions}_\alpha, \text{Effects}_\alpha)$$

We call Conditions_α the conditions that need to be satisfied before an attacker is able to perform α. This condition set is the union of three sets of conditions: the attacker conditions $\text{Conditions}_{attacker_\alpha}$, the conditions $\text{Conditions}_{dev_\alpha}$ on the source device and the network conditions $\text{Conditions}_{net_\alpha}$ (i.e., connectivity or co-location properties).

$$\text{Conditions}_\alpha = \text{Conditions}_{attacker_\alpha} \cup \text{Conditions}_{dev_\alpha} \cup \text{Conditions}_{net_\alpha}$$

We call Effects_α the conditions that are satisfied once α is performed. Similarly they can affect either the attacker, the destination device and the network conditions.

$$\text{Effects}_\alpha = \text{Effects}_{attacker_\alpha} \cup \text{Effects}_{dev_\alpha} \cup \text{Effects}_{net_\alpha}$$

Conditions and effects are similar to pre- and post-conditions used in existing attack graph methods. A relationship is found between two actions α_1 and α_2 if the effects of the first action α_1 can be found in the conditions of the second action α_2. There is an overlap between conditions and effects of the actions available to an attacker and the attacker and system state definitions, as these states can be partially included in the preconditions of the actions.

5.3.4 Graph Definition

Representing all the conditions required for performing an action in the final attack graph would hinder readability. We define the graph so that some information is embedded in its structure. This concerns the actions availability, device characteristics and the network configuration, namely the connectivity or co-location, since they are the first prerequisites checked to decide whether an action can be performed. If a relationship is found between two actions, which results in a transition drawn in the attack graph, it means that actions availability, device characteristics and network configuration requirements are verified.

On the other hand, the analysis performed on virtualization vulnerabilities in Chapter 3 allowed to uncover sets of conditions and effects that can be found across all vulnerabilities descriptions: in order to perform an exploit, it is indispensable for an attacker to be located at a specific location in the network, and to have a specific privilege at that location. In case of a successful exploit, the effects often impact the attacker's location and privilege level by allowing him to access the targeted device. This observation can not only be extended to a large number of vulnerabilities reported in databases, but also to attacker's actions that do not result from exploits. We hence identify these two properties in the conditions and effects of attacker's actions as being representative of any action $\alpha \in Act$, and of the attacker's state before and after a successful action. This observation allows us to determine the information contained in the attack graph vertices.

Definition 14. *We define an attack graph AG as a set of vertices $\mathcal{V}$ and a set of transitions $\mathcal{T}$:*

$$AG = (\mathcal{V}, \mathcal{T})$$

Definition 15. *We define a vertex $v \in \mathcal{V}$ in the attack graph as the tuple containing the following three components: the pair of the attacker's location and privilege at that location, the attacker state and the device state.*

$$V = ((attackerLocation(loc), privilege(p)), state_A, state_{sys_i})$$

As an example, a vertex V_1, defined as

$$\begin{gathered} V_1 = \\ (\\ (attackerLocation(vm_1), privilege(root)), \\ \{attackerLocation(vm_1), privilege(root)\}, \\ \{has(secretFile)\} \\ ) \end{gathered}$$

means that an attacker is located on the virtual machine vm_1, and that he is logged there as *root*, and that the file *secretFile* exists on vm_1.

Our representation is not purely host-centric, since in addition to the location, the vertex is associated with the current attacker level on the host. However, this approach still keeps the granularity to the device level, since the attacker's location is one of the core properties defining a state.

Definition 16. *A transition $\tau \in Act \times \mathcal{V} \times \mathcal{V}$ from $v_1 \in \mathcal{V}$ to $v_2 \in \mathcal{V}$, using $\alpha \in Act$ is defined as follows:*

$$\tau = (\alpha, v_1, v_2), \text{ with } \alpha = (i_1, i_2, ((Cond_{A_\alpha}, Cond_{dev_\alpha}), (Eff_{A_\alpha}, Eff_{dev_\alpha})))$$

Let $v_1 = ((i_1, priv_1), S_{A_1}, S_{sys_1})$ and $v_2 = ((i_2, priv_2), S_{A_2}, S_{sys_2})$, $\tau \in \mathcal{T}$ (i.e. τ belongs to the attack graph) if and only if:

$$\begin{cases} Cond_{A_\alpha} \subset S_{A_1} \\ Cond_{dev_\alpha} \subset S_{sys_1} \\ Eff_A \subset S_{A_2} \\ Eff_{dev_\alpha} \subset S_{sys_2} \end{cases}$$

An illustration of this transition is represented on Figure 5.7.

Definition 17. *Let v_0 be the source vertex of the attacker and v_k, its destination vertex. An attack path between v_0 and v_k is a walk of length $k \geq 0$ in which the vertices $v_0, ..., v_k$ are all distinct.*

In other words, the attacker can perform a succession of actions allowing him to start from v_0 and reach v_k.

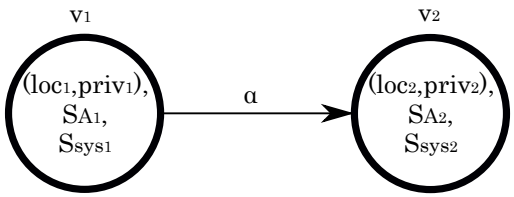


Figure 5.7: A transition between v_1 and v_2

The attack graph model, as presented here, keeps no records of the capabilities gathered by the attacker as he progresses in the network. However, an attacker can encounter scenarios in which the conditions required to perform an action α on a particular device are fulfilled by following a different path than the current one. It is the case for the satisfaction of the condition c_1 on Figure 5.6, which can only be obtained after reaching the host h_1. Such conditions are expressed as *guard conditions* on the transitions requiring α.

Definition 18. *In that context, the guard condition of $\tau = (\alpha, v_1, v_2)$ is defined as follows*

$$g_\alpha = Conditions_\alpha \setminus Conditions_{net_\alpha} \setminus \{S_{A_1} \cup S_{sys_1}\},$$

where $Conditions_\alpha$, $Conditions_{net_\alpha}$, S_{A_1} and S_{sys_1} are defined as previously respectively to α, v_1 and v_2.

In order to connect the transition having guard conditions and the vertices realizing them, we introduce an additional type of edge called a trigger. This trigger relates to the Boolean logic Driven Markov Process (BDMP) formalism [118, 119]. In BDMP, triggers are responsible for a mode change in the Markov processes associated to all or part of the leaves of the subtree the trigger points at, when the event at the origin of the trigger becomes true. We do not reuse the concept of Markov processes, but rather the notion of trigger, with the distinction that the origin of our trigger can be one or several vertices, while the destination is an edge. In BDMP however, origin and destination are both vertices.

Definition 19. *A trigger $t \in \subset 2^{\mathcal{V}} \times \mathcal{T}$ between a set of vertices v_i and a transition τ is defined as follows:*

$$t = (\{v_i\}_{0 \leq i \leq k}, \tau)$$

We call T the set of triggers.

Let

$$\tau = (\alpha, v_1, v_2),$$
$$\forall i \in [0,k], v_i = ((loc_i, priv_i), S_{A_i}, S_{sys_i}),$$
$$G_\alpha \text{ be the set of guard conditions},$$
$$G_{A_\alpha} \text{ be the set of guard conditions on the attacker},$$
$$G_{dev_\alpha} \text{ be the set of guard conditions on the device},$$
$$G_\alpha = G_{A_\alpha} \cup G_{dev_\alpha},$$

$t \in T$ *(i.e., t is a trigger between* $\{v_i\}$ *and* τ*) if and only if*

$$\begin{cases} G_{\alpha_A} \subset \bigcup\limits_{0 \le i \le k} S_{A_i} & (5.1) \\ G_{dev_\alpha} \subset \bigcup\limits_{0 \le i \le k} S_{sys_i} & (5.2) \\ \forall j \in [0,k], G_{A_\alpha} \not\subset (\bigcup\limits_{0 \le i \le k} S_{A_i}) \setminus S_{A_j} & (5.3) \\ \forall j \in [0,k], G_{\alpha_i} \not\subset (\bigcup\limits_{0 \le i \le k} S_{sys_i}) \setminus S_{sys_j} & (5.4) \end{cases}$$

The expressions 5.1 and 5.2 mean that the conditions contained in the set of guard conditions G_α are covered by the union of the attacker states and system states of the set of vertices v_i.

The expressions 5.3 and 5.4 represent the minimality of the set of vertices v_i.

An illustration of this trigger is represented on Figure 5.8. We extend the previous definition

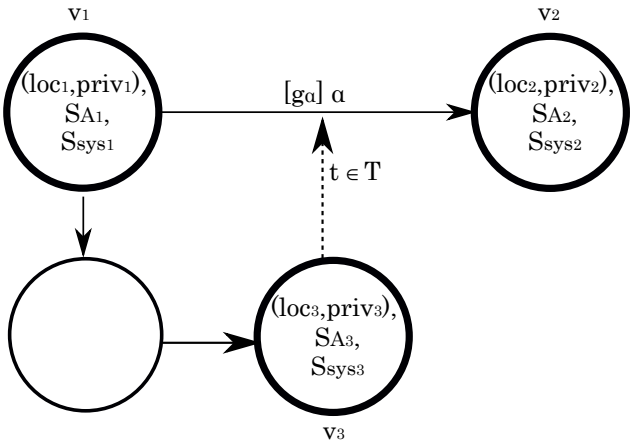


Figure 5.8: A transition with a guard condition and the associated trigger in the attack graph

of an attack graph to include the notion of triggers.

Definition 20. *We define the extended attack graph EAG as a set of vertices $\mathcal{V}$, a set of transitions $\mathcal{T}$ and a set of triggers T:*

$$EAG = (\mathcal{V}, \mathcal{T}, T)$$

Definition 21. *Let v_0 be the source vertex of the attacker and v_k, its destination vertex. An attack path in AG_E between v_0 and v_k is a walk of length $k \geq 0$ in which the vertices $v_0, ..., v_k$ are all distinct.*

In other words, the attacker is able to fire all transitions in the attack path, even the ones having guard conditions. This implies that the attacker first browsed a set of trees in which the leaf nodes are linked to the attack path using triggers.

5.3.5 Recapitulatory Example

To better illustrate the model introduced in this thesis, we will propose an example based on the MulVAL scenario, as shown on Figure 5.9. We extend it to take into account virtualization vulnerabilities.

In this topology, the *webserver*, *database server* and *workstations* are virtual machines belonging to $tenant_1$. We keep the same network configuration, connections and vulnerabilities as in the MulVAL scenario for $tenant_1$ virtual machines.

ext_1, ext_2 and ext_3 are external machines and do not belong to the virtual infrastructures considered. They are directly connected via *Internet* to the subnet *DMZ*. As a consequence, we find the same attack graph as the one presented on Figure 5.5. Since ext_1, ext_2 or ext_3 are machines located on the Internet, we keep the vertex with the generic parameter *internet* instead of adding three vertices having the exact same properties and edges as the one using the *internet* parameter.

The virtual machines of $tenant_1$ are instantiated by hypervisors H_1, H_2 and H_3. H_2 and H_3 have one *workstation* each, H_2 instantiates the *database server*, while H_1 instantiates the *webserver*.

We consider a second tenant $tenant_2$, who has the virtual machines *access* and *sensitive data* on H_1 and the virtual machine *key server* on H_3. All these virtual machines have no known vulnerabilities. $tenant_2$ stores sensitive information on the *sensitive data* VM that can only be accessed via the machine *access*. That machine is connected to the internet, but the connection is restricted to users able to present a cryptographic key that exists within a whitelist. The *key server* VM stores all the keys of the authorized users, the communication is only allowed from *private* to *internal* for administrative users.

While H_1 and H_2 contains no known vulnerabilities, H_3 exposes *CVE-2017-8903* , allowing 64 bits paravirtualized guests to escalate their privileges to that of the host, access the system memory or crash the host. Only the *workstations* are 64 bits PV guests. As a consequence, only the *workstation* VM on H_3 can be used by an attacker to exploit *CVE-2017-8903*. By doing so, an attacker might be able to retrieve a $tenant_2$ authorized user key from memory or by connecting to

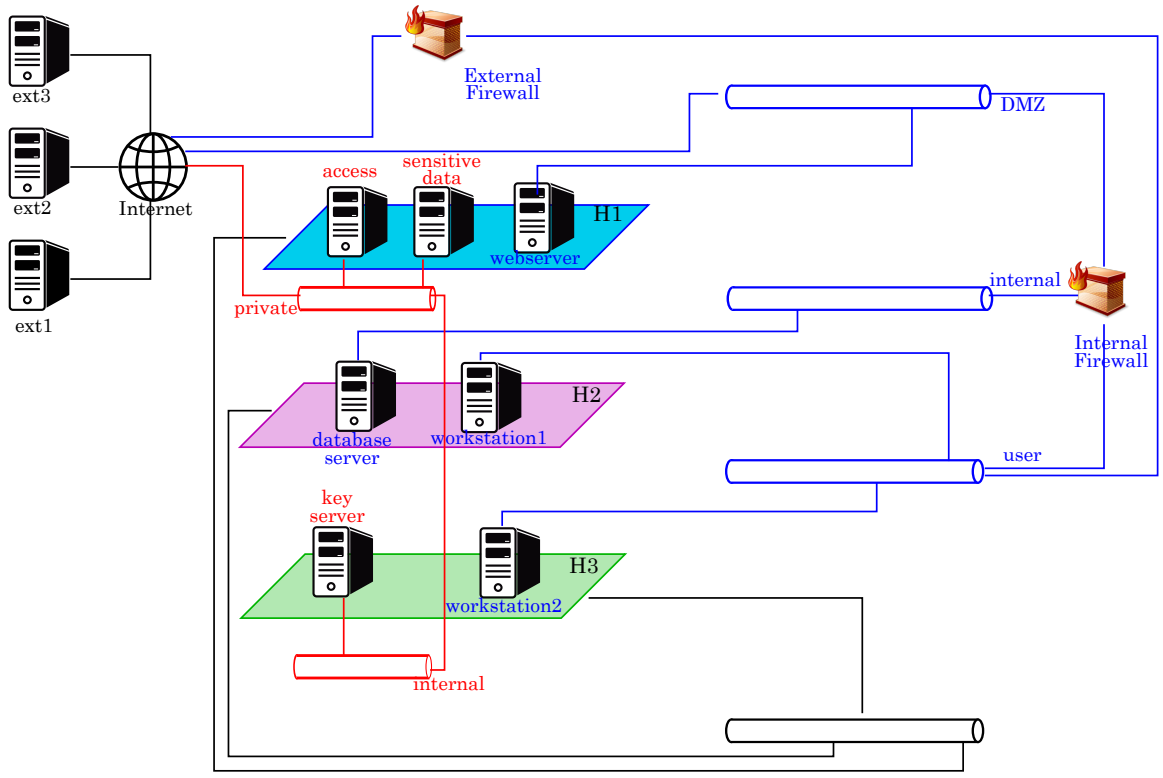


Figure 5.9: Extended MulVAL scenario

key server directly, and impersonate that user to legitimately connect to the *access* server from the Internet, and subsequently to *sensitive data* server.

