Security Auditing and Multi-Tenancy Threat Evaluation in Public Cloud Infrastructures

Taous Madi

A Thesis
in
The Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Information and Systems Engineering) at
Concordia University
Montréal, Québec, Canada

November 2018

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Taous Madi

Entitled: Security Auditing and Multi-Tenancy Threat Evaluation in Public Cloud Infrastructures

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy (Information and Systems Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. S. Samuel Li

_____ External Examiner
Dr. Nur Zincir-Heywood

_____ External to Program
Dr. Anjali Agarwal

_____ Examiner
Dr. Chadi Assi

_____ Examiner
Dr. Jun Yan

_____ Thesis Co-Supervisor
Dr. Mourad Debabbi

_____ Thesis Co-Supervisor
Dr. Lingyu Wang

Approved by _____
Dr. Chadi Assi, Graduate Program Director

February 26, 2019

_____
Dr. Amir Asif, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Security Auditing and Multi-Tenancy Threat Evaluation in Public Cloud Infrastructures

Taous Madi, Ph.D.

Concordia University, 2018

Cloud service providers typically adopt the multi-tenancy model to optimize resources usage and achieve the promised cost-effectiveness. However, multi-tenancy in the cloud is a double-edged sword. While it enables cost-effective resource sharing, it increases security risks for the hosted applications. Indeed, multiplexing virtual resources belonging to different tenants on the same physical substrate may lead to critical security concerns such as cross-tenant data leakage and denial of service. Therefore, there is an increased necessity and a pressing need to foster transparency and accountability in multi-tenant clouds. In this regard, auditing security compliance of the cloud provider's infrastructure against standards, regulations and customers' policies on one side, and evaluating the multi-tenancy threat on the other side, take on an increasing importance to boost the trust between the cloud stakeholders.

However, auditing virtual infrastructures is challenging due to the dynamic and layered nature of the cloud. Particularly, inconsistencies in network isolation mechanisms across the cloud stack layers (e.g., the infrastructure management layer and the implementation layer), may lead to virtual network isolation breaches that might be undetectable at a single layer. Additionally, evaluating multi-tenancy threats in the cloud requires systematic ways and effective metrics, which are largely missing in the literature. This thesis work addresses the aforementioned challenges and limitations and articulates around two main

topics, namely, security compliance auditing and multi-tenancy threat evaluation in the cloud.

Our objective in the first topic is to propose an automated framework that allows auditing the cloud infrastructure from the structural point of view, while focusing on virtualization-related security properties and consistency between multiple control layers. To this end, we devise a multi-layered model related to each cloud stack layer's view in order to capture the semantics of the audited data and its relation to consistent isolation requirements. Furthermore, we integrate our auditing system into OpenStack, and present our experimental results on assessing several properties related to virtual network isolation and consistency. Our results show that our approach can be successfully used to detect virtual network isolation breaches for large OpenStack-based data centers in a reasonable time.

The objective of the second topic is to derive security metrics for evaluating the multi-tenancy threats in public clouds. To this end, we propose security metrics to quantify the proximity between tenants' virtual resources inside the cloud. Those metrics are defined based on the configuration and deployment of a cloud, such that a cloud provider may apply them to evaluate and mitigate co-residency threats. To demonstrate the effectiveness of our metrics and show their usefulness, we conduct case studies based on both real and synthetic cloud data. We further perform extensive simulations using CloudSim and well-known VM placement policies. The results show that our metrics effectively capture the impact of potential attacks, and the abnormal degrees of co-residency between a victim and potential attackers, which paves the way for the design of effective mitigation solutions against co-residency attacks.

Acknowledgments

This work is the fruit of collaboration and support of many people, to whom I would like to express my sincere gratitude and appreciation.

First, I would like to express my deepest gratitude to Dr. Mourad Debbabi who opened up new and fascinating horizons in my life by giving me the opportunity to pursue my Ph.D. under his supervision. He has always enlightened me with wise advices and broadened my research perspectives thanks to his long experience and profound knowledge. Despite his very busy schedule, he has always dedicated time to listen to my worries and provide me with the much needed motivation, guidance and inspiration.

I would also like to express my sincere gratitude to my co-supervisor, Dr. Lingyu Wang, for all the efforts and the valuable time he spent for guiding me and sharpening my research skills. The continuous feedback emanating from his long experience and precise insight were key drivers to the achievement of this work. I feel fortunate that I had the chance to learn from him lifelong lessons of dedication, perseverance and self-determination.

I would like to thank the members of the examining committee: Dr. Nur Zincir-Heywood, Dr. Anjali Agarwal, Dr. Chadi Assi and Dr. Jun Yan, who honored me by accepting to evaluate this thesis. Their time and efforts are highly appreciated.

I would also like to thank Dr. Makan Pourzandi who closely followed the progress of this work and nourished it with his insightful suggestions and constructive criticism.

I am particularly grateful to Dr. Yosr Jarraya who was more than a colleague, more than a mentor, and more than a friend! So often, she had to bear with me and share my concerns

with a lot of patience and wisdom.

My special thanks are extended to my colleagues: Dr. Suryadipta Majumdar, Dr. Mengyuan Zhang, Yushun Wang, Amir Alimohammadifar, Gagandeep Singh Chawla, Meisam Mohammadi, Momen Oqaily, Azadeh Tabiban and Alaa Oqaily for their professionalism, kindness and support. I also acknowledge the financial support of NSERC, Ericsson Canada, Prompt Quebec and Concordia University.

No words can be enough to express my endless gratitude to my parents who have always given me unconditional love and support in every possible way and at every step. Their constant care and encouragement have sustained me through harsh circumstances and have gotten me through when I was about to give up. I also want to thank my brother for always being there for me.

Finally, I lovingly dedicate this dissertation to the glow of my eyes, my dear children, Ilham and Iyad. You have filled my life with so much happiness and made me stronger than I could have ever imagined. I love you till the moon and back!

Contents

List of Figures

List of Tables

Glossary

GRE A tunneling protocol developed by Cisco. It encapsulates a variety of protocol packet types inside IP tunnels to create a virtual point-to-point link over an IP network. 63

Layer 2 It is the second layer (Layer 2) of the well-known and standardized Open Systems Interconnection (OSI) model of computer networking. MAC addresses are used to identify networking devices that are at the same layer 2, which can reach each other by traffic broadcasting. 55, 56

Network Segments Isolated broadcast domains within a network. 54

Open vSwitch Open-source software switch implementation designed to be used in hypervisors to provide connectivity to guest virtual machines (VMs). 58

OpenStack It is an open-source cloud infrastructure management platform that is being used almost in half of private clouds and significant portions of the public clouds (see [1] for detailed statistics). 54, 57, 58

Overlay Networks Virtual networks that create a virtual topology on top of the physical network [2]. In our context, it is the cloud provider's physical network. They use overlay protocols such as VXLAN and GRE to provide scalable network isolation. 56

TPM Stands for Tursted Platform Module. It is a standard [3] for dedicated microcontrollers endowed with cryptographic keys to secure the hardware. 60

Virtual Networks Dedicated communication networks providing connectivity to a set of VMs possibly distributed over multiple hosts. Virtual networks share the same physical substrate and are logically segregated through network virtualization mechanisms. 55, 56

Virtual Switches Software-based switches running at the hypervisor-level and provide connectivity to VMs. 58

VLAN A standardized [4] implementation of a logically separated Local Area Network (LAN) that shares a single broadcast domain. Each VLAN has an associated numerical ID, also called VLAN tag, allocated between 1 and 4,095. We say VLAN_100 to refer to the VLAN with numerical ID 100. 55, 58

VTEP Virtual bridges responsible for encapsulating and de-encapsulating packets in overlay networks. 64

VXLAN A layer 2 in layer 3 tunneling protocol. It allows an overlay layer 2 network to spread across multiple underlay layer 3 network domains. It enables defining about 16 million virtual networks by encapsulating Ethernet frames into IP packets with a 24-bit tunneling header. 58

Chapter 1

Introduction

1.1 Motivation

Cloud computing is the paradigm of information technology (IT) as a utility, which has shifted over the past decade from a buzzword to an integral part of IT. Companies are rapidly incorporating the cloud to run their business applications instead of investing up-front capital and operational expenditures for deploying and maintaining heavy on-premise IT infrastructures.

Cloud service providers (CSPs) leverage large pools of high performance configurable computing platforms, virtualization technologies and high-speed networks to deliver ubiquitous, convenient and on-demand access to a seemingly unlimited amount of resources that can be rapidly commissioned, scaled in and out, and released with minimal interaction effort. There exist three well established cloud service models, namely, infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). Furthermore, cloud computing offerings can be classified based on their deployment models into two main categories: public and private clouds. In the former deployment, the cloud infrastructure is meant to be simultaneously used by multiple customers (e.g., industrial companies, government organizations, academic institutions, etc), while in the latter deployment, the

cloud infrastructure is meant to be exclusively provisioned by a unique customer. The work of this thesis specifically focuses on IaaS models in public cloud deployments.