As a result, despite having no vulnerabilities within its infrastructure, $Tenant_2$ will still suffer a data breach due to the vulnerability existing on H_3. The resulting attack graph using our representation model is presented on Figure 5.10.

In that figure, f_3 and f_4 are events that can only occur after f_1 or f_2.

Similarly, on Figure 5.10, the vertex ((*access, user1*), {}, {}) can only be reached after the attacker attains the vertex ((*H3, root*), {keyUser1}, {}).

5.3.6 Attack Graph Analysis Using the Proposed Model

Once the attack graph model is built, it needs to be processed in order to compute the risks faced by the assets and hence prioritize the location where counter-measures need to be applied. The common formula to calculate a risk is to multiply the probability of a disruptive event by its impact. If the impact due to an asset compromise is generally given by the infrastructure administrator, the probability is computed based on the attack graph analysis. In a risk context, we consider the worst case scenario, namely the fastest way an attacker can reach the asset. This can be obtained by running a shortest path algorithm on the attack graph. On the proposed model, it consists, on one hand, in identifying all the initial locations of an attacker. The corresponding

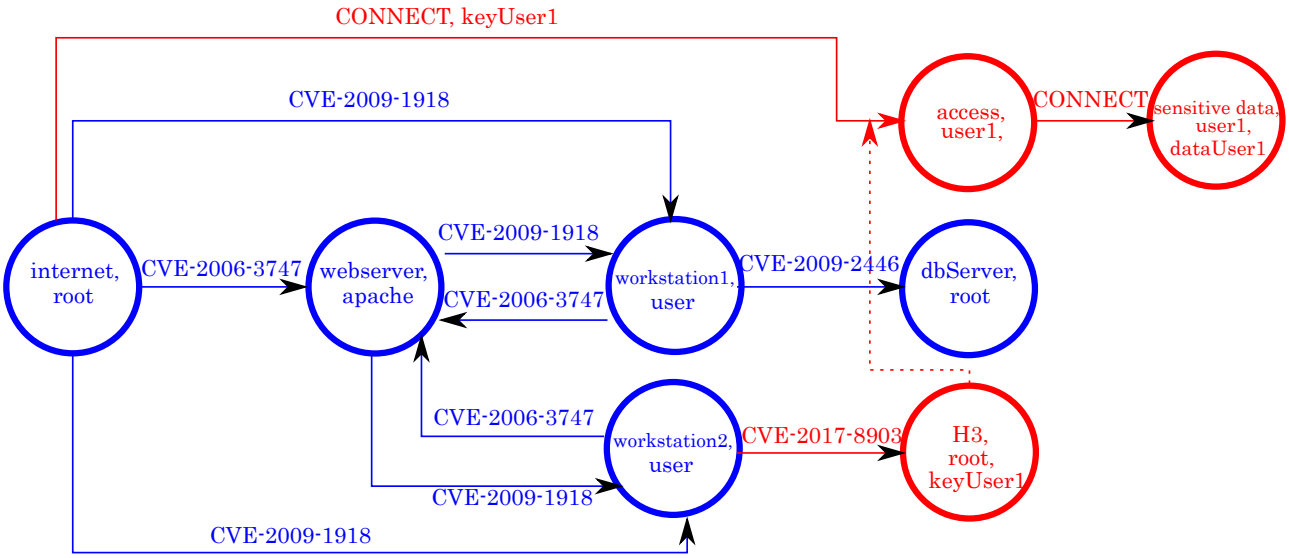


Figure 5.10: Attack graph corresponding to the extended MulVAL scenario

nodes are then linked to a virtual source node. On the other hand, we identify the location of the asset and link the corresponding nodes to a virtual target node. The edges can be weighted with the associated time required to perform the attack action labeling each edge.

If no edges with guard conditions are present in the paths from the virtual source to the virtual target, the shortest path algorithm can be run immediately.

When an edge with a guard conditions is present, it means that the attacker needs to follow a prior path before being able to fire the associated transition. As no indication is given on the time required to follow this path, an infinite value is given to the weight of each edge having a guard condition. In order to resolve these infinite values, we link all the nodes fulfilling the guard conditions to a second virtual target. The shortest path algorithm is then run between the virtual source and this second virtual target. If no paths are found, the weight stays infinite, meaning that an attacker is unable to fire the associated transition, since the conditions in the guard are not met. When a shortest path is found between the virtual source and the second virtual target, through a node fulfilling a guard condition, we assign the value obtained for the shortest path to the weight of the edge having the corresponding guard condition. The link between the second virtual target and the node fulfilling the guard condition is then removed. The shortest path algorithm is run again to find the values for the other nodes having guard conditions and linked to the second virtual target.

Once this process is done, we would have iteratively replaced the infinite values on the edges having guard conditions, where they could be replaced. It is now possible to run the shortest path algorithm between the virtual source and the virtual target in order to get the shortest path value encompassing the paths followed to meet the guard conditions.

Using the recapitulatory example previously introduced, Figure 5.11 illustrates the virtual

source and virtual targets, considering the Internet as the initial attacker location and the *sensitive data* server as the asset location.

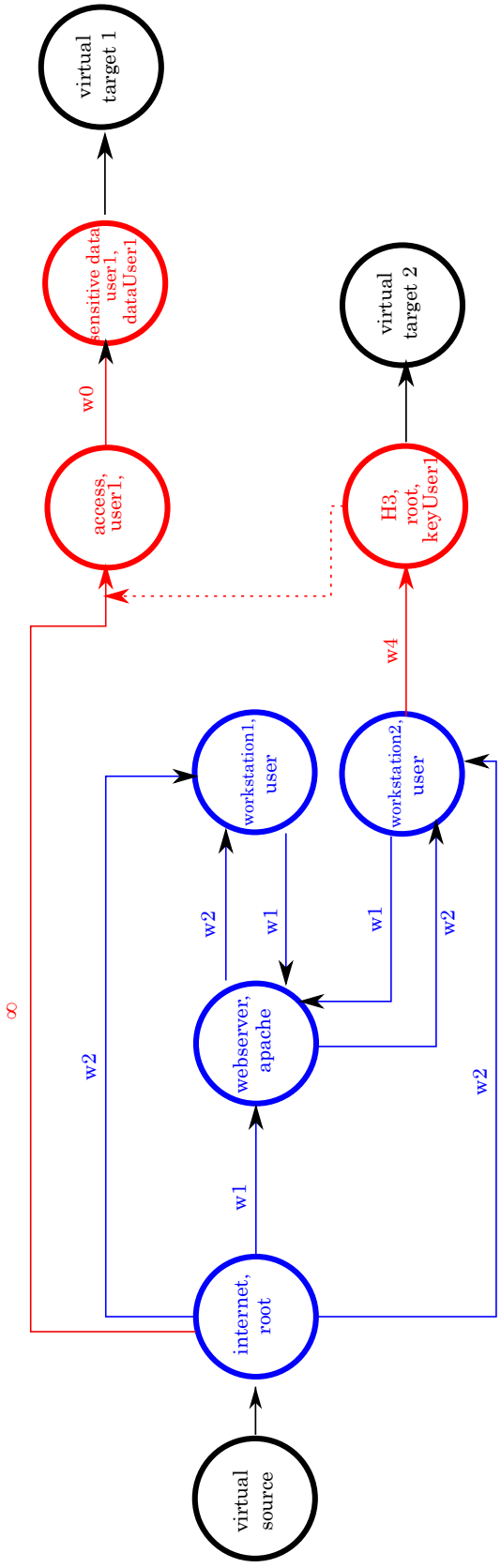


Figure 5.11: Attack graph corresponding to the extended MulVAL scenario

CHAPTER 6

ATTACK GRAPH CONSTRUCTION IN THE CLOUD

Based on the work done in Chapter 3, we are able to consider virtualization vulnerabilities in the attack graph. The extraction of the topology and connectivity performed in Chapter 4 provides us with the input needed for the graph generation. Using this prior work and the attack graph model presented in Chapter 5, we can introduce in this chapter the algorithms designed for the construction of an up-to-date attack graph in the context of the Cloud.

6.1 Attack Graph Generation in Cloud Environments

In this section, we describe the attack graph generation algorithm. Similarly to the topology and connectivity construction, it also contains two phases: a static phase and a dynamic phase, and takes as input the previously generated connectivity graph. This generation phase can then be started after the topology and connectivity builder static phase. Since the attack graph model has been presented extensively in Chapter 5, we shift the focus on the algorithms in the following section.

During the static phase, the attack graph is generated according to the current state of the Cloud infrastructure, namely the set of attack actions identified on the devices as well as the reachability information available for those devices, which is part of the set $\text{Conditions}_{net_\alpha}$ (for an action $\alpha \in Act$). During the dynamic phase, an event listener in the algorithm monitors the evolution of the infrastructure in time; it catches the notification messages generated when a modification event occurs and triggers the corresponding processing.

6.1.1 Static Phase

Methods used for building attack graphs can either begin from the attacker initial locations in the network or from the attacker goals, i.e., the infrastructure assets. In the first scenario, a forward propagation of the attacker is considered, which takes into account all the paths that he can explore, even if they do not necessarily result in an asset compromise. The second scenario considers a backward propagation, starting from the assets and gradually moving up the attack paths until potentially reaching the attacker initial locations. In this scenario, paths that do not connect with an attacker initial location are unfeasible, since the attacker has no way to reach his goals. Both methods present limitations. While with a backward propagation, we create unattainable attack paths, that should not be existing in the attack graphs due to their unavailability to the attacker, with a forward propagation, all the paths created can be browsed by the attacker, even if they do not necessarily reach the current assets considered in the infrastructure.

In our context, the attackers initial locations are considered as input data provided by the tenants of the Cloud. They represent entry points that the tenants provide to their clients in order to access their virtual infrastructure. The assets, namely the machines that need to be protected, are also an input provided by the tenants, since they know their environment better.

In the algorithms considered in this thesis, we will mainly use the forward propagation algorithm, since this offers more possibilities of evolution. Indeed, the attack graph created in this context has all the attacker possibilities. Thus, if additional assets are considered in the infrastructure while the attacker initial locations remain the same, it is possible to identify whether these new assets can be reached by the attacker by simply browsing the existing attack graph.

The automated construction of the attack graph is done in several steps, starting from the initial attacker locations. Two subsets of attack actions are considered in the generation process, and used in different steps of the algorithm. The distinction is established based on the attacker actions effects, not their conditions. The first subset concerns attack actions which only impact the attacker location and his level of privilege. The second subset concerns attack actions whose impact goes beyond the attacker location and his level of privilege. Each of these subsets causes the creation of a particular type of vertices in the attack graph, hence establishing such distinction allows to handle each type of vertices independently.

- Step 1: Firstly, the initial attack graph is built using attack actions whose only impacts are on the attacker location and his level of privilege. This represents the first subset of attack actions.

  This process is based on a depth first exploration of the connectivity graph. This type of exploration allows to imitate the way an attacker gradually progresses in the network: once an access is obtained on a machine, this machine can be used as a foothold to pivot in the

network and target directly connected devices, or hosting hypervisors. Hence from machine to machine a path can be created to reach the assets.

At each network host reached, we determine the following attack actions accessible on the host directly connected or on the hosting hypervisor. In this step, there is no notion of guard conditions: links are added in this first graph without determining whether the guard conditions g_α exist on the edges or can be fulfilled. Only the conditions and effects on the attacker location and his level of privilege are considered. This means that during this step of the algorithm, we build the links without any assumptions on the attacker's ability to fulfill these guard conditions.

- Step 2: Secondly, we introduce the guard conditions. We label the edges with the attack action conditions in Conditions_α, which are different from the attacker location and his level of privilege.

  This means that in the graph obtained after the first step, edges relative to attack actions requiring more than the correct attacker location and privilege level in order to be exploited are identified. These edges are updated to exhibit these supplementary conditions, which are the guard conditions.

- Step 3: Thirdly, we need to determine how these guard conditions can be satisfied in the network. To that end, we use the second subset of attack actions, which consists in actions impacting dimensions other than the attacker location and his level of privilege. This additional dimensions are the ones considered by the guard conditions.

  We build the intermediary goals based on attack actions that include the guard conditions in their effects. These intermediary goals represent the vertices that the attacker needs to reach to fulfill the guard conditions and traverse the paths labeled in the second step.

  It is possible that some guard conditions are not found in the effects of any of the attack actions in the second subset. In that case, it means that these guard conditions cannot be realized by the attacker and the related edges cannot be fired by the attacker.

- Step 4: The intermediary goals created in the third step are not yet connected to any part of the graph obtained in the second step. We build the paths going to and coming from these intermediary goals using a backward and forward exploration starting from the intermediary goals, using the first subset of attack actions, like in Step 1.

  This allows to either complete paths in the graph that are interrupted due to only considering a subset of attack actions in Step 1, create new individual paths or reconnect the intermediary goals with existing branches of the graph.

  Step 2, Step 3 and Step 4 are repeated if necessary, meaning if edges with guard conditions are added to these newly created paths. We stop when no new intermediary goals identical

to the ones already in the graph are added in Step 3. This means that, when possible, paths fulfilling all the guard conditions identified have been built.

However, not all the built paths are relevant, some are unfeasible if they do not directly connect to the attacker initial locations or existing vertices reachable to the attacker.

At the end of this step, we have the final attack graph comprising the paths to the assets and the paths followed by an attacker to fulfill the guard conditions (when these paths exist) enabling to fire edges related to these guards.

The triggers do not need to be explicitly added to the attack graph, as they are only used to connect an edge having a guard condition to the node where this condition is realized. As we presented in the analysis in Section 5.3.6, these links are not used when running the graph algorithm on the attack graph.

6.1.2 Dynamic Phase

After the static phase, the dynamic phase takes over and initializes an event listener. Since an attack graph is modified by vulnerabilities or device reachability changes, this approach requires the collection and processing of events originating from two sources: on one hand the Cloud Management Platform (CMP) which has the knowledge on every virtualized infrastructure change relative to the topology, and thanks to our topology and connectivity builder, changes relative to the connectivity; and on the other hand, vulnerability scanners reporting the existence of a vulnerability on a device or the resolution of the vulnerability, leading to its deletion from a device. Similarly to the CMP, vulnerability scanners should either be able to generate the vulnerability creation and deletion events according to each device in the infrastructure, or should be extended in order to so. Each notification message should at least contain information on the *vulnerability ID* and *device ID* on which it exists. These vulnerability-related events can then be leveraged in a up-to-date attack graph generation algorithm. We present the essential vulnerabilities events required, as well as their definitions in Table 6.1.