The multi-tenancy model enables CSPs to achieve the promised cost-effectiveness, which has been, so far, the key for the wide cloud adoption. However, virtualization technologies, which make resource multiplexing possible, typically do not provide perfect logical isolation between tenants' resources and add an important layer of complexity to the cloud-stack. This makes the security-level of public cloud vary inversely with the multi-tenancy model payoff [5]. Therefore, the cloud security concerns together with the loss of control, the lack of transparency and the non-compliance risks, make many business owners and prospective customers still reluctant towards the adoption of the cloud [6]. Auditing security compliance of cloud implementations with respect to standards, and threat evaluation constitute viable solutions to bring more visibility into the cloud and boost the trust of tenants in CSPs as for the proper management and the protection of their assets.

Security auditing is an assurance approach which consists of checking whether the cloud-stack implementations are compliant with regulatory requirements and predefined security policies, while threat-evaluation consists of identifying and quantifying potential security threats in cloud deployments as part of the risk assessment process. The outcome of both security auditing and threat evaluation processes enables CSPs, on one side, to enhance the transparency of their services by providing customers with more awareness regarding the contractual compliance aspects and the security risks related to their outsourced applications. On the other side, it enables CSPs to improve the applied security measures and management strategies to have better control over different risks.

However, auditing in the cloud constitutes a real challenge. First, the significant gap between the high-level description of compliance recommendations (e.g., Cloud Control Matrix (CCM) [7] and ISO 27017 [8]) and the low-level raw logging information drastically hinders the auditing automation process. Second, the coexistence of a large number

of virtual resources on one side (e.g., a decent-size cloud is said to have around 1,000 tenants and 100,000 users [2]), and the important complexity that is brought by the virtualization layer on the other side, constitute a real challenge for identifying the relationship between different components. Third, the layered nature of the cloud-stack and the inter-layer dependencies make existing per-layer verification approaches ineffective, as multiple layers maintain different but complementary views of the isolation mechanisms configuration. Finally, correctly identifying the relevant data and their sources in the cloud for each security requirement increases the complexity of auditing. From another perspective, some policies and business requirements, may naturally require quantitative approaches to provide tenants with more visibility about the security posture of their outsourced virtual infrastructures. For instance, a tenant may want a specific threshold for resource sharing to minimize the multi-tenancy threats. In this case, appropriate tools (i.e., security metrics) are needed for threat evaluation to inform the tenant about the degree of compliance with his predefined requirements.

There exist various efforts on cloud auditing, however, those works either focus on verifying the operational properties that assess the behavioral aspect of the cloud (e.g., network reachability) [9], and omit the structural settings of tenants' virtual infrastructures, or they conduct the auditing process at a single layer of the cloud, which makes their solution not effectively capturing isolation breaches [10]. Furthermore, quantitative approaches and supporting metrics for per tenant threat evaluation are largely missing in the literature (a detailed literature review will be provided in Section 2.3).

1.2 Problem Statement

The research problem addressed in this thesis is drawn from the above mentioned challenges and limitations. We consider the broad context of security compliance auditing and

multi-tenancy threat evaluation in public cloud virtualized infrastructures. Specifically, todays' IaaS cloud platforms (e.g., Amazon AWS EC2 [11], Google Cloud Platform (GCP) [12] and Microsoft Azure [13]) expose rich APIs through which tenants can create several virtual infrastructures, which are mainly composed of virtual machines (VMs) and their connecting virtual private networks. Tenants' virtual infrastructures are then implemented at the physical-level via the cloud infrastructure management system (e.g., OpenStack), which is in charge of maintaining the logical segregation between tenants' resources. However, the highly dynamic, elastic and self-service nature of the cloud, together with the continuously increasing size of the managed resources (e.g., AWS reports 100K new VM instances created per day [14]) introduce a very high-level of complexity that may prepare the floor for misconfigurations leading to non-compliance with security standards and increased threat-levels. Particularly, failure to properly implement network isolation mechanisms to segregate multiple segments of virtual private networks may lead to interference of traffic belonging to different corporations. Moreover, the cloud elasticity mechanisms may cause VMs belonging to different trust levels to co-reside within a close proximity, leading to potential threats (e.g., information leakage through side channel attacks [15]).

In this thesis, we propose approaches and tools to bring more transparency into the security posture of actual cloud implementations, which would provide CSPs with more credibility and increase tenants' trust.

Particularly, for security auditing, we address the following research questions:

- How to bridge the gap between high-level standards and low-level cloud implementation details, and automate the security auditing process?
- How to leverage the complex inter-dependencies between the cloud-stack layers to capture subtle virtual network isolation breaches in tenants' virtual infrastructures?

As for threat evaluation in cloud infrastructures, the questions we tackle are:

- How to evaluate the degree of exposure of tenants' virtual infrastructures with respect to other potentially distrusted tenants based on resource sharing at different levels of the cloud infrastructure?

- How to evaluate abnormal degrees of co-residency between a victim and potential attackers independently of specific attacks?

We elaborate on our contributions to address those questions in the following section.

1.3 Contributions

Our contributions mainly revolve around providing a formal verification support for the compliance auditing of cloud infrastructures, and security metric tools to support per tenant threat evaluation, which provides an increased level of transparency to tenants.

Automated security auditing of cloud virtualized infrastructures. The objective of the first work of the thesis is to provide an automated approach for the verification of the proper configuration of virtual resources based on structural properties (e.g., assignment of instances to physical hosts and configuration of virtualization mechanisms). To this end, we focus on filling the existing gap between the high-level security standards, and low-level cloud implementations. In this respect, we first compile a list of structural security properties relevant to the cloud virtualized environment. The latter list maps into different recommendations described in several security compliance standards in the field of cloud computing. Afterwards, we map each security property to the relevant set of cloud infrastructure data sources (e.g., configuration and logged information from different cloud layers). Additionally, we formalize the extracted properties into First Order Logic (FOL). Finally, we transform the formalized properties and the audit data into a constraint satisfaction problem (CSP) to verify the security properties and provide audit evidence using

an off-the-shelf CSP-solver. We furthermore implement our auditing approach into OpenStack [16], one of the most commonly used cloud infrastructure management systems, and conduct experiments to show the scalability of our approach (e.g., we audit a dataset of 300,000 virtual ports, 24,000 subnets, and 100,000 VMs in less than 8 seconds). We elaborate on the details of this work in Chapter 3.

Consistent virtual layer 2 network isolation verification. As network isolation failures are among the foremost security concerns in the cloud [17, 18], our objective in the second part of this work is to verify layer 2 virtual private networks isolation[1], taking into consideration the inter-dependencies between different cloud layers' views. To the best of our knowledge, this is the first effort on auditing cloud infrastructure isolation at layer 2 virtual networks and overlay taking into account cross-layer consistency in the cloud-stack. To capture the semantics of the audit data and its relation to consistent isolation requirements, we first devise a multi-layered model for data related to each cloud layer's view, then we derive a set of concrete security properties to check the proper configuration of layer 2 and overlay isolation mechanisms. Furthermore, we integrate our auditing system into OpenStack, and present our experimental results on assessing several properties related to virtual network isolation and consistency. Our results show that our approach can be successfully used to detect virtual network isolation breaches for large OpenStack-based data centers in a reasonable time (e.g., we audit a dataset of 60k VMs in less than 4.6 seconds). The details of this work are presented in Chapter 4.

Distance metrics for evaluating multi-tenancy threats. The multi-tenancy threats resulting form resource sharing at different levels of the cloud infrastructure, constitute some of the main security concerns as tenants' resources are exposed to distrusted parties. In this work, we define a multi-level physical distance that captures the threats related to cross-tenant attacks requiring resource sharing not only at the host-level but also at higher levels

[1] We refer to the network layers defined in the Open Systems Interconnection (OSI) model

of the cloud infrastructure. We then refine this physical distance along the compute and network dimensions to evaluate the degree of compute and network resource exposure with respect to potential attackers. We show the effectiveness of our metrics through case studies conducted both on real and synthetic cloud data. We further show the applicability of our metrics for quantified auditing and per tenant risk assessment. We elaborate on the details of this work in Chapter 5.

Security metrics for evaluating and mitigating co-residency threats. Although cross-tenant attacks may target different resources, they all require a degree of co-residency with the victim as a prerequisite for the attacks to succeed. While existing metrics enable only to detect attacks at run-time by monitoring resource usage (e.g., the throughput [19]), we propose a set of attack-agnostic security metrics that capture abnormal degrees of co-residency between tenants' virtual infrastructures along two different dimensions, namely, the attack extent and the attack intensity. Our metrics enable to apply mitigation measures (e.g., VM migration) in order to avoid large scale damage. To show the applicability of our metrics and their usefulness in capturing increased co-residency threats, we conduct a case study on a real cloud data, and perform extensive experiments using CloudSim. The obtained results show the applicability of our metrics and their effectiveness in capturing abnormal co-residency degrees with the increased attacks' success rates. The details of this work are presented in Chapter 6.