Vulnerabilities Events	**Definitions**
vulnerability.create.end	A new vulnerability is discovered in the infrastructure
vulnerability.add.end for device i	A given vulnerability is added to a particular device
vulnerability.remove.end for device i	A given vulnerability is removed from a particular device
vulnerability.delete.end	A given vulnerability is deleted from the infrastructure it doesn't exist on any device in the infrastructure anymore

Table 6.1: Vulnerability scanner minimal event list

In our context, for illustration purposes, we consider that we are able to track the vulnerability changes occurring in the infrastructure and obtain exploitable notification messages for the attack graph generation.

In Table 6.2, we present the possible impact of the events considered on the attack graph, provided the conditions, i.e, vulnerability or action existence, are verified. As a rule of thumb, the reachability and vulnerability change events are the ones we need to track in order to update our attack graph. As previously presented, connectivity events (in the reachability category) are generated by processing simple events provided by the CMP. Some simpler events can be

Events	**Impacts on the Attack graph**
connectivityLink.create.end	Create vertices $((attackerLocation(loc), privilege(p)), state_A, state_{sys_i})$ if not exists. Create outgoing transitions to connected destination if allowed by attack action.
connectivityLink.delete.end	Delete all transitions between source and destination.
virtualMachine.delete.end for device i	Delete all $((attackerLocation(loc), privilege(p)), state_A, state_{sys_i})$. Delete all outgoing transitions from $((attackerLocation(loc), privilege(p)), state_A, state_{sys_i})$. Delete all incoming transitions to $((attackerLocation(loc), privilege(p)), state_A, state_{sys_i})$.
hypervisor.delete.end	Delete all $((attackerLocation(loc), privilege(p)), state_A, state_{sys_h})$ where h is the hypervisor. Delete all $((attackerLocation(loc), privilege(p)), state_A, state_{sys_i})$ where i is a guest virtual machine. Delete all outgoing transitions from $((attackerLocation(loc), privilege(p)), state_A, state_{sys_h})$ and $((attackerLocation(loc), privilege(p)), state_A, state_{sys_i})$. Delete all incoming transitions to $((attackerLocation(loc), privilege(p)), state_A, state_{sys_h})$ and $((attackerLocation(loc), privilege(p)), state_A, state_{sys_i})$.
vulnerability.create.end on device dst	Create incoming transition τ from devices able to communicate with dst to dst
vulnerability.delete.end	Delete all transitions τ

Table 6.2: Modifications incurred by the events on the attack graph

analyzed as is, since they contain all the information needed for our graph update: that is the case for the deletion of a virtual machine, which we are directly able to process by deleting all the $((attackerLocation(loc), privilege(p)), state_A, state_{sys_i})$ vertices in the graph, as well as their incoming and outgoing edges, where i is the virtual machine in question.

The events received are treated according to the impacts described in Table 6.2. In Algorithm 3, we detail the processing applied to some of the events presented in Table 6.2.

In terms of complexity, the Depth First Search (DFS) algorithm being the main component of the static phase algorithm, the theoretic complexity of this phase should be close to the DFS complexity, which is $O(|V| + |E|)$ where $|V|$ and $|E|$ are the number of vertices and edges respectively.

By using the dynamic phase to keep the attack graph updated, we are able to reduce this complexity, since every event will only affect a portion of the global graph. Besides, this complexity is dependent on the event since some will only concern a portion of the edges, a portion of the vertices or a portion of both edges and vertices.

Algorithm 3: Dynamic Phase

Data: Attack actions database A_{db}, Reachability database R_{db}, Static Attack Graph AG
Result: Updated Attack graph of the virtualized environment

```
1  eventListener = initializeEventListener()
2  eventListener.listen()
3  if eventListener.notification then
4      message = eventListener.notification
5      if message.type = 'vulnerability.create.end' then
6          vulnLoc = message.payload['vulnLoc']
7          devices = getDevicesReachingLocation(vulnLoc)
8          conditions = getConditions(A_db, message.payload['vulnId'])
9          foreach device in devices do
10             possibleCurrentVertices = getPossibleCurrentVertices(AG, device)
11             foreach currentVertex in possibleCurrentVertices do
12                 if match(currentVertex, conditions) then
13                     createTransition(device, vulnLoc, message.payload['vulnId'])
14     if message.type ='connectivityLink.create.end' then
15         src = message.payload['source']
16         dst = message.payload['destination']
17         possibleNextActions_List = getDynamicNextActions(src, dst, A_db)
18         possibleCurrentVertices = getPossibleCurrentVertices(AG, src)
19         foreach action in possibleNextActions do
20             foreach currentVertex in possibleCurrentVertices do
21                 if match(currentVertex, conditions(action)) then
22                     createTransition(src, dst, getIdAction(action))
23     if message.type =... then
       /* processing the rest of the modification messages                    */
```

CHAPTER 7

EXPERIMENTATIONS

In order to validate the pertinence of our approach, we designed a working prototype deployed on a real platform at scale. In this chapter, we present this prototype and the experiment platform used in our study. Finally, we expose our experimentation strategy, as well as the results obtained during the test campaigns.

7.1 Experiment Platform

7.1.1 Cloud Management Platform

CloudStack [120] and Eucalyptus [121] are both open source softwares for deploying IaaS clouds, and are alternatives that could be used as Cloud Management Platforms.

However, we based our solution on OpenStack due to its modularity enabling the deployment of a subset of services required, compared with more monolithic platforms like CloudStack. It is also a platform regularly used in the research community, and that has great traction with enterprises and service providers [1].

OpenStack is an open source software for building Cloud platforms and controlling pools of compute, storage and networking resources. Regarding network management, this CMP presents a proven integration with the chosen SDN controller, comforting us in our choice to use it.

Information regarding the state of the platform are stored in databases by OpenStack. It follows a modular design as presented in Section 4.1.1.1. We deploy the minimum configuration, which includes the Identity, Compute and Network services, respectively dubbed Keystone, Nova and Neutron, each of them having a dedicated database. In the Keystone database, we retrieve information on the tenants and their associated projects, a project being the canvas in which

[1] https://www.openstack.org/analysts/

users define their virtual environments (networks, virtual machines, security groups, etc.) The information on the hypervisors and the virtual machines they host is obtained by interrogating the Nova database, while the networks, subnets, routers, security groups and security rules are extracted from the Neutron database.

7.1.1.1 SDN Controller

Given the scale of the Cloud and requirements for availability and performance, we opted for the use of ONOS (Open Network Operating System) [122], an SDN controller oriented towards service providers. It is designed to scale with the size of the network with the ability to get a cluster of controllers, hence it represents a good fit for service providers. Besides, its integration with the OpenStack platform and its rich API, easing new applications development, are additional reasons motivating our choice. Alternatively, OpenDaylight [123] is an SDN controller that could be used in replacement of ONOS, and is also well integrated with OpenStack. In ONOS, the interaction with the OpenStack platform is realized by the intermediary of the SONA (Simplified Overlay Network Architecture application, which is responsible for intercepting and interpreting the networking requests sent by the infrastructure customers. It is an optimized tenant network virtualization service for Cloud-based data center. We leveraged the ONOS *FlowRuleService* to get information on flow rule events in virtual switches, namely updated, created or deleted flow rules.

Flow rules unique identifiers allow to track their life-cycle. In flow rules, we first extract information on priority and matching patterns targeted (protocol, transport layer port, ip address, virtual switch port number, mac addresses of virtual switch ports connected to virtual machines, etc.). Next, we associate it to the corresponding virtual machine in OpenStack. Then, we identify the treatment applied to the selected traffic (drop, allow) for that virtual machine. Both pieces of information are stored in the SDN rule database. When an event notification is made, the rule ID is used to modify, create or delete the corresponding entry.

7.1.2 Graph Database

In our study, we opted for the use of a graph database as storage solution for our topology and connectivity builder, as well as our attack graph generation algorithm. Compared with relational databases, graph databases are a better fit for highly interrelated data. Indeed, this kind of data can lead to complex and costly *JOIN* operations to obtain a result to a formulated request. The expression of path queries is also difficult, as they requires the use of closure tables and fixed-point data types. Additionally, using a graph database offers more flexibility, since the data is not constrained to a rigid structure like in relational tables, and attributes can be added and removed easily. This is useful for semi-structured data whose representation would result in lots of *NULL* column values in relational databases.

In our context, the highly interrelated information needed to reconstruct tenants' topology and connectivity is scattered in several database tables in the CMP. Hence, adopting the traditional relational database model of OpenStack would be detrimental to performance. Following on our graph representation of the environment, we found a suitable and flexible storage solution in graph databases. Indeed, such solutions represent a natural fit for scenarios such as network topology retrieval, as they natively store data as graphs, with edges representing relationships between typed vertices. Pattern matching and graph traversal operations allows a fast query processing. We opted for the use of Neo4j [124], an open-source graph database sthat shows interesting performance as illustrated in [125]. The use of Neo4J as database allows to build labeled vertices and edges as defined by the environment graph model (Figure 4.4). Furthermore, they can be augmented with additional properties. For instance, by adding a tenant id property to each concerned node, we avoid cluttering the graph with multiple ownership relationships while maintaining an efficient traversal.

Time complexity of the queries depend on the query pattern and what is really requested in the query. However, we can have a sense of the complexity for simpler traversals. In Neo4j, graph nodes are linked to their neighbors directly in the storage [2], following the index-free adjacency principle. With this principle, graph databases do a direct walk of memory, since they have direct physical RAM addresses between a node and its neighboring counterparts. Since links are explicitly added between related nodes, graph databases do not rely on intermediary data structures or indexes for hopping from one node to the next. As a result, a graph query allowing to retrieve a subgraph is simply performed by following the links in the storage, resulting in a constant *O(1)* complexity for a node to node traversal. By adequately constraining the query to touch only the relevant part of the global graph, the final complexity of a query is hence dependent on the number of elements in the final result: *O(k)* for instance, for k element in the final result. Performing a friend of a friend lookup to find all indirect friends of Bob for example, would have a complexity proportional to the number of Bob's indirect friends.

7.1.3 Hardware Platform

Experiments presented are realized using the Grid'5000 testbed, which is supported by a scientific group hosted by Inria and including the CNRS, RENATER and several universities as well as other organizations [126]. It offers a Hardware as a Service platform, allowing its users to reserve entire physical servers, that can deployed and configured at will, according to their experiments' requirements.

For the purpose of our experiment, we selected physical servers having 2 Intel Xeon E5-2630 v3 CPUs, 8 cores per CPU and 128 GB of RAM.

We used a total of 63 servers, distributed as follows:

[2] https://dzone.com/articles/why-are-native-graph-databases-more-efficient-than

- 1 server serving as Cloud controller: this machine hosts the OpenStack identity service (Keystone), as well as the control section of the compute (Nova) and network service (Neutron),

- 1 server hosting the SDN controller, ONOS,

- 1 server hosting our topology and connectivity builder as well as the attack graph generation algorithm. The Neo4j graph database is also implemented on this server for data storage,

- 60 servers hosting the compute service of the OpenStack platform to provide resources to the tenants.

We keep the default routing configuration of the Grid'5000 platform in which all physical machines belong to the same subnet and can communicate with each other.

7.1.4 Tenants' Virtual Infrastructure

A great difficulty in the elaboration of the experimentation strategy was the ability to access real-life Cloud provider information regarding the virtual infrastructures deployed on their platforms. This includes on one hand the approximate workload (i.e., the number of virtual machines) of their customers, as well as the most common infrastructures deployed. Such information would allow to properly scale the infrastructure and pinpoint the relevant parameters to examine. Regarding the customers workload, we finally relied on the RightScale 2017 report [127], in which a survey is made on the VM loads most commonly run by Openstack users. As a result, the largest fraction of Cloud users (16%) have between 1 and 50 VMs per OpenStack instance, while only 4% have over 1000 VMs, with no indications on the number of tenants in the infrastructure. As a consequence, we vary the number of VMs between 100 and 1000, for the tenants virtual infrastructure. Regarding the topology of this virtual infrastructure, we leveraged infrastructure templates provided by the Amazon Web Service (AWS) platform, with the assumption that the majority of the tenants would use it as a foundation for the deployment of their own environment. In order to have a slightly more complex deployment than the basic 3-tier application, we opted for the *Asynchronous Online Gaming* template [13], that we adapted to our environment. AWS proposes to use this architecture template for the deployment of mobile and online games. It is architected to cope with unexpected traffic patterns and highly demanding request rates. It also offers the possibility to start with a smaller environment and power up the architecture according to the number of players. The resulting virtual tenant infrastructure deployed in OpenStack is represented on Figure 7.1. The virtual machines groups with a yellow background are the ones belonging to a scaling group and which can be dynamically provisioned to cope with an increase in traffic.

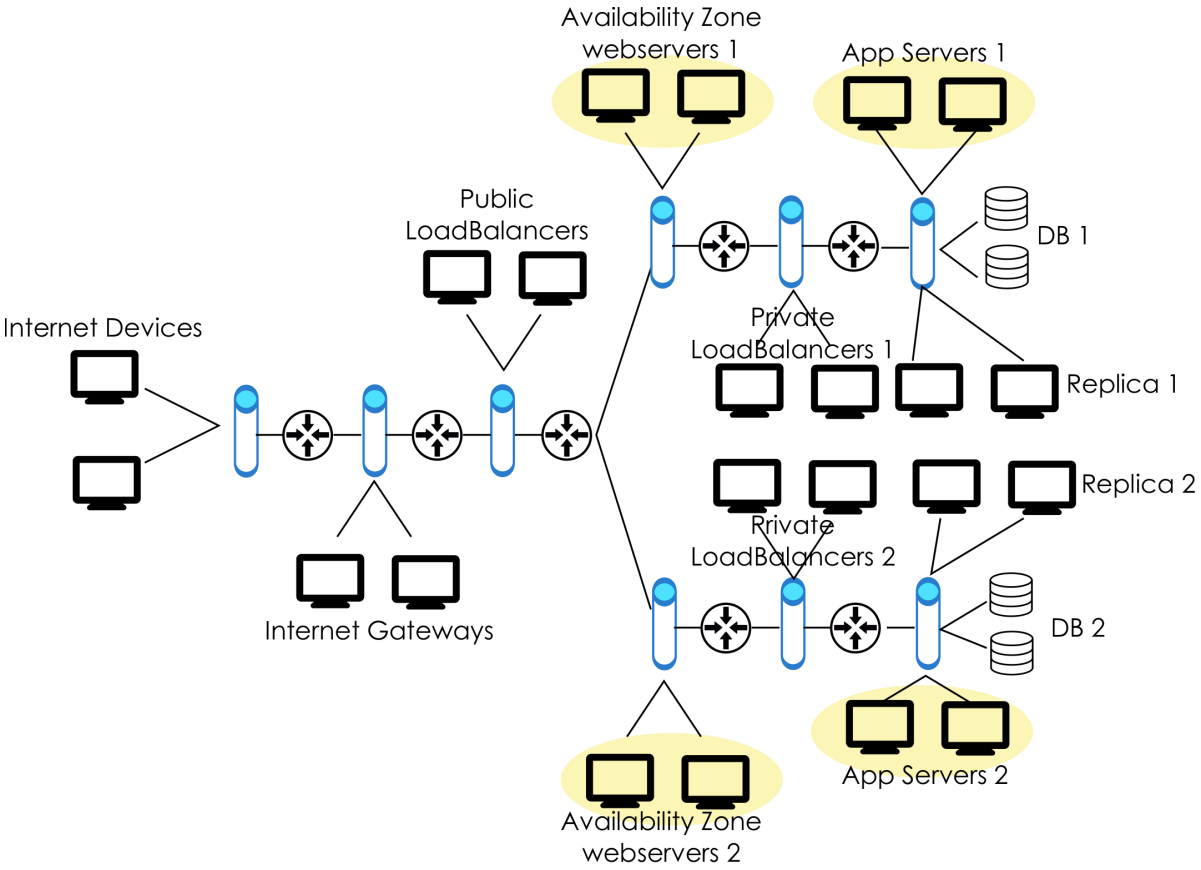


Figure 7.1: Tenant's virtual infrastructure following the Asynchronous Online Gaming AWS template [13]

7.2 Prototype

The prototype implements the proof of concept of our approach towards an up-to date connectivity retrieval and attack graph generation in Cloud environments. It is built to interact with the OpenStack platform, the ONOS controller, and the Neo4J database. The following elements can be found in the prototype:

- a first graph database that holds on one hand the topology and connectivity view of the infrastructure, as well as the vulnerabilities found on each device;
- a second graph database containing the attack graph generated;
- a messaging queue holding the notifications intercepted from the CMP and the SDN controller, or generated by the prototype we developed;
- a monitoring application installed within the SDN controller and storing the relevant notification messages in the messaging queue;

- a static topology and connectivity algorithm responsible for establishing the baseline view of the infrastructure based on data retrieved from the CMP and the SDN controller;
- a static attack graph generation algorithm that builds the initial attack graph based on the output of the static topology and connectivity algorithm and the current vulnerability inventory;
- a dynamic algorithm that simultaneously intercepts connectivity- and vulnerability- related events, stores them as they arrive in the messaging queue, processes queue events in a FIFO manner, and selectively updates on one hand the topology and connectivity graph, and on the other the attack graph generated accordingly.

7.3 Global Experiment Scenario

In this section, we present the methodology used to perform our experiments. We start by deploying and configuring the physical infrastructure obtained via Grid'5000 hardware reservations, in order to obtain a Cloud infrastructure leveraging OpenStack and the ONOS controller. In a second phase, we provision the tenants' virtual infrastructures following the AWS template presented earlier. The servers in the *web server* and *application* clusters are scaled up with each running experiment in order to simulate increasing workloads.

With this virtual infrastructure provisioned, we can assign vulnerabilities to a fraction of the physical and virtual devices, in order to simulate a vulnerable environment. These vulnerabilities are picked from the NIST vulnerability database and depicted in Table 7.1. They represent a mix of hypervisor and traditional environment vulnerabilities. This selection is used to illustrate different types of vulnerability impacts, namely privilege escalation, code execution, data leakage and denial of service. Concerning virtualization vulnerabilities, they also express different type of constraints required for a successful exploit, namely the type of virtual machines involved, the platform architectures, type of processors or attacker privileges. The vulnerabilities in Table 7.1 are assigned randomly to virtual machines in the infrastructure for traditional environment vulnerabilities. A similar assignment process is performed between the infrastructure hypervisors and the virtualization vulnerabilities. The number of vulnerabilities affected to each device is also determined randomly. A random approach is used since there is no study or information on the number or repartition of vulnerabilities in the tenants infrastructures in the Cloud.

Using this initial setup, we build the baseline attack graph, which takes as input the initial connectivity and vulnerability inventory retrieved from the infrastructure and obtained using our algorithm. Subsequently, to represent the life cycle of the virtual infrastructure, change events are generated, namely virtual machine creation, deletion and migration. In addition, vulnerability modifications are also triggered by adding them to or removing them from the devices. The events are randomly generated and serve to establish the dynamic performances of the attack graph construction algorithm. A hundred events of each type are generated for virtual

CVE Identifier	Vulnerability Description from the NVD database
CVE-2017-12611	In Apache Struts 2.0.1 through 2.3.33 and 2.5 through 2.5.10, using an unintentional expression in a Freemarker tag instead of string literals can lead to a RCE attack.
CVE-2017-9791	The Struts 1 plugin in Apache Struts 2.3.x might allow remote code execution via a malicious field value passed in a raw message to the ActionMessage.
CVE-2017-0144	The SMBv1 server in Microsoft Windows Vista SP2; Windows Server 2008 SP2 and R2 SP1; Windows 7 SP1; Windows 8.1; Windows Server 2012 Gold and R2; Windows RT 8.1; and Windows 10 Gold, 1511, and 1607; and Windows Server 2016 allows remote attackers to execute arbitrary code via crafted packets, aka "Windows SMB Remote Code Execution Vulnerability." This vulnerability is different from those described in CVE-2017-0143, CVE-2017-0145, CVE-2017-0146, and CVE-2017-0148.
CVE-2014-0160	The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the Heartbleed bug.
CVE-2014-6271	GNU Bash through 4.3 processes trailing strings after function definitions in the values of environment variables, which allows remote attackers to execute arbitrary code via a crafted environment, as demonstrated by vectors involving the ForceCommand feature in OpenSSH sshd, the mod_cgi and mod_cgid modules in the Apache HTTP Server, scripts executed by unspecified DHCP clients, and other situations in which setting the environment occurs across a privilege boundary from Bash execution, aka "ShellShock." NOTE: the original fix for this issue was incorrect; CVE-2014-7169 has been assigned to cover the vulnerability that is still present after the incorrect fix.
CVE-2015-7547	Multiple stack-based buffer overflows in the (1) send_dg and (2) send_vc functions in the libresolv library in the GNU C Library (aka glibc or libc6) before 2.23 allow remote attackers to cause a denial of service (crash) or possibly execute arbitrary code via a crafted DNS response that triggers a call to the getaddrinfo function with the AF_UNSPEC or AF_INET6 address family, related to performing "dual A/AAAA DNS queries" and the libnss_dns.so.2 NSS module.
CVE-2014-3566	The SSL protocol 3.0, as used in OpenSSL through 1.0.1i and other products, uses nondeterministic CBC padding, which makes it easier for man-in-the-middle attackers to obtain cleartext data via a padding-oracle attack, aka the "POODLE" issue.
CVE-2015-3456	The Floppy Disk Controller (FDC) in QEMU, as used in Xen 4.5.x and earlier and KVM, allows local guest users to cause a denial of service (out-of-bounds write and guest crash) or possibly execute arbitrary code via the (1) FD_CMD_READ_ID, (2) FD_CMD_DRIVE_SPECIFICATION_COMMAND, or other unspecified commands, aka VENOM.
CVE-2017-8903	Xen through 4.8.x on 64-bit platforms mishandles page tables after an IRET hypercall, which might allow PV guest OS users to execute arbitrary code on the host OS, aka XSA-213.
CVE-2017-8905	Xen through 4.6.x on 64-bit platforms mishandles a failsafe callback, which might allow PV guest OS users to execute arbitrary code on the host OS, aka XSA-215.
CVE-2017-12137	arch/x86/mm.c in Xen allows local PV guest OS users to gain host OS privileges via vectors related to map_grant_ref.
CVE-2017-12855	Xen maintains the _GTF_read,writing bits as appropriate, to inform the guest that a grant is in use. A guest is expected not to modify the grant details while it is in use, whereas the guest is free to modify/reuse the grant entry when it is not in use. Under some circumstances, Xen will clear the status bits too early, incorrectly informing the guest that the grant is no longer in use. A guest may prematurely believe that a granted frame is safely private again, and reuse it in a way which contains sensitive information, while the domain on the far end of the grant is still using the grant. Xen 4.9, 4.8, 4.7, 4.6, and 4.5 are affected.
CVE-2017-15595	An issue was discovered in Xen through 4.9.x allowing x86 PV guest OS users to cause a denial of service (unbounded recursion, stack consumption, and hypervisor crash) or possibly gain privileges via crafted page-table stacking.

Table 7.1: Vulnerabilities used in the experiment

machine creation, virtual machine deletion, virtual machine migration, vulnerability removal from a device or virtual machine addition to a device.

In a first phase of event generation, virtual machine creation, deletion and migration events are triggered in a random order to simulate the life-cycle of a real Cloud virtual infrastructure. These events account for a total of 300 notifications.

In a second phase, events related to vulnerability removal from or virtual machine addition to a device are triggered, also in a random order, to illustrate the discovery or treatment of vulnerabilities in the infrastructure. These events account for a total of 200 notifications.

7.4 Results

The experiments performed aim to evaluate our approach for attack graph construction considering the different phases involved in the algorithm, i.e. on one hand, static and dynamic topology and connectivity construction, and on the other hand, static and dynamic attack graph construction.

7.4.1 Vertices and Edges generated

Figure 7.2 and Figure 7.3 represents the number of nodes and edges created in the connectivity graph on one hand, and in the attack graph on the other hand. The red line represents the linear regression of the data points on the graph. The plot relative to the connectivity graph illustrates a clear linear progression in the number of virtual machines. Concerning the resulting attack graph, the shape of the plot relative to the number of edges is explained by the randomized nature of the vulnerabilities assignment. Indeed, only a fraction of the virtual devices created in the infrastructure will effectively have a vulnerability. Moreover, the vulnerability distribution in the infrastructure does not necessarily result in a successful attacker exploit, since conditions required to chain vulnerabilities are not necessarily satisfied, due to their non-deterministic dispersion. This explains why the number of attack paths created in the attack graph has a greater variance with the number of virtual machines, since its depends on additional parameters, namely the presence of vulnerabilities in the infrastructure and their ability to be successfully leveraged by an attacker in a chain of actions in order to compromise a final goal. Every data point corresponds to a single instance of the experiment with the corresponding workload, and not the mean of several experiments performed with the same number of virtual machines. This is justified by the fact that we use a hardware as a service platform, with constraints on the duration of the reservation of the hardware provided and specific hours for running the experiments. This implies to rebuild the provisioned infrastructure from scratch every time an experiment needs to be run. In addition, the provisioning of the infrastructure itself and configuration of the virtual infrastructure takes a significant amount of time of the reservation, which can go up to six hours for higher workloads in the best case scenario, when no failure in the infrastructure deployment occurs.

7.4.2 Time Performance and Static Phase Algorithm Complexity

Figure 7.4 represents on the left side the connectivity construction time and on the right side the attack graph construction time.

The static connectivity curve is represented on Figure 7.4 as a function of the number of virtual machines. It shows an experimental complexity that is quadratic in the number of virtual machines.

Regarding the attack graph construction complexity, it should be close to the Depth First Search algorithm complexity which is $O(|V|+|E|)$ where $|V|$ is the number of vertices and $|E|$ the number of edges. It is indeed this traversal method that is the principal mechanism use in our attack graph construction algorithm. We hence represented on Figure 7.4 the time required for the attack graph construction as a function of the number of vertices and edges generated by the algorithm. We can observe on this plot a linear trendline that is coherent with the theoretic complexity of this section of the algorithm. However, the benefit from the event-based processing we adopted is that we are freed from successively repeating the time overhead incurred by the