In summary, the main contributions of this thesis are the following.

- We provide an automated solution for auditing the security compliance of cloud infrastructures against standards and tenants' predefined requirements. As per our knowledge, this is the first effort for formally verifying security properties related to the structural settings of tenants' virtual infrastructures implementations in the cloud.

- While existing works focus on one cloud layer only, we propose an automated framework for auditing consistent isolation between virtual networks in OpenStack-managed

cloud spanning over overlay networks and layer 2 virtual networks by considering multiple cloud layers' views.

- We integrate our auditing systems into OpenStack, and present our experimental results to show the applicability and the scalability of our auditing solutions.

- We propose suites of security metrics to evaluate the threat-level related to the multi-tenancy situation in public cloud. We conduct case studies and experiments on both real and fictitious clouds. The obtained results show the effectiveness and applicability of our metrics.

1.4 Thesis Structure

This thesis is organized into six chapters as follows. Chapter 2 provides a background on cloud computing, virtualization and multi-tenancy, and discusses existing works on security auditing and threat evaluation in the cloud. Chapter 3 presents our security auditing solution for virtualized cloud infrastructures. Therein, we further detail our formal verification methodology, elaborate on the implementation details, and discuss the experimental results. In Chapter 4, we detail our approach for layer 2 virtual networks isolation verification in OpenStack-managed cloud deployments. In Chapter 5, we present our multi-level security metrics for evaluating the distance between tenants' virtual infrastructures inside cloud deployments. We further show through several use cases, the applicability and usefulness of our metrics. Chapter 6 details our methodology for deriving security metrics to evaluate the proximity between tenants' resources starting from the common prerequisites to co-residency attacks in multi-tenant clouds. We show the usefulness of our metrics through use cases and extensive simulations. The conclusion and discussion on potential future works are summarized in Chapter 7.

Chapter 2

Background and Related Work

2.1 Introduction

Cloud service providers typically adopt the multi-tenancy model to optimize resources usage and achieve the promised cost-effectiveness. Sharing resources between different tenants and the underlying complex technology increase the necessity for transparency and accountability. In this regard, auditing security compliance of the provider's infrastructure against standards, regulations and customers' policies, and evaluating the potential threats related to the multi-tenancy situation, take on an increasing importance to boost the trust between the cloud stakeholders. In this chapter, we provide a description of the cloud and its services. Furthermore, we briefly elaborate on virtualization and multi-tenancy, as those are the main aspects of IaaS around which articulates our thesis work. Finally, we provide a literature review on cloud security auditing and multi-tenancy threats evaluation.

2.2 Cloud Computing

To run their business workloads, enterprises traditionally have to invest prohibitive costs for owning or licensing, running and maintaining data center equipment. In this respect,

cloud offerings propose to substantially lighten this burden by providing underlaying infrastructures and services enabling customers to more focus on their core business.

In [20], the National Institute of Standards and Technology (NIST) defines the cloud as *"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"*. Furthermore, NIST specifies three main cloud models based on the delivered resources and customers' capabilities. Those models can be described as follows:

- *Software as a service (SaaS)*. In this model, CSPs offer ready to use applications that can be enjoyed by the customers through the Internet. SaaS offerings are currently massively used both by individuals and organizations. For instance, Google offers the Google Calendar application. In this model, tenants do not have control, neither over the applications' development platform nor over the virtual infrastructure.

- *Platform as a service (PaaS)*. In this model, CSPs provide ready to use development platforms. Tenants run their applications on top of those frameworks and control their settings and configuration, but they do not have control over the underlying virtual infrastructure. An example of PaaS provider is Google App Engine[1].

- *Infrastructure as a service (IaaS)*. In this model, tenants can provision basic computing resources (i.e., processing, networking and storage) to deploy and run their own virtual infrastructures. Those virtual infrastructures are mainly composed of VMs and their connecting virtual private networks. Tenants' VMs are self-controlled and are allocated into virtualised physical machines using VM placement policies. Virtual private networks connecting those VMs are implemented using software networking devices (e.g., Open vSwitch [21]) and network virtualization mechanisms

[1] https://cloud.google.com/appengine/docs/

such as virtual LAN (VLAN) and Virtual Extended LAN (VXLAN). Examples of commercial IaaS providers are Amazon AWS EC2 [11], Google Cloud Platform (GCP) [12] and Microsoft Azure [13].

In this thesis, we are more interested in IaaS model, where tenants have control over their virtual infrastructures, which are directly implemented on top of the cloud facilities. Specifically, we focus on public cloud deployments, where virtualization is leveraged to enable resource sharing between multiple tenants.

2.2.1 Virtualization

Virtualization is the key underlaying technology for IaaS cloud. It provides the required elasticity for on demand services, and enables resource sharing to achieve cost-effectiveness. Host virtualization enables to run multiple VMs on top of one hardware platform. Those VMs are managed by a software called hypervisor (also called virtual machine monitor). The latter partitions the physical machine's resources and provides a logical isolation between VMs so that each VM has access and visibility to its assigned resources only [5].

To provide network connectivity between tenants' VMs, especially in large scale cloud infrastructures, network virtualization plays a vital role, as an example, virtual switches such as Open vSwitch (OVS) [22] are used. Furthermore, in order to support the highly dynamic and elastic nature of tenants' virtual infrastructures, virtual switches export interfaces for remote and runtime configuration, as a response to various events (e.g., VMs creation, shut down or migration).

Those interacting layers of virualization induce an increased complexity in cloud infrastructures, which opens up the floor for security breaches and vulnerabilities that can be exploited by malicious entities. For instance, a poorly configured hypervisor can be an easy target to different attacks (e.g., escape and hyper-jacking attacks) [23], which would threaten all the VMs running on top of it. Furthermore, the complex interactions between

the cloud management layers and the virtualized network layer at the implementation level, together with the large number of tenants' virtual private networks, may result in network misconfigurations leading to traffic interference (a detailed example will be discussed in Chapter 4). Additionally, virtualization technologies inevitably create side channels that can be exploited by malicious insiders to breach tenants' confidentiality (e.g., Hammer attack [24]).

2.2.2 Multi-Tenancy

CSPs adopt the multi-tenancy model through resource sharing to increase the financial gain, which is one of the driving factors to the adoption of public cloud. On the down side, by allowing co-residency between business competitors or selfish and malicious customers [25], multi-tenancy significantly expands the attack surface of shared cloud environments [26, 27]. In fact, recent works have demonstrated the feasibility of real-life attacks conducted in commercial clouds including Amazon EC2, aiming at forcing malicious virtual machines (VMs) to co-reside with targeted VMs either inside the same host or at higher proximity levels (e.g., the rack level) inside the cloud data center [28, 29]. Once co-residency achieved, attackers can mount harmful attacks against integrity, availability and confidentiality of their target's assets via side-channels, covert-channels, etc.

2.3 Related Work

This section reviews the related work on cloud security auditing and multi-tenancy threats evaluation.

2.3.1 Security Auditing

Table 2.1 summarizes the qualitative comparison between existing works on compliance verification in the cloud and our work. We compare the proposals based on the types of verified properties, structural or operational (structural properties are related to the static configuration of the virtualized infrastructure, while operational properties are related to the forwarding network functionality), the coverage of multiple cloud stack layers and cross-layer consistency, and finally the approach, which is either retroactive (off-line) or intercept-and-check (on-line) [30].

Proposal	Properties		Coverage		Approach	
	Structural	Operational	One/Multiple layers	Cross-layer	Retroactive	Intercept-and-check
Anteater [31]		•	One		•	
Hassel [32]		•	One		•	
VeriFlow [33]		•	One			•
NetPlumber [34]		•	One			•
Save [35]	•		One		•	
CloudRadar [36]	•		One			•
Xu et al. [37]	•		Multiple	•	•	
Congress [38]	•		One		•	•
Majumdar et al. [39]	•		One		•	
Majumdar et al. [40]	•		One			•
Madi et al. [41]	•		Multiple	•	•	
ISOTOP [42]	•		Multiple	•	•	

Table 2.1: Comparing features of existing solutions with our works. The symbol (•) indicates that the proposal offers the corresponding feature

To the best of our knowledge, our works on security auditing are the first to tackle the verification of the structural configurations and the topology isolation and consistency between cloud stack layers' views of the virtual layer 2 and overlay networks.

Several works target the verification of forwarding and routing rules, particularly in OpenFlow networks (e.g., [9, 43]). For instance, Anteater [31] verifies network invariants by translating them into instances of SAT problems and translating data plane information into boolean expressions. Then, it uses a SAT solver to check the resulting SAT formulas

to detect violations of key network invariants such as absence of loops and black-holes.

Hassel [32] is a protocol agnostic tool for checking network invariants and reachability-related policies. It is built on a geometric model where packet headers are modeled as points in a geometric space and network devices are modeled as invertible transfer functions defined on the same space. Then, custom algorithms are used to check network invariants and reachability-related policies.

VeriFlow [33], NetPlumber [34] (extension of [32]), and AP verifier [44] propose a near real-time verification, where network events are monitored for configuration changes, and verification is performed only on the impacted part of the network. Libra [9] uses a divide and conquer technique to verify forwarding tables in large networks. It encompasses a technique to capture stable and consistent snapshots of the network state and a verification approach based on graph search techniques that detects loops, black-holes and other reachability failures. Sphinx [43] enables incremental real-time network updates and constraints validation. It allows detecting both known and potentially unknown security attacks on network topology and forwarding plane. These works are complementary to our work as they aim at verifying operational properties of networks including reachability, isolation and absence of layer 3 network misconfiguration (e.g., loops, black-holes, etc.). However, they target mainly SDN environments and not necessarily the cloud, whereas our focus is more oriented towards auditing the structural properties of cloud virtualized infrastructures.

Some other works focus on security as a service to provide needed security. For instance, Mundada et al. [45] propose SilverLine, a collection of techniques that enables cloud providers to enforce data and network isolation for a cloud tenant's service. It uses a transparent operating system-level information-flow tracking layer assisted by an enforcement layer in the virtual machine monitor to provide data isolation. Our work aims at auditing compliance of security controls, which is considered as security assurance, and thus can be applied to such proposed security enforcement services.

In the context of cloud auditing, several works (e.g., [10, 46]) focus on firewalls and security groups. Probst et al. [46] present an approach for the verification of network access controls implemented by stateful firewalls in cloud computing infrastructures. Their approach combines static and dynamic verification with a discrepancy analysis of the obtained results against the clients' policies. Bleikertz [10] analyzes Amazon EC2 cloud infrastructures using reachability graphs and vulnerability discovery and builds attack graphs to find the shortest paths, which represent the critical attack scenarios against the cloud. The proposed approaches tackle layer 3 isolation mechanisms, but do not address challenges related to network virtualization mechanisms configuration issues and their impact on layer 2 virtual networks isolation, which are addressed by our work on cloud security auditing.

Other works focus on virtualization aspects (e.g., [35, 47, 48]). Bleikertz et al. [35, 47] propose SAVE, a static information flow analysis system for virtualized infrastructures based on graph traversal towards verifying information flow isolation. The configuration information is captured from the virtualization infrastructure via a set of probes created for different virtualization technologies. Then, the approach transforms the discovered configuration input into a graph, where vertices are resources such as virtual machines, hypervisors, physical machines, storage and network resources and edges represent information flows. The graph is traversed based on explicitly specified trust rules and information flow rules. Bleikertz et al. [36] extend the previous work to tackle near-real time security analysis of the virtualized infrastructure in the cloud. Their objective is mainly the detection of configuration changes that impact the security. A differential analysis based on computing graph deltas (e.g., added or removed nodes and edges) is proposed based on change events. The graph model is maintained synchronized with the actual configuration changes through probes that are deployed over the infrastructure and intercept events that may have a security impact. Contrarily to our auditing approach, this works do not involve properties

verification at multiple layers and cross-layer consistency verification, which reduces the scope of violations that can be detected compared to our approach. In our case, we correlate audit data collected form different sources and at different layers in order to detect violations that would definitely go unnoticed if relying only on one cloud layer at a time.

In [49], an autonomous agent-based incident detection system is proposed. The system detects abnormal infrastructure changes based on the underlying business process model. The framework is able to detect cloud resource and account misuse, distributed denial of service attacks and VM breakout. This related work is more oriented towards monitoring changes in cloud instances and infrastructures and evaluating the security status with respect to security business flow-aware rules.

Xu et al. [37] investigate network inconsistencies between network states extracted from OpenStack and the configuration of network devices. They use Binary Decision Diagrams (BDDs) to represent and verify these states. Similarly to our work, they tackle inconsistency verification. Xiang et al. [50] propose a graph-based OpenStack debugging approach enabling to extract the interaction between different modules from log files and databases. However, in these works, authors do not check isolation properties across different layers as suggested by our work. Furthermore, we are interested in auditing, thus our approach supports a wider view than simple verification, where log files are as important source of information as configuration.

In [51], authors propose a cross-layer data collection approach to reconstruct the network connectivity graph in cloud infrastructures. Our work can be extended to use the constructed connectivity graphs in order to audit tenant predefined security policies.

There exist other works (e.g., [38], [52], [30]) offering runtime security policy checking and enforcement in the cloud. Our work in [30] proactively verifies security compliance efficiently through pre-computation by utilizing dependency models. Weatherman [52] aims

at mitigating misconfigurations and enforcing security policies in a virtualized infrastructure. CloudSight [53] is a transparency-as-a-service abstraction that enables to track state changes of different tenants' components (e.g., VMs and virtual interfaces) by inserting monitoring functions in the infrastructure management system's modules.

In [54], authors propose a framework to evaluate the CSP's services prior to and after cloud adoption. However, the proposed framework only defines a set of generic concepts around which the auditing process should be articulating (e.g., actors, goals, risks and evidences) without providing any concrete implementation.

Congress [38] is an open project for OpenStack platforms. It enforces policies expressed by tenants and then monitors the state of the cloud to check its compliance. Furthermore, Congress attempts to correct policy violations when they occur. Our work shares the policy inspection aspect with Congress. Therefore, we integrated our solution in Congress as part of our contributions (see details in Section 4.4).

In the same fashion as the current work, formal verification approaches in [39, 55, 56] are proposed for checking security compliance in other security domains, mainly, Identity and Access Control. Majumdar et al. [39] propose auditing the multi-domain cloud at the user level with OpenStack as an application, which is a complementary effort to our work. Cotrini et al. [55] use FOL to express Role-based Access Control (RBAC) policies and rely on an off-the-shelf SMT solver to analyze them. In [56], authors apply model checking techniques to verify that access control policies implemented locally at the VM and hypervisor levels actually satisfy the global access control policies.

2.3.2 Multi-Tenancy Threats Evaluation

As per our knowledge, the work of this thesis is the first to propose metrics for quantifying the distance between tenants' virtual infrastructures inside cloud deployments.

Few works provide quantitative assessment frameworks to evaluate Security SLAs (SecLAs) in the cloud [57, 58, 59, 60]. For instance, Luna et al. [57] developed a set of metrics to quantitatively compare, benchmark and evaluate the security level of CSPs' reference SecSLAs. Authors in [58] propose a framework enabling cloud customers to choose the appropriate CSP according to their security requirements. In the same fashion, authors of [59] propose a CSPs' ranking mechanism based on Analytic Hierarchy Process (AHP). The latter work is extended in [60] by leveraging specific notions of Cloud SecSLAs adopted from current standardization efforts and real-world case studies. While those approaches provide valuable frameworks for prospective cloud customers to choose the right CSP based on the advocated SecLAs, our security metrics enable tenants to have more visibility on the multi-tenancy threat-level with respect to cloud implementation measurements.

Since VM-placement policies constitute the cloud infrastructure management component in charge of mapping tenants' VMs to hosts, many efforts have been deployed to harden placement policies against co-residency attacks. For instance, Han et al. [61, 62] proposed a VM-placement policy, namely, PSSF, to increase attackers' difficulty in achieving malicious co-residency with the victim. To evaluate the resistance of their placement policy to co-residency attacks, they further proposed a theoretical model composed of three metrics (efficiency, coverage and VM_{min}). Contrarily to our metrics, the proposed model is based on the assumption that the attacker is known, which is not the case in real cloud. By considering that any tenant sharing the cloud is a potential attacker, our metrics enable CSPs to proactively mitigate co-residency attacks, and tenants to have visibility on the security posture of their virtual infrastructures from the co-residency point of view.

SMOOP [63] is a security aware multi-objective VM-placement algorithm, which is based on risk assessment. The latter relies on a set of metrics to evaluate the cloud-level risk from multiple perspectives (VMs, hosts and network connections). Although, our

metrics do not provide risk assessment, they can be used to assess the co-residency threat related to different cloud deployments of tenants' virtual infrastructures.

In [29], authors propose an experimental approach to assess real cloud placement policies against co-residency attacks. They consider the random-placement policy as a yardstick against which they evaluated the success rate and relative cost of malicious VM-launch strategies for actual cloud VM-placement policies. Similarly, authors in [28] propose an experimental study based on intensive measurement probing to evaluate how resistant modern IaaS models (e.g., Amazon EC2 and their VPC) are against co-residency threats both at the host-level and at the rack-level. In this work, we take a complementary direction as we propose a suite of security metrics that enables to evaluate the co-residency threats according to tenants' virtual infrastructures with respect to current cloud deployments.