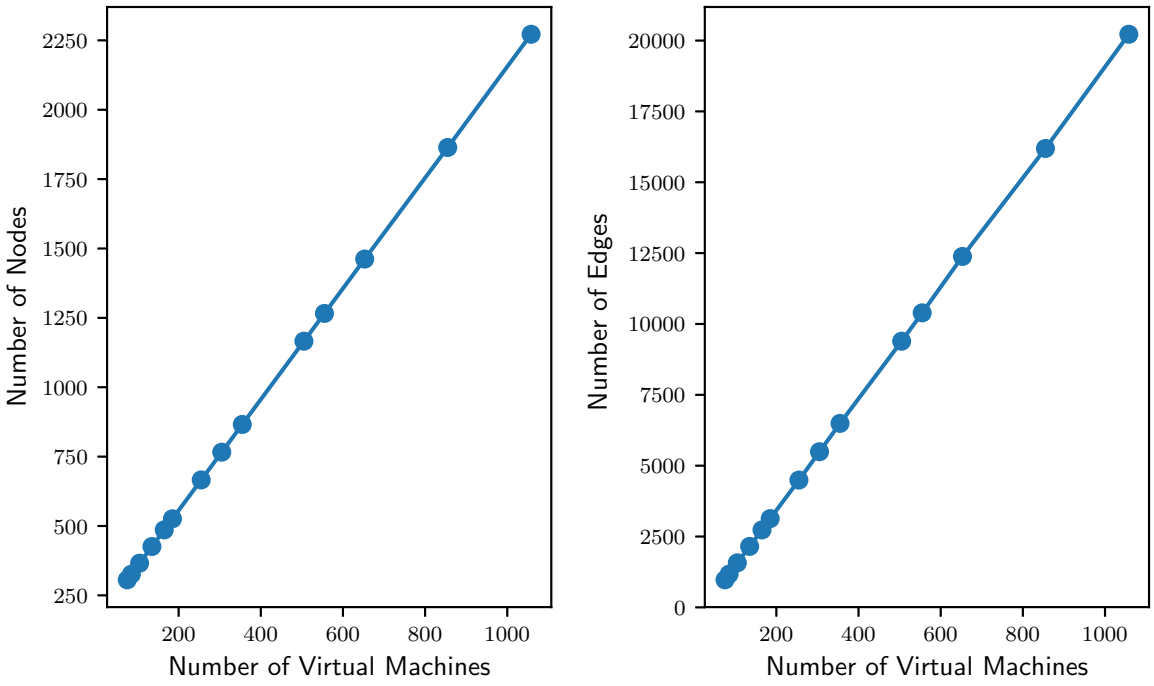


Figure 7.2: Number of nodes and edges generated in the connectivity graph

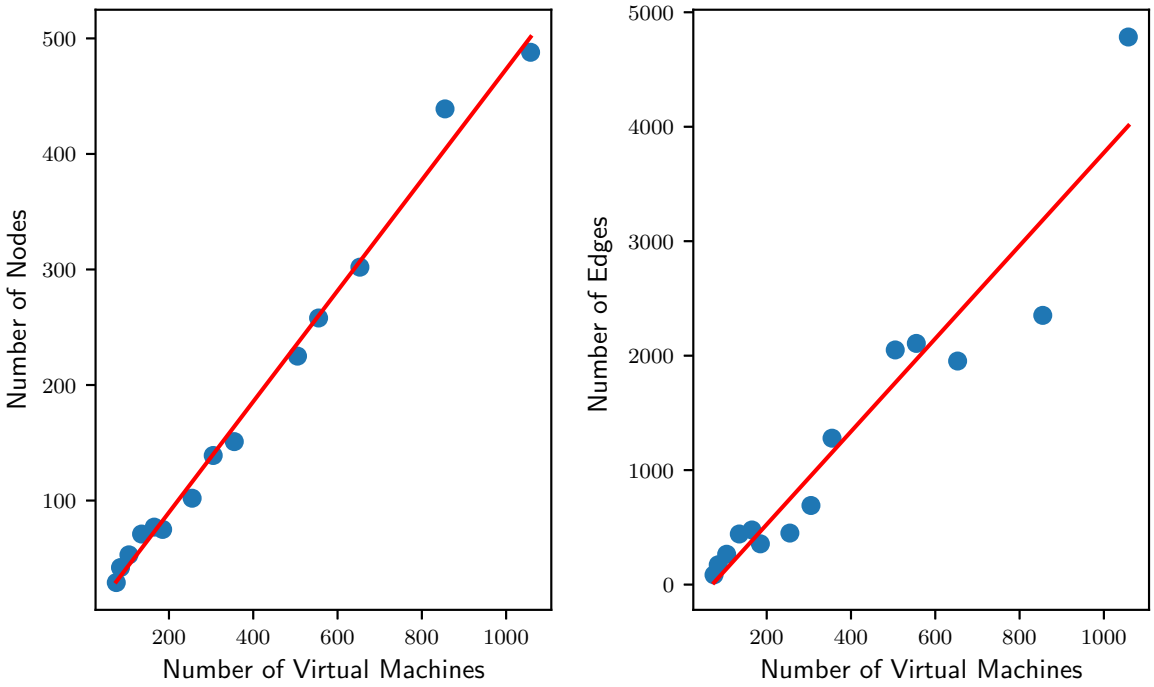


Figure 7.3: Number of nodes and edges generated in the attack graph

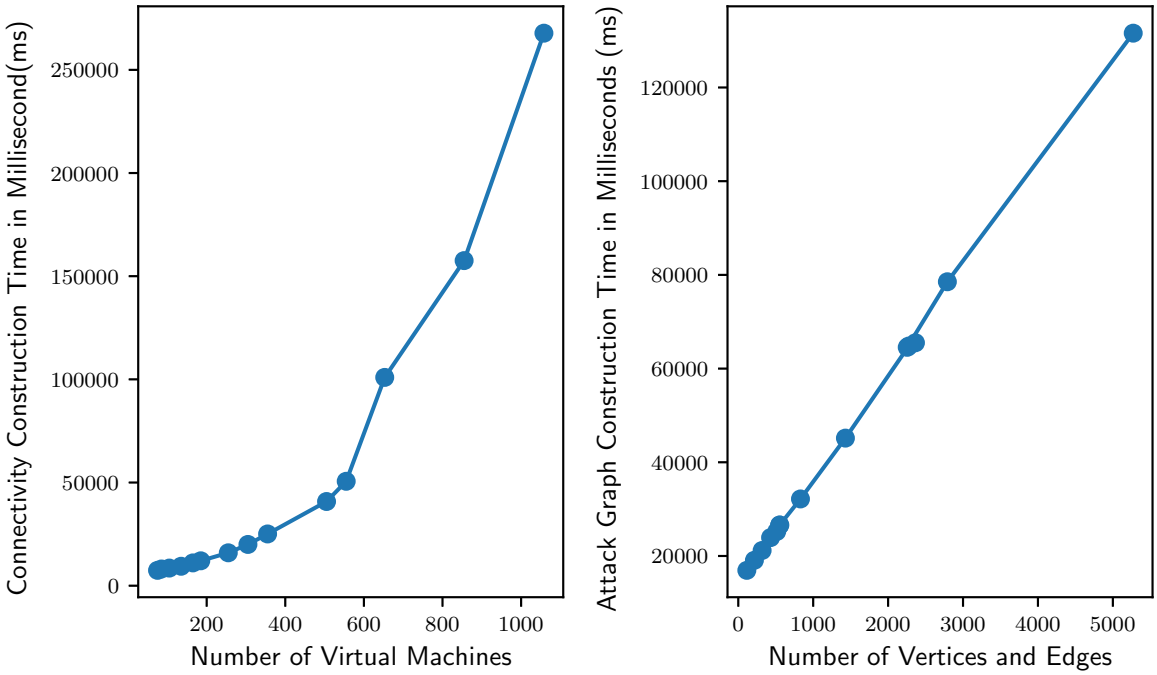


Figure 7.4: (Leftside) Connectivity construction time relative to the number of virtual machines - (Rightside) Attack graph construction time relative to the number of vertices and edges

static phase. Indeed, the topology and connectivity as well as the attack graph is not rebuilt from scratch at each event received, but merely updated with the delta incurred. It allows to save resources but also increases execution time.

7.4.3 Time performance of event-based processing

To illustrate this behaviour, we analyzed the processing performances upon receiving virtual machine creation, migration and deletion events, as well as vulnerability addition to and removal from a device, according to increasing workloads. The results obtained for the VMs creation events are showed on Figure 7.5. The remainder of the results can be found in Section A, on Figure A.1, Figure A.2, Figure A.3 and Figure A.4 respectively. In ordinates, we represent the event processing time, while in abscissa we represent the index of the generated event. As a reminder, we consecutively and randomly trigger 300 virtual machines related events, a hundred of each type in a non-deterministic order. These events are also randomly assigned to the virtual machines created in the infrastructure. We proceed similarly for the vulnerability related event, which results in 200 modification events, equally distributed in vulnerability addition and removal events. Similarly, these events are randomly assigned to the virtual machines created in the infrastructure, but also to the hypervisors.

We notice that depending on the interconnection of the device affected by the event received

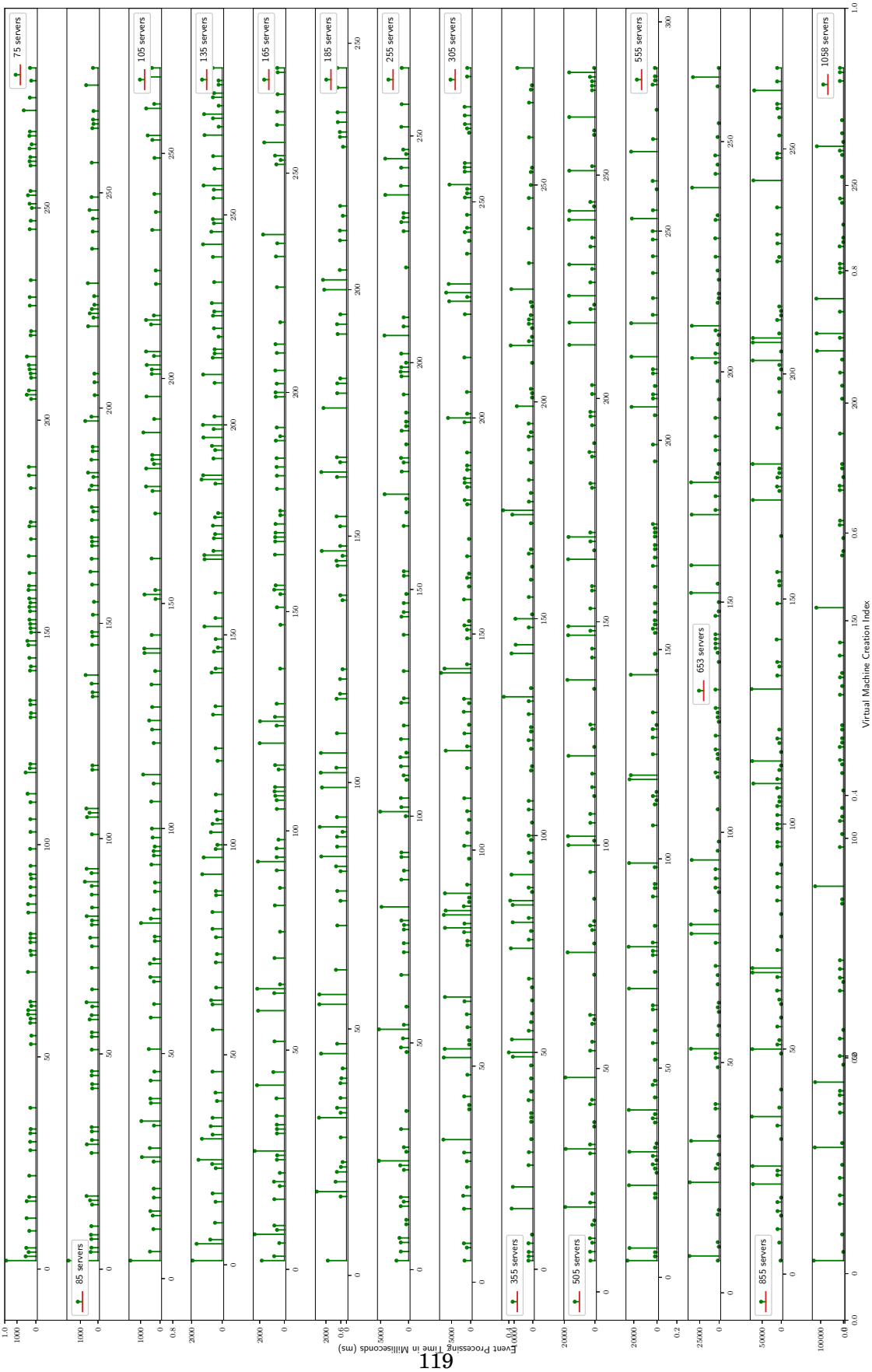


Figure 7.5: Processing time for virtual machine creation events according to various workloads

with its neighbors, the processing time can differ drastically, especially for the virtual machine creation events. Those events are also the most time consuming. This result is expected since such events affect both dimensions of our algorithm: the connectivity and the attack graph. They first require to determine the resulting connectivity, then to analyze all the vulnerabilities existing on all the devices involved in either an ingoing or outgoing connectivity with the concerned VM, as well as the potential co-located virtual machines. As a result, they take longer to process, compared with a vulnerability addition event which has no impact on the connectivity and will only affect the attack graph.

On Figure 7.6, we represent the minimum, maximum and mean processing time values depending on the workload and the type of event. We represent in ordinates the event processing time in milliseconds, and in abscissa the number of VMs obtained at the end of the static phase, before triggering the modification events.

The minimum and maximum values are obtained by comparing all the processing times obtained for a specific event type, given a specific workload, and recording the minimum and maximum processing time. The mean value is obtained by summing the processing times obtained for a specific event type, and dividing it by the number of generated events of that type.

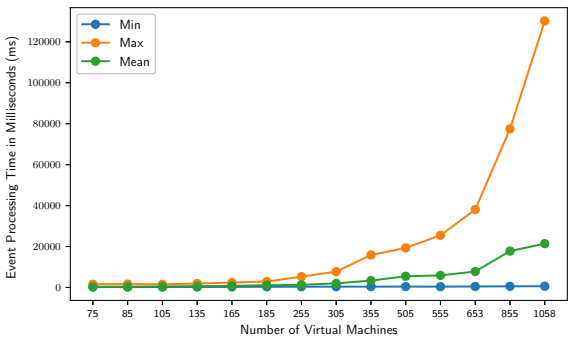
We can conclude from these plots that, on average, the processing time of an event is way inferior to the maximum value. For any event, the maximum processing time value needed to have an updated attack graph is always inferior to the sum of the topology and connectivity builder and the attack graph construction. In our experiments, and in the worst case scenario, we can have a 3.7 factor of gain in the processing time for the larger workload of 1058 virtual machines. This factor can go up to 13 for lower workloads (75 virtual machines). This illustrates the benefit of adopting an event-based approach.

Having such values represented for different workloads and different type of events allows us to estimate the mean interval required before the last event in the algorithm messaging queue will be processed. Considering a situation in which a lot of events would arrive at once as input of the algorithm, we are thus able to provide an indicator to the administrator hinting the time interval in which he can expect to have an up-to-date attack graph.

Let us consider the following parameters:

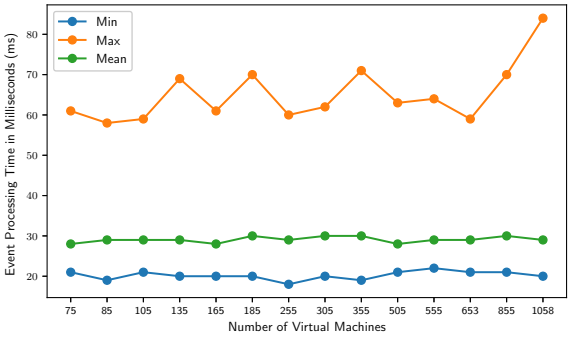
- S, the set of events in the messaging queue,
- N_{Events}, the number of events in the messaging queue,
- x_i, an event of type $event_i$ in the queue,
- δ_{x_i}, the average processing time of x_i ,
- E_{event_i}, an event of type $event_i$ newly appended to the messaging queue.