In [64], authors propose a CSP-assisted VM migration service that aims at limiting the information leakage due to side channels, by applying the moving target defense technique. Migrate [65] is another VM migration-based solution that mitigates side channel attacks in multi-tenant clouds. Our metrics can be used to evaluate the effectiveness of those approaches in reducing the multi-tenancy threats in cloud deployments.

Chapter 3

Auditing Security Compliance of the Virtualized Infrastructure in the Cloud: Application to OpenStack

3.1 Introduction

Several security challenges faced by the cloud, mainly the loss of control and the difficulty to assess security compliance of the cloud providers, leave potential customers reluctant towards its adoption. These challenges stem from cloud-enabling technologies and characteristics. For instance, virtualization introduces complexity, which may lead to new vulnerabilities (e.g., incoherence between multiple management layers of hardware and virtual components). At the same time, concurrent and frequent updates needed to meet various requirements (e.g., workload balancing) may create even more opportunities for misconfiguration, security failures, and compliance compromises. Cloud elasticity mechanisms may cause virtual machines (VMs) belonging to different corporations and trust levels to interact with the same set of resources, causing potential security breaches [66]. Therefore,

cloud customers take great interest in auditing the security of their cloud setup.

Security compliance auditing provides proofs with regard to the compliance of implemented controls with respect to standards as well as business and regulatory requirements. However, auditing in the cloud constitutes a real challenge. First, the coexistence of a large number of virtual resources on one side and the high frequency with which they are created, deleted, or reconfigured on the other side, would require to audit, almost continuously, a sheer amount of information, growing continuously and exponentially [67]. Furthermore, a significant gap between the high-level description of compliance recommendations (e.g., Cloud Control Matrix (CCM) [7] and ISO 27017 [8]) and the low-level raw logging information hinders auditing automation. More precisely, identifying the right data to retrieve from an ever increasing number of data sources, and correctly correlating and filtering it constitute a real challenge in automating auditing in the cloud.

We propose in this work to focus on auditing security compliance of the cloud virtualized environment. More precisely, we focus primarily on virtual resources isolation based on structural properties (e.g., assignment of instances to physical hosts and the proper configuration of virtualization mechanisms), and consistency of the configurations in different layers of the cloud (infrastructure management layer, software-defined networking (SDN) controller layer, virtual layer and physical layer). Although there already exist various efforts on cloud auditing (a detailed review of related works is given in Section 2.3.1), to the best of our knowledge, none has facilitated automated auditing of structural settings of the virtual resources while taking into account the multi-layer aspects.

Motivating example. The following illustrates the challenges to fill the gap between the high-level description of compliance requirements as stated in the standards and the actual low-level raw audit data. In CCM [7], the control on Infrastructure & Virtualization Security Segmentation recommends *"isolation of business critical assets and/or sensitive*

user data, and sessions". In ISO 27017 [8], the requirement on segregation in virtual computing environments mandates that *"cloud service customer's virtual environment should be protected from other customers and unauthorized users"*. Moreover, the segregation in networks requirements recommends *"separation of multi-tenant cloud service customer environments"*.

Clearly, any overlap between different tenants' resources may breach the above requirements. However, in an SDN/Cloud environment, verifying the compliance with the requirements requires gathering information from many sources at different layers of the cloud stack: the cloud infrastructure management system (e.g., OpenStack [16]), the SDN controller (e.g., OpenDaylight [68]), and the virtual components and verifying that effectively compliance holds in each layer. For instance, the logging information corresponding to the virtual network of tenant `0848cc1999-e542798` is available from at least these different sources:

- Neutron databases, e.g., records from table "Routers" associating tenants to their virtual routers and interfaces of the form `0848cc1999e542798 (tenants_id)` ‖ `420fe1cd-db14-4780` (vRouter_id) ‖ `6d1f6103-9b7a-4789-ab16 (vInterface_id)`.

- Nova databases, e.g., records from table "Instances" associating VMs to their owners and their MAC addresses as follows: `0721a9ac-7aa1-4fa9 (VM_ID)` ‖ `0848cc1999e542798 (tenants_id)` and `fa:16:-3e:cd:b5:e1 (MAC)` ‖ `0721a9ac-7aa1-4fa9(VM_ID)`.

- Open vSwitch databases information, where ports and their associated tags can be fetched in this form `qvo4429c50c-9d (port_name)` ‖ `1084(VLAN_ID)`.

As illustrated above, it is difficult to identify all the relevant data sources and to map information from those different sources at various layers to the standard's recommendations.

Furthermore, potential inconsistencies in these layers make auditing tasks even more challenging. Additionally, as different sources may manipulate different identifiers for the same resource, correctly correlating all these data is critical to the success of the audit activity.

To facilitate automation, we present a compiled list of security properties relevant to the cloud virtualized environment that maps into different recommendations described in several security compliance standards in the field of cloud computing. Our auditing approach encompasses extracting configuration and logged information from different layers, correlating the large set of data from different origins, and finally relying on formal methods to verify the security properties and provide audit evidence. We furthermore implement the verification of these properties and show how the data can be collected and processed in the cloud environment with an application to OpenStack. Our approach shows scalability as it allows auditing a dataset of 300,000 virtual ports, 24,000 subnets, and 100,000 VMs in less than 8 seconds.

The main contributions of our work are as follows:

- To the best of our knowledge, this is the first effort on auditing cloud virtualized environment from the structural point of view taking into account consistency between multiple control layers in the cloud.
- We identify a list of security properties from the literature that may fill the gaps between security standards recommendations and actual compliance validation and allows audit automation.
- We report real-life experience and challenges faced when trying to integrate auditing and compliance validation into OpenStack.
- We conduct experiments whose results show scalability and efficiency of our approach.

3.2 Methodology

In this section, we present some preliminaries and describe our approach for auditing and compliance validation.

3.2.1 Threat Model

We assume that the cloud infrastructure management system has implementation flaws and vulnerabilities, which can be potentially exploited by malicious entities. For instance, a reported vulnerability in OpenStack Nova networking service, OSSN-0018/2014 [69], allows a malicious VM to reach the network services running on top of the hosting machine, which may lead to serious security issues. We trust cloud providers and administrators, but we assume that some cloud users and operators may be malicious [70]. We trust the cloud infrastructure management system for the integrity of the audit input data (e.g., logs, configurations, etc.) collected through API calls, events notifications, and database records (existing techniques on trusted auditing may be applied to establish a chain of trust from TPM chips embedded inside the cloud hardware to auditing components, e.g., [71]). We assume that not all tenants trust each other. They can either require not to share any physical resource with all the other tenants, or provide a white (or black) list of trusted (or untrusted) customers that they are (not) willing to share resources with. Although our auditing framework may catch violations of specified security properties due to either misconfiguration or exploits of vulnerabilities, our focus is not on detecting specific attacks or intrusions.

Example 3.1. *For illustrating purposes in our running example, we consider two tenants. Tenant Alpha can be exposed to malicious outsiders and insiders. A malicious insider could be either an adversary (tenant Beta) sharing the same cloud resources with tenant Alpha or a malicious operator with a higher access privilege.*

3.2.2 Modeling the Virtualized Infrastructure

In a multi-tenant cloud Infrastructure as a Service (IaaS) model, the provider's physical and virtual resources are pooled to serve on demands from multiple customers. The IaaS cloud reference model [72] consists of two layers: The physical layer composed of networking, storage, and processing resources, and the virtualization layer that is running on top of the physical layer and enabling infrastructure resources sharing. Figure 3.1 refines the virtualization layer abstraction in [72] by considering tenant specific virtual resources such as virtual networks and VMs. Accordingly, a tenant can provision several VM instances and virtual networks. VMs may run on different hosts and be connected to many virtual networks through virtual ports. Virtualization techniques are used to ensure isolation among multiple tenants' boundaries. Host virtualization technologies enable running many virtual machines on top of the same host. Network virtualization mechanisms (e.g., VLAN and VXLAN) enable tenants' network traffic segregation, where virtual networking devices (e.g., Open vSwitches) play a vital role in connecting VM instances to their hosting machines and to virtual networks.

In addition to these virtual and physical resources illustrated as nodes, Figure 3.1 shows the relationships between tenants' specific resources and cloud provider's resources. These relations will be used in Section 3.4 for the formalization of both the virtualized infrastructure model and the security properties. For instance, *IsAttachedOnPort* is a relationship with arity 3. It attaches a VM to a virtual subnet through a virtual port. This model can be refined with several levels of abstraction based on the properties to be checked.