The time required to process the event E_{event_i} is:



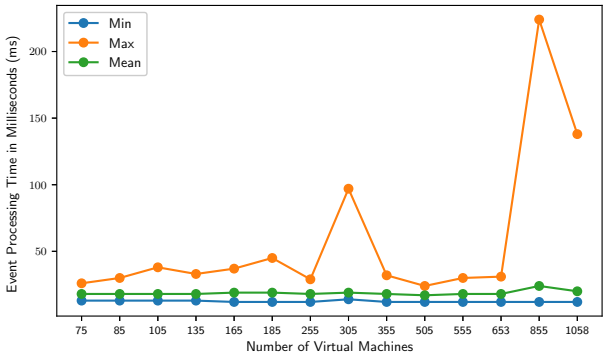
a

Virtual Machine Creation Events



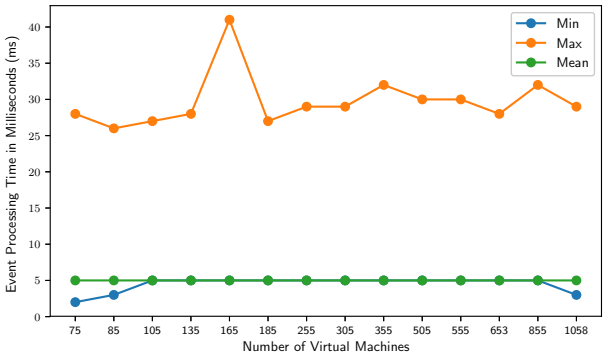
b

Virtual Machine Migration Events



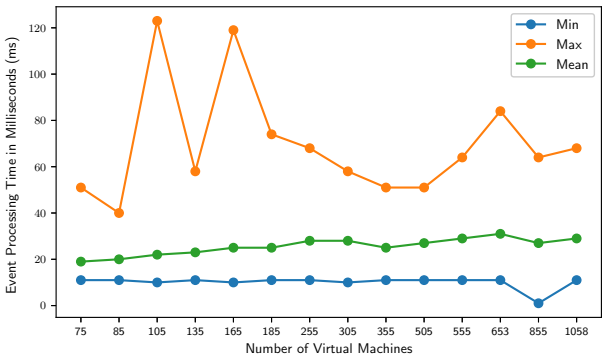
c

Virtual Machine Deletion Events



d

Vulnerability Addition Events



e

Vulnerability Removal Events

Figure 7.6: Minimum, maximum and mean values for various events type depending on the workload

$$\Delta E_{event_i} = \sum_{x_i \in S} \frac{\delta_{x_i}}{N_{Events}}$$

For instance, if we consider the failure of a number of physical servers, this scenario would

necessitate the migration all the virtual machines hosted on these hypervisors. As a result, several migration notification events would be generated in the infrastructure, rendering the previously generated attack graph obsolete. An administrator choosing this period to request the attack graph would be provided with a graph in an unstable state, due to the incremental processing of the migration events in the queue. The formula presented earlier would allow to provide the administrator with the countdown to getting a steady attack graph, depending on the current load.

In summary, we presented in this section the experiment platform as well as the prototype designed during this thesis. Using a real environment, at scale, we evaluated the feasibility of our approach, as well as the efficiency of the algorithms deployed to address the challenge of attack graph generation in the Cloud context. This first prototype allowed us to confront theory and practice, and provided valuable insights into the algorithm behavior for its different phases and for different events processing.

An additional challenge emerged to implement this approach, which is the necessity for existing vulnerability scanners to provide notifications upon vulnerability status changes in the infrastructure. These notifications were simulated in the experiments performed based on what we identified as useful messages that should be generated. In a real Cloud environment, the vulnerability scanners should be extended to send these notifications directly, similarly to the Cloud Management Platform.

CHAPTER 8

CONCLUSION

8.1 Conclusion

The Cloud succeeded in providing companies and individual users with a high level of flexibility and mobility for the deployment and usage of IT resources and applications. Resource allocation, resource provisioning, Cloud analytics and tenant billing have been automated to provide an ecosystem that is more and more prevalent and attractive. However, an area that remains to be automated is the security.

On one hand, companies externalizing their infrastructures to the Cloud may lack in-house resources for securing their virtualized platforms. On the other hand, it is impossible for human operators to manually address the volume of alerts potentially generated in a large scale environment such as the Cloud. This thesis is our contribution to the automation of Cloud environment security, by using attack graphs which can be leveraged in a risk management methodology. They not only represent a tool for an administrator to have more insights into the security of his infrastructure, they are also beneficial in identifying critical vulnerabilities to address to reduce the risks, based on economical or operational constraints.

As presented in the problem statement in Section 1.3, several challenges needed to be tackled for that purpose. In a first step, we identified the novelty of the Cloud environment as a source of concerns, since the introduction of the hypervisor represents a novel attack surface. As determined in the state of the art, existing attack graph methods did not really include the virtualized dimension in the vulnerability templates considered for attack graph generation. In order to address this knowledge gap and provide an accurate view of the vulnerabilities existing in an environment such as the Cloud, we performed an exhaustive analysis of virtualization vulnerabilities present on real-life hypervisors. This study allowed to uncover the novel attack

vectors that can be leveraged by an attacker, as well as their requirements to be successfully exploited. By using this knowledge, we can extend existing attack graph generation models to take into account the virtualized layer. Thus, we can consider both traditional and virtualized attack actions available to an attacker in a Cloud environment, leading to a more precise attack graph.

In a second step, we addressed the issue of building an attack graph that could be automatically generated and updated, taking into account the dynamic nature of the Cloud as well as its scale. To prove the viability of our approach, we strove to provide algorithms ranging from data collection to graph generation.

Existing methods often suppose that data is provided by the administrators, without detailing the way it is obtained. However as presented in the state of the art, obtaining data is not always a straightforward process, fact which can significantly hinder the adoption of attack graphs for risk management. To that end, we focused on one hand on the retrieval of the topology and connectivity of the infrastructure, a critical input for attack graph generation. Taking into account the opportunities provided by the available technologies, this was realized by using data collected from the Cloud Management Platform and the SDN controller, and by tracking changes occurring in the infrastructure in order to timely update the topology and connectivity initially built.

In addition, we proposed a hybrid attack graph model based on both states- and host-based models. Adopting this modeling approach for attack graph representation is a preparatory step to the graph analysis using graph theory algorithms. This should allow us to benefit from these algorithms' performances, with the end goal of improving the risk analysis phase of the risk management methodology.

With these foundations, namely an understanding of virtualization vulnerabilities, an up-to-date topology and connectivity view of the infrastructure, as well as an attack graph model suited to the Cloud, we designed an event-based algorithm for attack graph construction. It is able to provide Cloud security administrators with the current security exposure of their infrastructure relying on an attack graph regularly updated at each occurring event. The prototype designed to evaluate our proposal in a real life environment with plausible workloads presents promising performances and confirms the benefits and feasibility of our approach.

8.2 Limitations

We present in the following section some of the limitations incurred by the methods presented in this thesis.

We have proposed a fully automated approach, with no intervention from the customers of the infrastructure. As we adopt the standpoint of the Cloud administrator, we only rely on data directly available to him in various management databases. As a consequence, we build our attack graph according to the configured infrastructure, and do not identify any divergence between the

infrastructure a customer wishes to configure and the one that he effectively configures. Hence we do not identify human errors leading to the weakening of the infrastructure security state.

Besides, since the tenants can deploy any software in their virtual machines, especially in IaaS Clouds, nothing prevents them from configuring software firewalls within their virtual machines to block additional traffic directly at the VM level and not the Cloud provider network level. This means that more communication links would be detected using our connectivity reconstruction algorithm. This is however tolerable in a risk management context, since this corresponds to a worst case scenario in which we would identify more attack paths than effectively feasible in the infrastructure.

The method developed during this thesis is oriented toward a pro-active treatment of the vulnerabilities existing in the infrastructure. Given a particular configuration, the attack graph generated presents all possibilities available to an attacker for compromising the network, allowing the administrators to address the weaknesses before they can be leveraged in an attack. This approach is based on known vulnerabilities existing in the network, and does not take into account zero-days. In addition, ongoing attack scenarios calling for reactive countermeasures are also not considered by this model. However, if these ongoing attack scenarios rely on known vulnerabilities, they are listed in the attack paths existing in the attack graph, and could be addressed pro-actively.

8.3 Perspectives

A perspective to improve the work proposed during this thesis is to put real efforts in vulnerability description. During prototyping, we selected a number of vulnerabilities that we analyzed manually to extract their pre- and post- conditions on the infrastructure. This is due to the fact that, oftentimes, the vulnerabilities are described in a text. However this is not sustainable in the long run, and to be able to leverage the attack graph approach, we would need a way to extract the pre-condition and post-condition fields of the vulnerability, without resorting to a human operator. This could be done by improving the current vulnerability report process to include an enumeration of the conditions and consequences, that could be directly accessed through an API. Another avenue would be to leverage natural language processing methods to extract those fields from the existing descriptions, without having to modify the current vulnerability reporting process. This would allow to eliminate the need for security experts for specifying the pre- and post- conditions of any new vulnerability discovered in the infrastructure, hence accelerating the consideration of new attack actions in the attack graph generation, resulting in more accurate graphs.

Continuing on the topic of vulnerability, their inventory should also be obtained automatically in a manner that is transparent to the client. Indeed in this work, we focused on the connectivity reconstruction, but the vulnerabilities also represent a critical input for the attack graph gen-

eration. This inventory can be gathered using different approaches. The Cloud providers can either constrain their users to the use of specific virtual machine images already configured with vulnerability scanners or leave it to their clients to provide such information, using the methods they want in their customized virtual machines. Both options are impossible to generalize in real life, as they are dependent on the context and customers' demands. For instance, customers may have legacy applications needing specific operating systems and software versions not provided by off-the-shelf images maintained by the providers. On the other hand, regulatory compliance might constrain them to a certain level of privacy incompatible with an outsourced processing of their data. While with off-the-shelf images, providers impact the flexibility of the deployment, by giving all freedom to the users, we suffer either the issue of knowledge gap if the tenants do not provide the data, or the issue of freshness of information, which would lead to an inaccurate attack graph. An approach to tackle this issue, would be to investigate the possibility to implement virtual machine introspection in the Cloud infrastructure, which would permit from the outside of the virtual machine, to have an insight on the running services and their versions, and make the parallel with existing vulnerability databases to deduce the potential vulnerabilities in each virtual device. The main challenge would be the semantic gap existing between information that can be retrieved from within the virtual machine and from outside the virtual machine. Besides, an introspection mechanism would also not be applicable in all cases or to all tenants, and might be subject to contractual restrictions.

A longer term perspective is to actually leverage this attack graph for enhancing the security of the infrastructure. This entails determining the most relevant security questions for an administrator, that can be answered using an attack graph, and identifying which graph theory algorithm can be used to answer them: for instance, a security administrator might be interested in knowing the shortest path to a target or the minimum amount of paths that should be cut in order to prevent an attacker from reaching an asset. These algorithms can rely on optimized data structures to speed up the analysis process. The challenge in that context is identifying how these structures can be maintained considering the frequent changes of the attack graph. The next step with the result of this analysis, is to determine how the countermeasures can be deployed in the infrastructure in a way that balances at the same time the cost of the countermeasures, but also the risks incurred by the vulnerabilities and the ones caused by the countermeasures themselves. For instance, identifying precisely the conditions of virtualization vulnerabilities allows to identify the security exposure caused by a given resource allocation configuration. By performing resource allocation in a security-aware manner, we could react proactively by re-affecting the virtual machines to safer hosts and neighbors, but this would require to take into account, for instance, costs potentially generated by lack of physical server consolidation or increased network bandwidth for communicating between VMs that cannot be co-located.

Besides, the automated deployment of these countermeasures is also a requirement for an end-to-end automated solution. We provided the basis for this automation by delegating the

network provisioning to the SDN controller in our infrastructure. We envision to leverage the programmability enabled by the SDN paradigm for countermeasures deployment. Indeed, the objective would be to implement an SDN application responsible for enforcing the countermeasures proposals received as input from the attack graph analysis. In addition, the Cloud might present additional type of countermeasures, compared with traditional environments, enabling more flexibility in the mitigation of the vulnerabilities.

Appendix A

Figure A.1: Processing time for virtual machine migration events according to various workloads

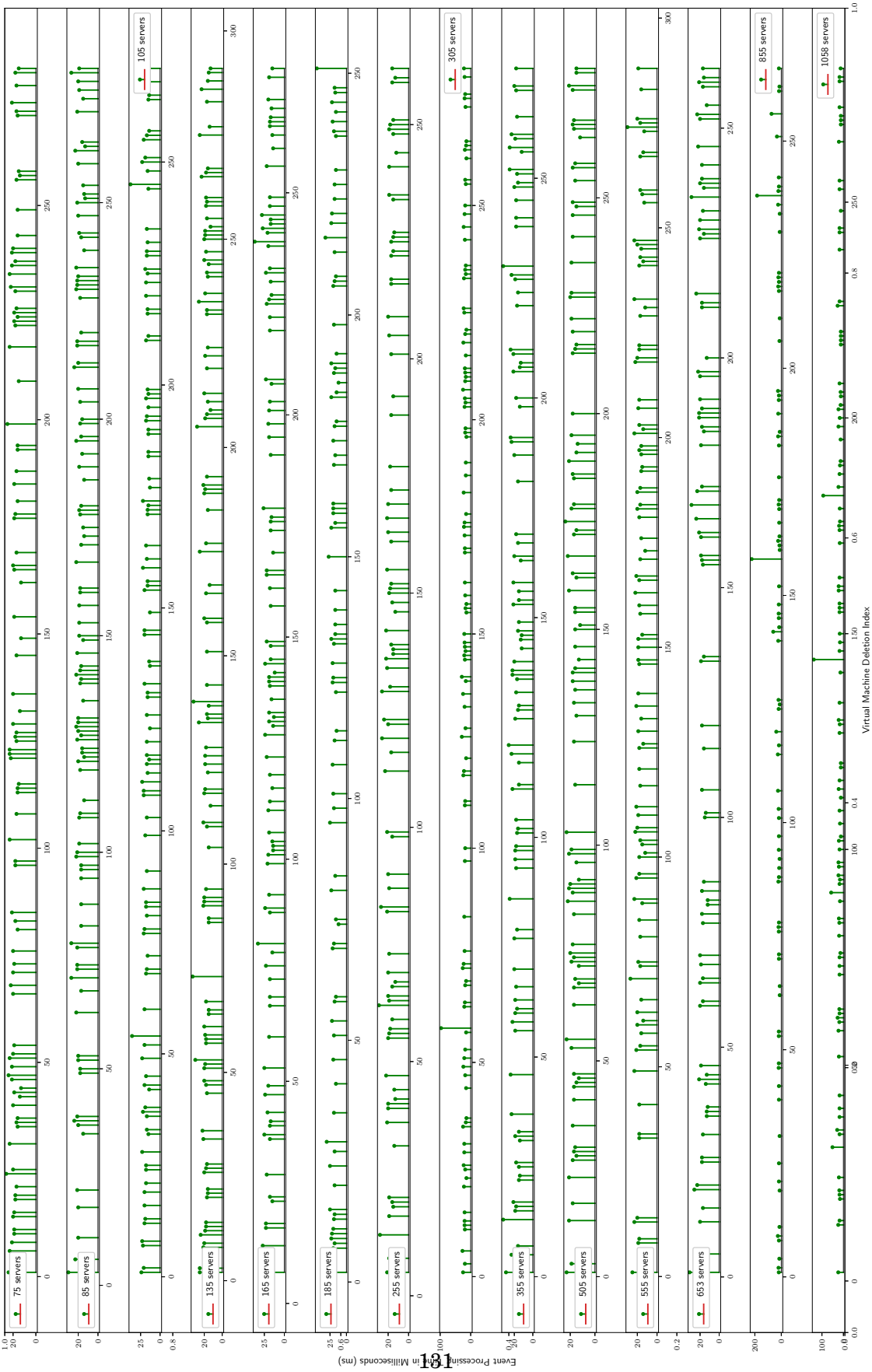


Figure A.2: Processing time for virtual machine deletion events according to various workloads