3.2.3 Cloud Auditing Properties

We classify virtualization related-properties into two categories: Structural and operational properties. Structural properties are related to the static configuration of the virtualized

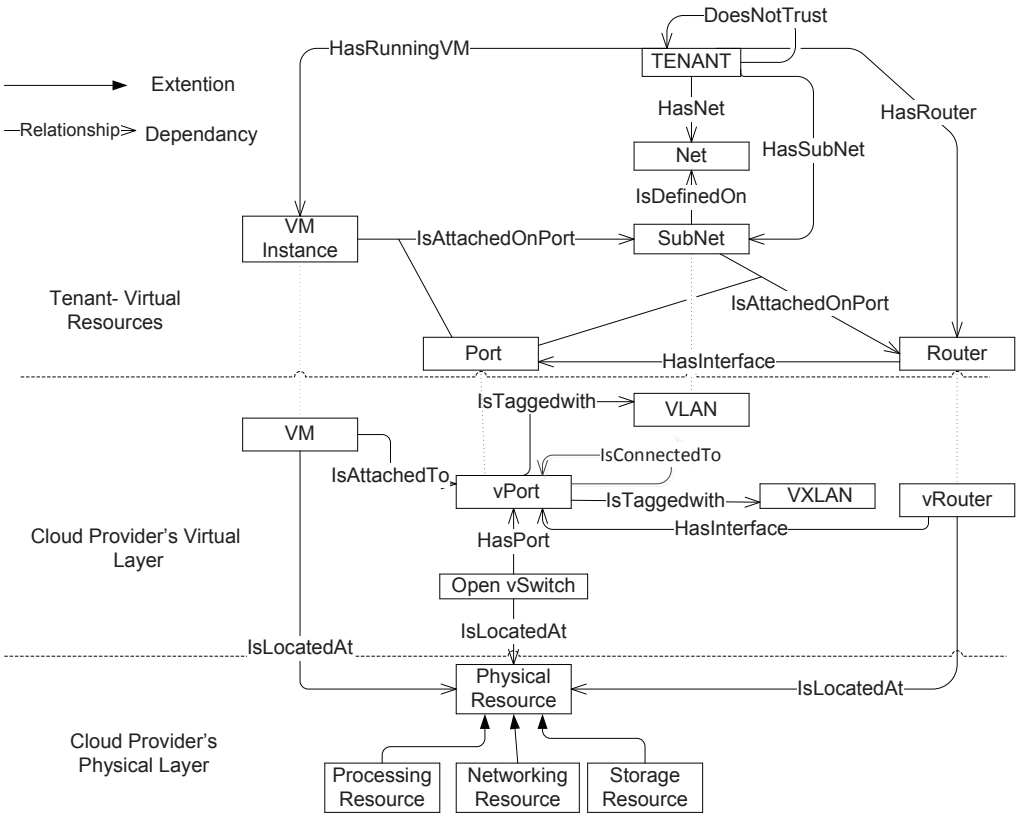


Figure 3.1: A generic model of the virtualized infrastructures in the cloud

infrastructure such as the assignment of instances to physical hosts, the assignment of virtual networking devices to tenants, and the proper configuration of isolation mechanisms such as VLAN configuration of each port. Operational properties are related to the forwarding network functionality. Those are mainly reachability-related properties such as loop-free forwarding and absence of black holes. Since the latter category has received significant attention in the literature (e.g., [9], [10], [43]), the former category constitutes the main focus of the current work. As the major goal of this work is to establish a bridge between high-level guidelines in the security standards and low-level logs provided by current cloud systems, we start by extracting a list of concrete security properties from those standards and the literature in order to more clearly formulate the auditing problem. Table 3.1 presents an excerpt of the list of security properties we consider for auditing relevant

standards (e.g., ISO 27002 [73], CCM [7]). Therein, we also classify properties based on their relevance to the stakeholders. In the following, we provide a brief description followed by an illustrating example for the sample properties, namely, absence of common ownership of resources, no co-residency, and topology consistency.

Virtual Resource Isolation (No common ownership). Resource sharing technology was not designed to offer strong isolation properties for a multi-tenant architecture and thus has been ranked by the CSA among the nine notorious threats related to the cloud [74]. The related risks include the failure of logical isolation mechanisms to properly segregate virtual resources assigned to different tenants, which may lead to situations where one tenant has access to another tenant's resources or data. The no common ownership property aims at verifying that no virtual resource is co-owned by multiple tenants. Tenants are generally allowed to interconnect their own virtual resources to build their cloud virtual networks by modifying their configurations. However, if a virtual resource (e.g., a router or a port) is co-owned by multiple tenants, it can be part of several virtual networks belonging to different tenants, which can potentially create a breach of isolation.

Example 3.2. *(No common ownership) This property has been violated in a real-life OpenStack deployment by exploiting the vulnerability OSSA-2014-008 [75] reported in the Neutron networking service, which allows a tenant to create a virtual port on another tenant's router. An instance of our model can capture this violation as illustrated in Figure 3.2. The model instance on the left side illustrates the initial entities and their relationships before exploiting the vulnerability. Assume that* `Tenant_Beta`*, by exploiting the said vulnerability, created* `vPort_21`*, and plugged it into* `vRouter_A1`*, which belongs to* `Tenant_Alpha`*. This would modify the model instance as illustrated on the right side showing the violation of no common ownership. Indeed,* `Tenant_Beta` *is the owner* `vPort_21` *as he is the initiator of the port creation. But since the port is connected to*

vRouter_A1*, the created port would be considered as a common resource for both tenants.*

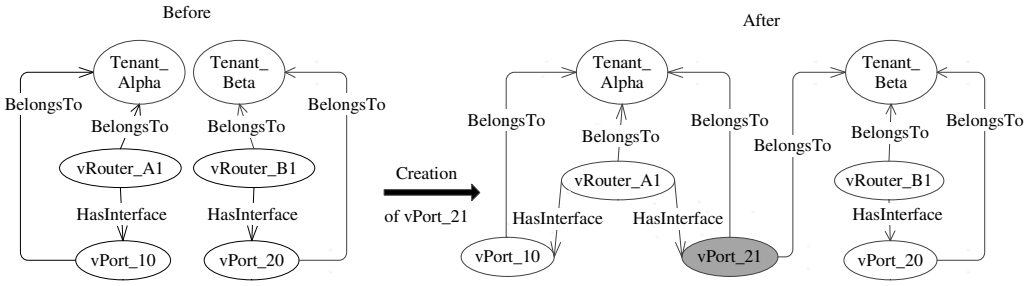


Figure 3.2: Model instances for the *no common ownership* property before and after the violation of *no common ownership* property. After creating port vPort_21, the latter becomes owned by two tenants.

Physical Isolation (No VM co-residency). To maximize resources utilization, cloud providers consolidate virtual machines, possibly belonging to competing customers, to be run on the same physical machine, which may cause major security concerns as described in [76]. Physical isolation [77] aims at preventing side and covert channel attacks, and reducing the risk of attacks staged based on hypervisor and software switches vulnerabilities (e.g., [78]) by hosting VMs in different physical servers. Such attacks might lead to performance degradation, sensitive information leakage, and denial of service.

Example 3.3. *(No VM co-residency) Figure 3.3 consists of two subsets of instances of the virtual infrastructure model presented in Section 3.2.2. At the left side of the figure, we have two virtual machines* VM_A1 *and* VM_A2 *belonging to* Tenant_Alpha *and running at compute node* CN_1*, and* VM_B1 *owned by* Tenant_Beta *while running at compute node* CN_2*. Because of lack of trust,* Tenant_Alpha *may require physical isolation of his VMs from those of* Tenant_Beta*. However, as illustrated at the right side of Figure 3.3,* VM_A2 *can be migrated from* CN_1 *to* CN_2 *for load balancing. This new instance of the model after migration illustrates the violation of physical isolation.*

Topology consistency. As stated in [79], it is critical to maintain consistency among cloud

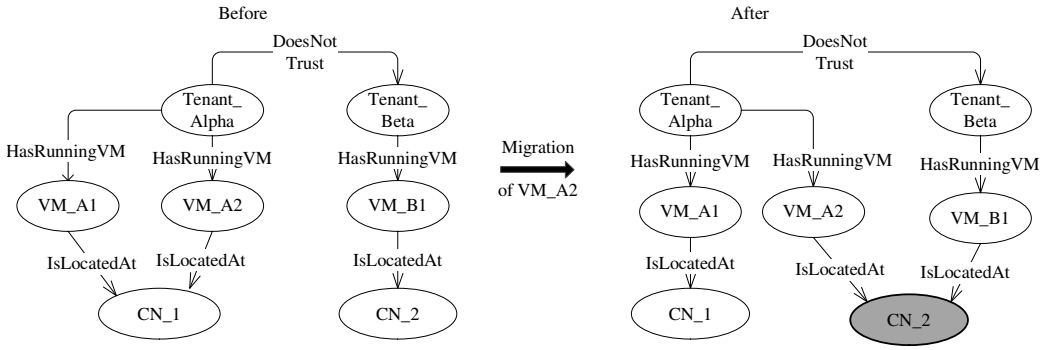


Figure 3.3: Subsets of the virtual infrastructure model instances before and after violation of the *no VM co-residency* property illustrating an example of data on VM locations. After migration, `VM_A2` becomes co-resident with `VM_B1` at compute node `CN_2`.

layers. The architectural model of the cloud can be described as a stack of layered services: physical layer, system resources layer, virtualized resources layer, support services layer, and at the top cloud-delivered services. Additionally, using SDN to implement network services increases management flexibility but also adds yet another layer in the stack. The presence of inconsistencies between these layers may lead to security breaches, which in turn makes the security controls at higher layers inefficient. Topology consistency consists of checking whether the topology view in the cloud infrastructure management system, matches the actual implemented topology, while considering different mappings between the physical infrastructure, the virtual infrastructure, and the tenants' boundaries.