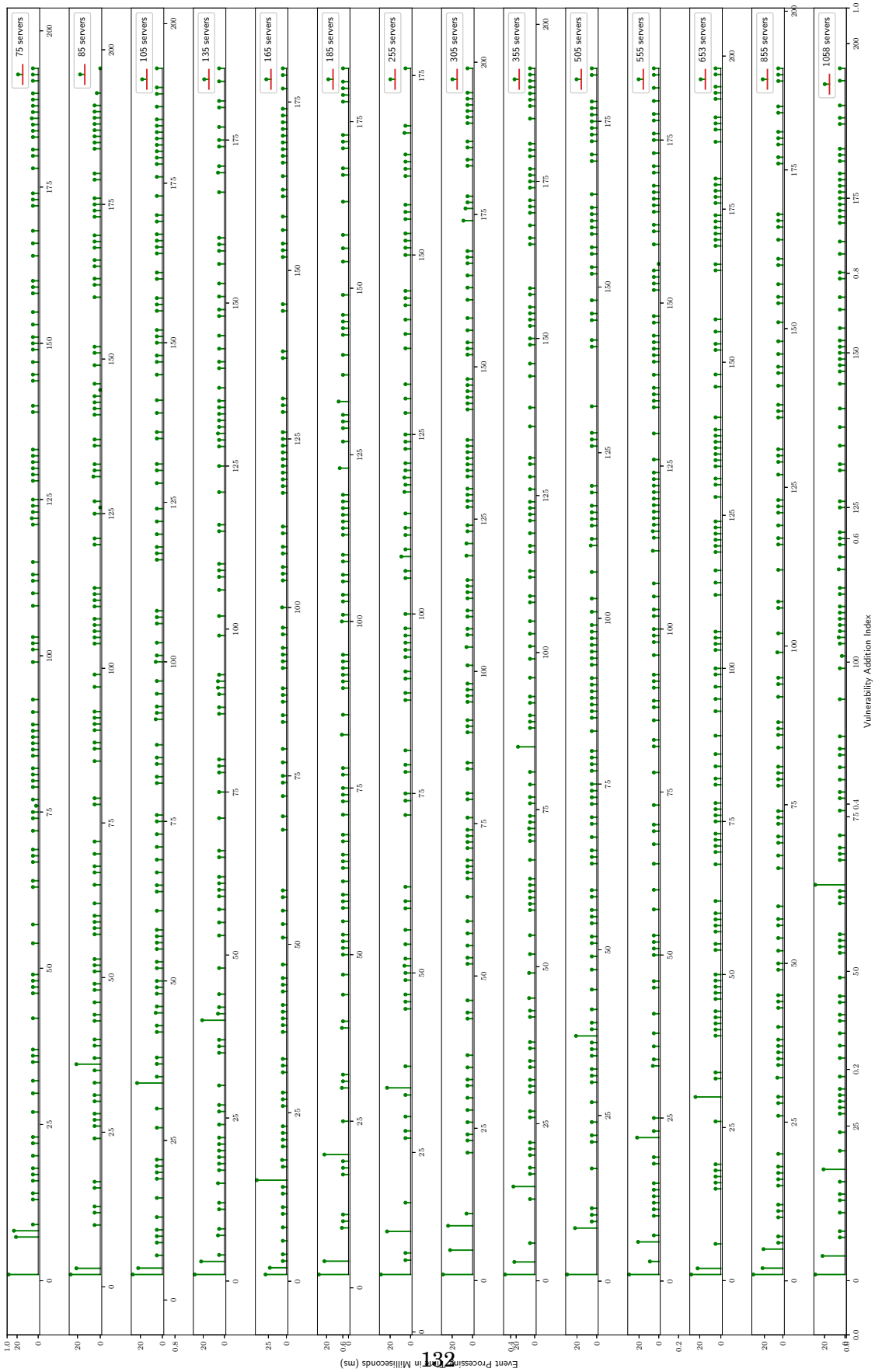


Figure A.3: Processing time for vulnerability addition events according to various workloads

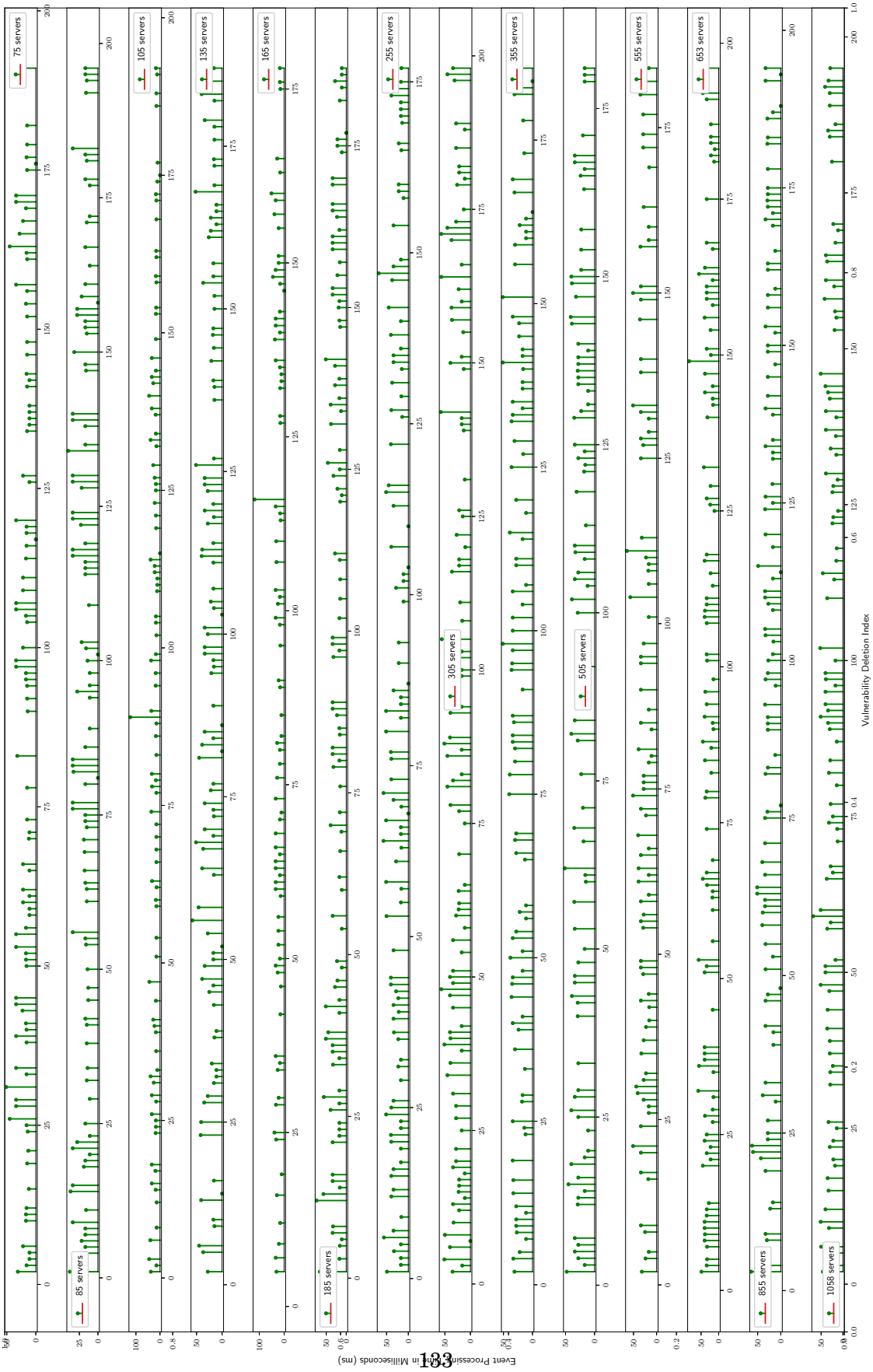


Figure A.4: Processing time for vulnerability removal events according to various workloads

BIBLIOGRAPHY

[1] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proceedings of the 1998 workshop on New security paradigms*, pp. 71–79, ACM, 1998.

[2] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-attack graph generation tool," in *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, vol. 2, pp. 307–321, IEEE, 2001.

[3] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pp. 273–284, IEEE, 2002.

[4] O. Sheyner and J. Wing, "Tools for generating and analyzing attack graphs," in *International Symposium on Formal Methods for Components and Objects*, pp. 344–371, Springer, 2003.

[5] P. Ammann, J. Pamula, R. Ritchey, and J. Street, "A host-based approach to network attack chaining analysis," in *Computer Security Applications Conference, 21st Annual*, pp. 10–pp, IEEE, 2005.

[6] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 217–224, ACM, 2002.

[7] S. Jajodia, S. Noel, and B. O'Berry, "Topological analysis of network attack vulnerability," in *Managing Cyber Threats*, pp. 247–266, Springer, 2005.

[8] K. Ingols, R. Lippmann, and K. Piwowarski, "Practical attack graph generation for network defense," in *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pp. 121–130, IEEE, 2006.

[9] "SDN Architecture - Open Networking Foundation." `https://www.opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf`.
Accessed 2nd April 2019.

[10] "Cloud management platform architecture." http://crmtrilogix.com/Cloud-Blog/Cloud-Models/Cloud-Management-Platform-Architecture/180.
Accessed 2nd April 2019.

[11] "Specification, OpenFlow Switch, V1. 3.1," tech. rep., Open Networking Foundation, 2012.

[12] "Specification, OpenFlow Switch, V1. 5.1," tech. rep., Open Networking Foundation, 2015.

[13] "AWS template - asynchronous online gaming." https://d1.awsstatic.com/architecture-diagrams/Asynchronous-Online-Gaming-Basic.pdf.
Accessed 2nd April 2019.

[14] Gartner, "Gartner revenue prediction 2018." https://www.gartner.com/en/newsroom/press-releases/2018-04-12-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-21-percent-in-2018.
Accessed 5th February 2019.

[15] Gartner, "Gartner revenue prediction 2017." https://www.gartner.com/en/newsroom/press-releases/2017-10-12-gartner-forecasts-worldwide-public-cloud-services-revenue-to-reach-260-billionin-2017.
Accessed 5th February 2019.

[16] T. Garfinkel and M. Rosenblum, "When virtual is harder than real: Security challenges in virtual machine based computing environments.," in *HotOS*, 2005.

[17] K. Hashizume, D. G. Rosado, E. Fernández-Medina, and E. B. Fernandez, "An analysis of security issues for cloud computing," *Journal of internet services and applications*, vol. 4, no. 1, p. 5, 2013.

[18] O. AbdElRahem, A. M. Bahaa-Eldin, and A. Taha, "Virtualization security: A survey," in *Computer Engineering & Systems (ICCES), 2016 11th International Conference on*, pp. 32–40, IEEE, 2016.

[19] J. Oberheide, E. Cooke, and F. Jahanian, "Exploiting live virtual machine migration," *BlackHat DC Briefings*, 2008.

[20] "Survey report: Behind the growing confidence in cloud security," tech. rep., MIT SMR Custom Studio, 2017.

[21] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *ArXiv e-prints*, Jan. 2018.

[22] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *ArXiv e-prints*, Jan. 2018.

[23] K. Kortchinsky, "Cloudburst: A vmware guest to host escape story," *Black Hat USA*, vol. 19, 2009.

[24] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using bayesian attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2012.

[25] S. Noel, S. Jajodia, L. Wang, and A. Singhal, "Measuring security risk of networks using attack graphs," *International Journal of Next-Generation Computing*, vol. 1, no. 1, pp. 135–147, 2010.

[26] L. Wang, S. Jajodia, A. Singhal, and S. Noel, "k-zero day safety: Measuring the security risk of networks against unknown attacks," in *European Symposium on Research in Computer Security*, pp. 573–587, Springer, 2010.

[27] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*, pp. 49–63, IEEE, 2002.

[28] S. Noel and S. Jajodia, "Optimal ids sensor placement and alert prioritization using attack graphs," *Journal of Network and Systems Management*, vol. 16, no. 3, pp. 259–275, 2008.

[29] C. Liu, A. Singhal, and D. Wijesekera, "Using attack graphs in forensic examinations," in *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*, pp. 596–603, IEEE, 2012.

[30] W. Kanoun, S. Papillon, and S. Dubus, "Elementary risks: Bridging operational and strategic security realms," in *11th International Conference on Signal-Image Technology & Internet-Based Systems, SITIS 2015, Bangkok, Thailand, November 23-27, 2015*, pp. 278–286, 2015.

[31] T. Garfinkel, M. Rosenblum, *et al.*, "A virtual machine introspection based architecture for intrusion detection.," in *Ndss*, vol. 3, pp. 191–206, 2003.

[32] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee, "Virtuoso: Narrowing the semantic gap in virtual machine introspection," in *Security and Privacy (SP), 2011 IEEE Symposium on*, pp. 297–312, IEEE, 2011.

[33] B. D. Payne, "Simplifying virtual machine introspection using libvmi," *Sandia report*, pp. 43–44, 2012.

[34] S. Rigas, G. Debrosses, K. Haralampidis, F. Vicente-Angulo, K. A. Feldman, A. Grabov, L. Dolan, and P. Hatzpoulos, "Trh1 encondes a potassium transporter required for tip growth in arabidopsis root hairs," *The Plant Cell*, vol. 13, pp. 139–151, 2001.

[35] R. Motamedi, R. Rejaie, and W. Willinger, "A survey of techniques for internet topology discovery," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 1044–1065, 2015.

[36] B. Donnet and T. Friedman, "Internet topology discovery: a survey," *IEEE Communications Surveys and Tutorials*, vol. 9, no. 4, pp. 2–15, 2007.

[37] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-level traffic measurements from the sprint IP backbone," *IEEE Network*, vol. 17, pp. 6–16, Nov. 2003.

[38] W. Tu, P. Thangaraj, J.-h. Chiang, and T.-c. Chiueh, "Automated service discovery for enterprise network management," 2009.

[39] B. Eriksson, P. Barford, and R. Nowak, "Network Discovery from Passive Measurements," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, (New York, NY, USA), pp. 291–302, ACM, 2008.

[40] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*.
Addison-Wesley Longman Publishing Co., Inc., 1998.

[41] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz, "Topology discovery in heterogeneous IP networks: the NetInventory system," *IEEE/ACM Transactions on Networking*, vol. 12, pp. 401–414, June 2004.

[42] B. Lowekamp, D. O'Hallaron, and T. Gross, "Topology Discovery for Large Ethernet Networks," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, (New York, NY, USA), pp. 237–248, ACM, 2001.

[43] B. T. Nassu, T. Nanya, and E. P. Duarte, "Topology discovery in dynamic and decentralized networks with mobile agents and swarm intelligence," in *Intelligent Systems Design and Applications, 2007. ISDA 2007. Seventh International Conference on*, pp. 685–690, IEEE, 2007.