Example 3.4. *(Port consistency) We suppose that a malicious insider managed to deliberately create a virtual port vPort_40 on Open_vSwitch_56 and label it with the VLAN identifier VLAN_100 that is already assigned to tenant Alpha. This would allow the malicious insider to sniff tenant Alpha's traffic by mirroring the VLAN_100 traffic to the created port, vPort_40. This clearly would lead to the violation of the network isolation property.*

As illustrated in Figure 3.4, we build two views of the virtualized topology: The actual topology is built based on data collected directly from the networking devices running at the virtualization layer (Open vSwitches), and the perceived topology is obtained from the infrastructure management layer (Nova and Neutron OpenStack databases). The dashed

lines map one to one the entities between the two topologies (not all the mappings are shown for more readability). We can observe that vPort_40 is attached to VLAN_100, which maps to Net_01 (tenant Alpha's network), but there is no entity at the infrastructure management layer that maps to the entity vPort_40 at the virtualization layer, which reveals a potential security breach.

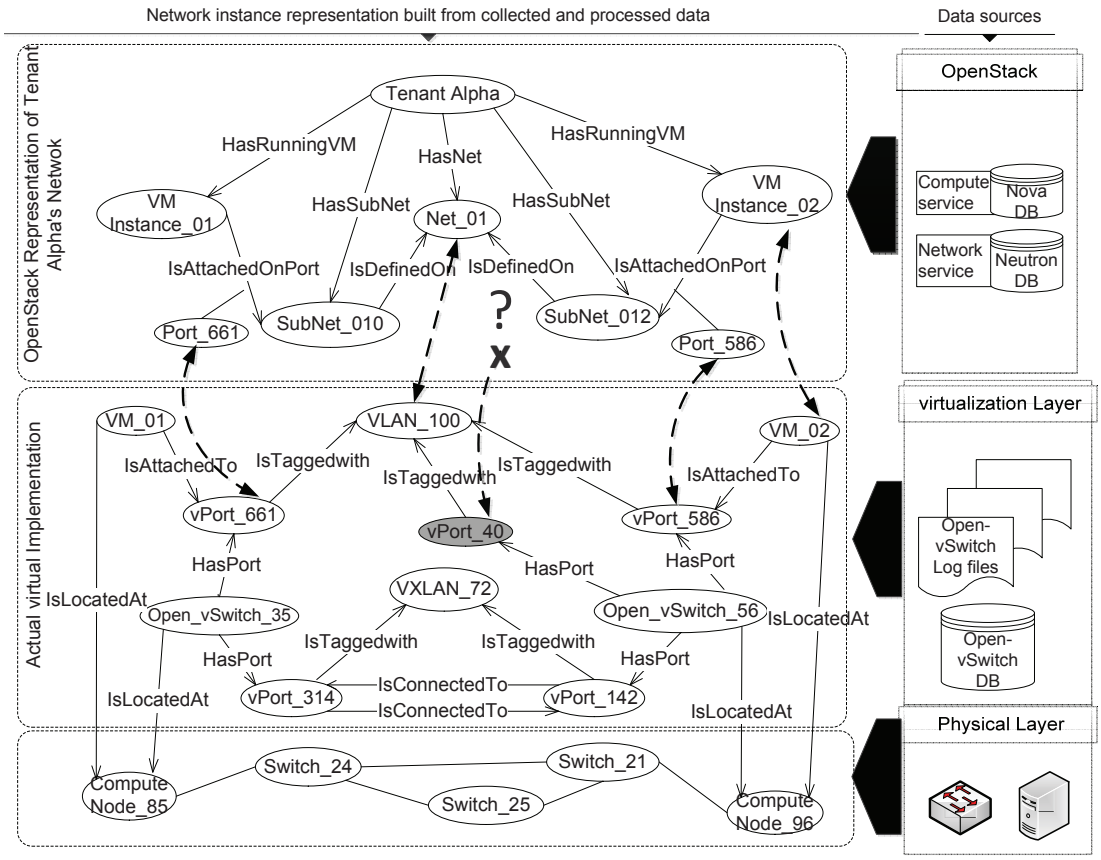


Figure 3.4: Virtualized infrastructure model instance showing an OpenStack representation and the corresponding actual virtual layer implementation. VXLAN_72 and its ports are part of the infrastructure implementation and do not correspond to any component in tenant Alpha's resources.

Other Security Properties. In the following, we briefly describe other security properties presented in Table 3.1.

- *Data and processing location correctness* One of the main cloud specific security

Subject	Properties and Sub-Properties		Standards			
			ISO27002 [73]	ISO27017 [8]	NIST800 [81]	CCM [7]
Tenant	Data and processing location correctness		18.1.1	18.1.1	IR-6, SI-5	SEF-01, IVS-04
	Virt. resource isolation (e.g., no common ownership)		-	CLD.9.5.1	-	STA-5, IVS-09
	Physical isolation (e.g., no co-residency)		-	13.1.3	SC-2	IVS-8, IVS-9
	Fault tolerance	Facility duplication	17.1, 17.2	12.1.3, 17.1, 17.2	PE-1, PE-13	BCR-03
		Storage service duplication				
		Redundant network connectivity				
Provider	No abuse of resources	Max number of VMs	-	-	-	IVS-11
		Max number of virtual networks				
	No resource exhaustion		-	-	-	IVS-05
Both	Topology consistency	inf. management view/virtual inf.	-	13.1.3	SC-2	IVS-8, IVS-9
		SDN controller view/ virtual inf.				

Table 3.1: An excerpt of security properties

issues is the increased complexity of compliance with laws and regulations [80]. The cloud provider might have data centers spread over different continents and governed by various court jurisdictions. Data and processing can be moved between the cloud provider's data centers without tenants' awareness, and fall under conflicting privacy protection laws.

- *Redundancy and fault tolerance* Cloud providers have to apply several measures to achieve varying degrees of resiliency following the criticality of tenants' applications. Duplicating facilities in various locations, and replicating storage services are examples of the measures that could be undertaken. Considering additional redundancy of network connectivity and information processing facilities has been mentioned in ISO 27002:2013 [73] as one of best practices.

- *No abuse of resources* Cloud services can be used by legitimate anonymous customers as a basis to illegitimately lead criminal and suspicious activities. For example, cloud services can be used to stage DDoS attacks [74].

- *No resource exhaustion* The ease with which virtual resources can be provisioned in the cloud introduces the risk of resource exhaustion [82]. For example, creating a huge amount of VMs within a short time frame drastically increases the odds of misconfiguration, which opens up several security breaches [83].

3.3 Audit Ready Cloud Framework

Figure 3.5 illustrates a high-level architecture of our auditing framework. It has five main components: data collection and processing engine, compliance validation engine, audit report engine, dashboard, and audit repository database. The framework interacts mainly with the cloud management system, the cloud infrastructure system (e.g., OpenStack), and

elements in the data center infrastructure to collect various types of audit data. It also interacts with the cloud tenant to obtain the tenant requirements and to provide the tenant with the audit result. Tenant requirements encompass both general and tenant-specific security policies, applicable standards, as well as audit queries. In the following, we only focus on the major components.

Our data collection and processing engine is composed of two sub-engines: the collection engine and the processing engine. The collection engine is responsible for collecting the required audit data in a batch mode, and it relies on the cloud management system to obtain the required data. The role of the processing engine is to filter, format, aggregate, and correlate this data. The required audit data may be distributed throughout the cloud and in different formats. The processing engine must pre-process the data in order to provide specific information needed to verify given properties. The last processing step is to generate the code for compliance validation and then store it in the audit repository database to be used by the compliance validation engine. The generated code depends on the selected back-end verification engine.

The compliance validation engine is responsible for performing the actual verification of the audited properties and the detection of violations, if any. Triggered by an audit request or updated inputs, the compliance validation engine invokes our back-end verification and validation engines. We use formal methods to capture formally the system model and the audit properties, which facilitates automated reasoning and is generally more practical and effective than manual inspection. If a security audit property fails, evidence can be obtained from the output of the verification back-end. Once the outcome of the compliance validation is ready, audit results and evidences are stored in the audit repository database and made accessible to the audit reporting engine. Several potential formal verification engines can serve our needs, and the actual choice may depend on the property being verified.