[44] X. Jin, W. Yiu, G. S. H. Chan, and Y. Wang, "Network topology inference based on end-to-end measurements," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2182–2195, 2006.

[45] X. Jin, Y. Wang, and S.-H. Chan, "Fast overlay tree based on efficient end-to-end measurements," in *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, vol. 2, pp. 1319–1323, IEEE, 2005.

[46] V. Jacobson, "Traceroute software," *Lawrence Berkeley Laboratories*, 1988.

[47] R. Siamwalla, R. Sharma, and S. Keshav, "Discovering internet topology," *Unpublished manuscript*, 1998.

[48] R. Bush, O. Maennel, M. Roughan, and S. Uhlig, "Internet optometry: assessing the broken glasses in internet reachability," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pp. 242–253, ACM, 2009.

[49] R. Beverly, A. Berger, and G. G. Xie, "Primitives for active internet topology mapping: Toward high-frequency characterization," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 165–171, ACM, 2010.

[50] D. McRobb, K. Claffy, and T. Monk, "Skitter: Caida's macroscopic internet topology discovery and tracking tool, 1999."

[51] F. Georgatos, F. Gruber, D. Karrenberg, M. Santcroos, A. Susanj, H. Uijterwaal, and R. Wilhelm, "Providing active measurements as a regular service for isps," in *PAM*, 2001.

[52] Y. Shavitt and E. Shir, "Dimes: Let the internet measure itself," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 71–74, 2005.

[53] B. Donnet, P. Raoult, T. Friedman, and M. Crovella, "Efficient algorithms for large-scale topology discovery," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, pp. 327–338, ACM, 2005.

[54] B. Donnet, T. Friedman, and M. Crovella, "Improved algorithms for network topology discovery," in *International Workshop on Passive and Active Network Measurement*, pp. 149–162, Springer, 2005.

[55] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 336–345, ACM, 2006.

[56] X. Ou, S. Govindavajhala, and A. W. Appel, "Mulval: A logic-based network security analyzer.," in *USENIX Security Symposium*, pp. 8–8, Baltimore, MD, 2005.

[57] S. Zhong, D. Yan, and C. Liu, "Automatic generation of host-based network attack graph," in *Computer Science and Information Engineering, 2009 WRI World Congress on*, vol. 1, pp. 93–98, IEEE, 2009.

[58] M. Dacier and Y. Deswarte, "Privilege graph: an extension to the typed access matrix model," in *European Symposium on Research in Computer Security*, pp. 319–334, Springer, 1994.

[59] M. Dacier, Y. Deswarte, and M. Kaâniche, "Quantitative assessment of operational security: Models and tools," *Information Systems Security, ed. by SK Katsikas and D. Gritzalis, London, Chapman & Hall*, pp. 179–86, 1996.

[60] S. Jajodia and S. Noel, "Topological vulnerability analysis: A powerful new approach for network attack prevention, detection, and response," in *Algorithms, architectures and information systems security*, pp. 285–305, World Scientific, 2009.

[61] S. Jajodia and S. Noel, "Topological vulnerability analysis," in *Cyber situational awareness*, pp. 139–154, Springer, 2010.

[62] M. L. Artz, *Netspa: A network security planning architecture*.
PhD thesis, Massachusetts Institute of Technology, 2002.

[63] R. W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pp. 156–165, IEEE, 2000.

[64] C. Ramakrishnan and R. Sekar, "Model-based analysis of configuration vulnerabilities 1," *Journal of Computer Security*, vol. 10, no. 1-2, pp. 189–209, 2002.

[65] M. S. Barik, A. Sengupta, and C. Mazumdar, "Attack graph generation and analysis techniques," *Defence Science Journal*, vol. 66, no. 6, p. 559, 2016.

[66] J. T. W. Jing, L. W. Yong, D. M. Divakaran, and V. L. Thing, "Augmenting mulval with automated extraction of vulnerabilities descriptions," in *Region 10 Conference, TENCON 2017-2017 IEEE*, pp. 476–481, IEEE, 2017.

[67] P. Rao, K. Sagonas, T. Swift, D. S. Warren, and J. Freire, "Xsb: A system for efficiently computing well-founded semantics," in *International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 430–440, Springer, 1997.

[68] A. Colmerauer, "An introduction to prolog iii," in *Computational Logic*, pp. 37–79, Springer, 1990.

[69] R. Lippmann, K. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, and R. Cunningham, "Validating and restoring defense in depth using attack graphs," 2006.

[70] NIST, "https://nvd.nist.gov/."
Accessed 20th May 2019.

[71] "Responsibilities in the cloud - microsoft developer website." `https://blogs.msdn.microsoft.com/azuresecurity/2016/04/18/what-does-shared-responsibility-in-the-cloud-mean/`.
Accessed 2nd April 2019.

[72] R. J. Creasy, "The origin of the vm/370 time-sharing system," *IBM Journal of Research and Development*, vol. 25, no. 5, pp. 483–490, 1981.

[73] R. A. Meyer and L. H. Seawright, "A virtual machine time-sharing system," *IBM Systems Journal*, vol. 9, no. 3, pp. 199–218, 1970.

[74] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SIGOPS operating systems review*, vol. 37, pp. 164–177, ACM, 2003.

[75] A. Wang, M. Iyer, R. Dutta, G. N. Rouskas, and I. Baldine, "Network virtualization: Technologies, perspectives, and frontiers," *Journal of Lightwave Technology*, vol. 31, no. 4, pp. 523–537, 2013.

[76] J. Moy, "Rfc 7348: Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks," *Tech. Rep.*, 2014.

[77] P. Garg and Y. Wang, "Nvgre: Network virtualization using generic routing encapsulation," tech. rep., 2015.

[78] J. Gross, T. Sridhar, P. Garg, C. Wright, I. Ganga, P. Agarwal, K. Duda, D. Dutt, and J. Hudson, "Geneve: Generic network virtualization encapsulation," *IETF draft*, 2014.

[79] "SNIA technical tutorial - storage virtualization," tech. rep., Storage Networking Industry Association (SNIA).
Accessed 20th May 2019.

[80] R. Sherwood and N. Spring, "Touring the internet in a tcp sidecar," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pp. 339–344, ACM, 2006.

[81] B. Donnet, P. Raoult, T. Friedman, and M. Crovella, "Deployment of an algorithm for large-scale topology discovery," *IEEE journal on selected areas in communications*, vol. 24, no. 12, pp. 2210–2220, 2006.

[82] B. Fabian, A. Baumann, and J. Lackner, "Topological analysis of cloud service connectivity," *Computers and Industrial Engineering*, vol. Volume 88, pp. 151–165, 10 2015.

[83] S. Paredes, N. N. El-Aawar, G. R. Ratterree, T. J. Williamson, and T. Wagner, "Virtualized connectivity in a cloud services environment," June 14 2012.
US Patent App. 13/310,589.

[84] L. Velasco, A. Asensio, J. L. Berral, A. Castro, and V. López, "Towards a carrier sdn: An example for elastic inter-datacenter connectivity," *Optics express*, vol. 22, no. 1, pp. 55–61, 2014.

[85] S. Chen, S. Nepal, and R. Liu, "Secure connectivity for intra-cloud and inter-cloud communication," in *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on*, pp. 154–159, IEEE, 2011.

[86] T. Mundt and J. Vetterick, "Network topology analysis in the cloud," in *Proceedings on the International Conference on Internet Computing (ICOMP)*, p. 1, The Steering Committee of The World Congress in Computer Science, Computer . . . , 2011.

[87] T. Madi, S. Majumdar, Y. Wang, M. Pourzandi, and L. Wang, "Auditing security Compliance of the Virtualized Infrastructure in the Cloud: Application to OpenStack," in *6th ACM Conference on Data and Application Security and Privacy ACM CODASPY 2016*, 2016.

[88] S. Bleikertz, C. Vogel, and T. Gross, "Cloud Radar: Near Real-Time Detection of Security Failures in Dynamic Virtualized Infrastructures," in *Annual Computer Security Applications Conference (ACSAC)*, 2014.

[89] S. Bleikertz, T. Gross, M. Schunter, and K. Eriksson, "Automated Information Flow Analysis of Virtualized Infrastructures," in *Proceedings of the 16th European Conference on Research in Computer Security*, ESORICS'11, (Berlin, Heidelberg), pp. 392–415, Springer-Verlag, 2011.

[90] P. Dinadayalan, S. Jegadeeswari, and D. Gnanambigai, "Data security issues in cloud environment and solutions," in *Computing and Communication Technologies (WCCCT), 2014 World Congress on*, pp. 88–91, IEEE, 2014.

[91] M. D. Ryan, "Cloud computing security: The scientific challenge, and a survey of solutions," *Journal of Systems and Software*, vol. 86, no. 9, pp. 2263–2268, 2013.

[92] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security & Privacy*, no. 6, pp. 24–31, 2010.

[93] M. Ali, S. U. Khan, and A. V. Vasilakos, "Security in cloud computing: Opportunities and challenges," *Information sciences*, vol. 305, pp. 357–383, 2015.

[94] N. C. Paxton, "Cloud security: a review of current issues and proposed solutions," in *Collaboration and Internet Computing (CIC), 2016 IEEE 2nd International Conference on*, pp. 452–455, IEEE, 2016.

[95] N. H. Ab Rahman and K.-K. R. Choo, "A survey of information security incident handling in the cloud," *Computers & Security*, vol. 49, pp. 45–69, 2015.

[96] N. Kheir, N. Cuppens-Boulahia, F. Cuppens, and H. Debar, "A service dependency model for cost-sensitive intrusion response," in *European Symposium on Research in Computer Security*, pp. 626–642, Springer, 2010.

[97] A. Shameli-Sendi, N. Ezzati-Jivan, M. Jabbarifar, and M. Dagenais, "Intrusion response systems: survey and taxonomy," *Int. J. Comput. Sci. Netw. Secur*, vol. 12, no. 1, pp. 1–14, 2012.

[98] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems," *IEEE transactions on dependable and secure computing*, vol. 10, no. 4, pp. 198–211, 2013.

[99] O. Mjihil, D. S. Kim, and A. Haqiq, "Masat: Model-based automated security assessment tool for cloud computing," in *Information Assurance and Security (IAS), 2015 11th International Conference on*, pp. 97–103, IEEE, 2015.

[100] N. Alhebaishi, L. Wang, S. Jajodia, and A. Singhal, "Threat modeling for cloud data center infrastructures," in *International Symposium on Foundations and Practice of Security*, pp. 302–319, Springer, 2016.

[101] S. Bleikertz, M. Schunter, C. W. Probst, D. Pendarakis, and K. Eriksson, "Security audits of multi-tier virtual infrastructures in public infrastructure clouds," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pp. 93–102, ACM, 2010.

[102] D. Perez-Botero, J. Szefer, and R. B. Lee, "Characterizing hypervisor vulnerabilities in cloud computing servers," in *Proceedings of the 2013 international workshop on Security in cloud computing*, pp. 3–10, ACM, 2013.

[103] S. T. King and P. M. Chen, "Subvirt: Implementing malware with virtual machines," in *Security and Privacy, 2006 IEEE Symposium on*, pp. 14–pp, IEEE, 2006.

[104] T. Katsuki, "Crisis: The advanced malware," *Internet security threat report-2013., Symantec Corporation*, vol. 18, 2012.

[105] I. Studnia, E. Alata, Y. Deswarte, M. Kaâniche, and V. Nicomette, "Survey of security problems in cloud computing virtual machines," in *Computer and Electronics Security Applications Rendez-vous (C&ESAR 2012). Cloud and security: threat or opportunity*, pp. p–61, 2012.

[106] G. Pék, L. Buttyán, and B. Bencsáth, "A survey of security issues in hardware virtualization," *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, p. 40, 2013.

[107] J. S. Reuben, "A survey on virtual machine security," *Helsinki University of Technology*, vol. 2, no. 36, 2007.

[108] P. Ferrie, "Attacks on more virtual machine emulators," *Symantec Technology Exchange*, vol. 55, 2007.

[109] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 199–212, ACM, 2009.

[110] A. Gkortzis, S. Rizou, and D. Spinellis, "An empirical analysis of vulnerabilities in virtualization technologies," in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*, pp. 533–538, IEEE, 2016.

[111] F. Zhang, J. Chen, H. Chen, and B. Zang, "Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 203–216, ACM, 2011.

[112] T. Probst, *Evaluation et analyse des mécanismes de sécurité des réseaux dans les infrastructures virtuelles de cloud computing*.
PhD thesis, INP Toulouse, 2015.

[113] P. Mensah, S. Dubus, W. Kanoun, C. Morin, G. Piolle, and E. Totel, "Connectivity graph reconstruction for networking cloud infrastructures," in *Network Computing and Applications (NCA), 2017 IEEE 16th International Symposium on*, pp. 1–9, IEEE, 2017.

[114] P. R. Halmos and A. H. S. P. Book, *Graduate Texts in Mathematics*.
Springer, 1971.

[115] J. Bang-Jensen and G. Z. Gutin, *Digraphs: theory, algorithms and applications*.
Springer Science & Business Media, 2008.

[116] Oracle and KPMG, "Oracle and KPMG Cloud Threat Report," tech. rep., Oracle and KPMG, 2018.

[117] X. Ou and A. Singhal, "Attack graph techniques," in *Quantitative Security Risk Assessment of Enterprise Networks*, pp. 5–8, Springer, 2012.

[118] L. Piètre-Cambacédès and M. Bouissou, "The promising potential of the bdmp formalism for security modeling," in *Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009), Supplemental Volume*, 2009.

[119] S. Kriaa, M. Bouissou, and L. Piètre-Cambacédès, "Modeling the stuxnet attack with bdmp: Towards more formal risk assessments," in *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*, pp. 1–8, IEEE, 2012.

[120] "Apache cloudstack." https://cloudstack.apache.org/index.html.
Accessed 2nd April 2019.

[121] "Eucalyptus." https://www.eucalyptus.cloud/.
Accessed 2nd April 2019.

[122] "ONOS." http://onosproject.org/.
Accessed 20th May 2019.

[123] "Opendaylight." https://www.opendaylight.org/.
Accessed 2nd April 2019.

[124] "Neo4j." https://www.neo4j.com/.
Accessed 20th May 2019.

[125] S. Jouili and V. Vansteenberghe, "An empirical comparison of graph databases," in *Social Computing (SocialCom), 2013 International Conference on*, pp. 708–715, IEEE, 2013.

[126] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, *et al.*, "Grid'5000: A large scale and highly reconfigurable experimental grid testbed," *The International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, 2006.

[127] RightScale, "State Of The Cloud Report," tech. rep., RightScale, 2017.