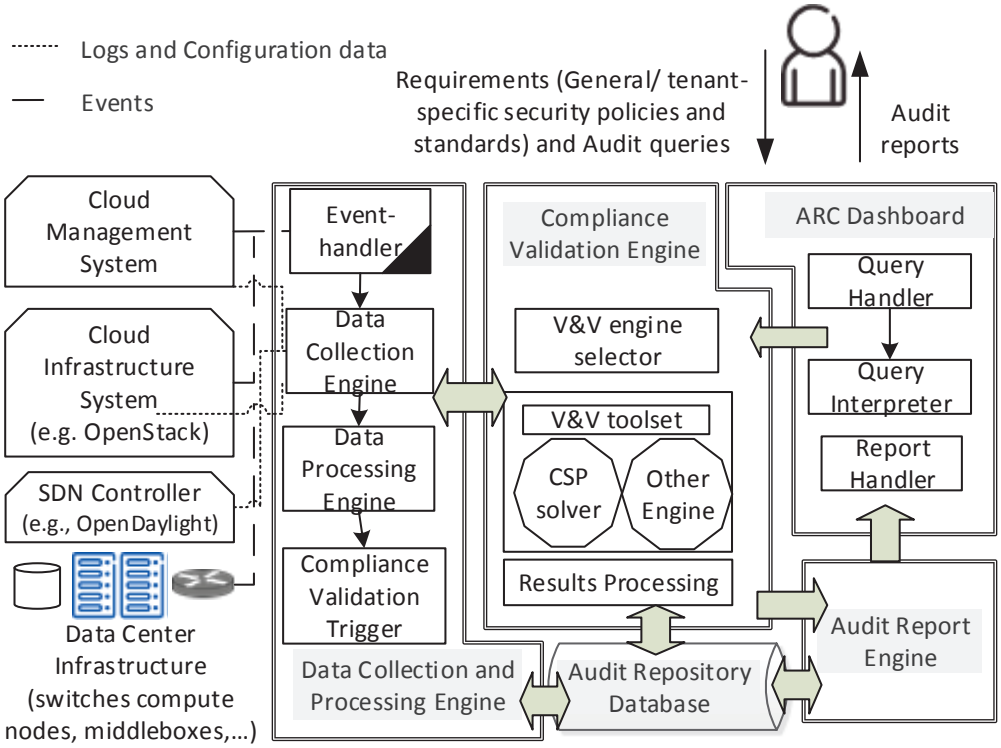


Figure 3.5: A high-level architecture of our cloud auditing framework

3.4 Formal Verification

As a back-end verification mechanism, we propose to formalize audit data and properties as Constraint Satisfaction Problems (CSP) and use a constraint solver, namely Sugar [84], to validate the compliance. CSP allows formulation of many complex problems in terms of variables defined over finite domains and constraints. Its generic goal is to find a vector of values (a.k.a. assignment) that satisfies all constraints expressed over the variables. If all constraints are satisfied, the solver returns SAT, otherwise, it returns UNSAT. In the case of a SAT result, a solution to the problem is provided. The key advantage of using CSP comes from the fact that it enables uniformly presenting the system's setup and specifying the properties in a clean formalism (e.g., First Order Logic (FOL) [85]), which allows to check a wide variety of properties [86]. Moreover using CSP avoids the state space traversal,

which makes our approach more scalable for large data sets.

3.4.1 Model Formalization

Depending on the properties to be checked, we encode the involved instances of the virtualized infrastructure model as CSP variables with their domains definitions (over integer), where instances are values within the corresponding domain. For example, *Tenant* is defined as a finite domain ranging over integer such that *(domain TENANT 0 max_tenant)* is a declaration of a domain of tenants, where the values are between 0 and *max_tenant*. Relations between classes and their instances are encoded as relation constraints and their supports, respectively. For example, *HasRunningVM* is encoded as a relation, with a support as follows: (relation $HasRunningVM$ 2 ($supports(vm1,t1)(vm2,t2)$)). The support of this relation will be fetched and pre-processed in the data processing step. The CSP code mainly consists of four parts:

- *Variable and domain declaration*. We define different entities and their respective domains. For example, t is a variable defined over the domain $TENANT$, which ranges over integers.

- *Relation declaration*. We define relations over variables and provide their support from the audit data.

- *Constraint declaration*. We define the negation of each property in terms of predicates over the involved relations to obtain a counter-example in case of a violation.

- *Body*. We combine different predicates based on the properties to verify using Boolean operators.

Relations in Properties	Evaluate to *True* if
$BelongsTo(r,t)$	The resource r is owned by tenant t
$HasRunningVM(vm,t)$	The tenant t has a running virtual machine vm
$DoesNotTrust(t1,t2)$	Tenant $t2$ is not trusted by tenant $t1$ which means that $t1$'resources should not share the same hardware with $t2$' instances
$IsLocatedAt(vm,cn)$	The instance vm is located at the compute node cn
IsAssignedPortVLAN (p,v,t)	the port p is assigned to the VLAN v which is in turn assigned to tenant t
$HasPortVLAN(vs,p,v)$	The port p is created at the virtual switch vs and assigned to VLAN v

Table 3.2: First Order Logic predicates

3.4.2 Properties Formalization

Security properties would be expressed as predicates over relation constraints and other predicates. We express the sample properties in FOL. Table 3.2 summarizes the predicates required for expressing the properties. Those predicates correspond to CSP relation constraints used to describe the current configuration of the system. Note that predicates that do not appear as relationships in Figure 3.1 are inferred by correlating other available relations.

No Common Ownership. We check that a tenant-specific virtual resource belongs to a unique tenant.

$$\forall \texttt{r} \in \texttt{Resource}, \forall \texttt{t1}, \texttt{t2} \in \texttt{TENANT} \tag{1}$$

$$\texttt{BelongsTo}(\texttt{r},\texttt{t1}) \wedge \texttt{BelongsTo}(\texttt{r},\texttt{t2}) \quad \rightarrow (\texttt{t1} = \texttt{t2})$$

No Co-residence. Based on the collected data, we check that the tenant's instances are not co-located in the same compute node with adversaries' instances.

$$\forall \texttt{t1}, \texttt{t2} \in \texttt{TENANT}, \forall \texttt{vm1}, \texttt{vm2} \in \texttt{INSTANCE}, \tag{2}$$
$$\forall \texttt{cn1}, \texttt{cn2} \in \texttt{COMPUTEN} :$$
$$\texttt{HasRunningVM}(\texttt{vm1}, \texttt{t1}) \wedge \texttt{HasRunningVM}(\texttt{vm2}, \texttt{t2}) \wedge$$
$$\texttt{DoesNotTrust}(\texttt{t1}, \texttt{t2}) \wedge \texttt{IsLocatedAt}(\texttt{vm1}, \texttt{cn1}) \wedge$$
$$\texttt{IsLocatedAt}(\texttt{vm2}, \texttt{cn2}) \rightarrow \texttt{cn1} \neq \texttt{cn2}$$

Topology Consistency. We check that mappings between virtual resources over different layers are properly maintained and that the current view of the cloud infrastructure management system on the topology, matches the actual topology of the virtual layer. In the following, we consider port consistency as a specific case of topology consistency. We check that the set of virtual ports assigned to a given tenant's VLAN by the provider correspond exactly to the set of ports inferred from data collected from the actual infrastructure's configuration for the same tenant's VLAN.

$$\forall \texttt{vs} \in \texttt{vSWITCH}, \quad \forall \texttt{p} \in \texttt{Port} \quad \forall \texttt{t} \in \texttt{TENANT} \quad \forall \texttt{v} \in \texttt{VLAN} \tag{3}$$
$$\texttt{HasPortVlan}(\texttt{vs}, \texttt{p}, \texttt{v}) \Leftrightarrow \texttt{IsAssignedPortVLAN}(\texttt{p}, \texttt{v}, \texttt{t})$$

Example 3.5. *Listing 3.1 is the CSP code to verify the no common ownership, no co-residence and port consistency properties for our running example. Variables along with their respective domains are first declared. Based on the properties of interest, a set of relations are defined and populated with their supporting tuples, where the support is generated from actual data in the cloud. Then, the properties are declared as predicates over*

these relations. Finally, the disjunction of the predicates is instantiated for verification. As we are formalizing the negation of the properties, we are expecting the UNSAT result, which means that none of the properties holds (i.e., no violation of the properties). We present the verification outputs in Section 3.5.

Listing 3.1: Sugar source code for common ownership, co-residence and port consistency property verification

```
1  //Declaration
2  (domain TENANT 0 60000) (domain RESOURCE 0 216000)
3  (domain INSTANCE 0 100000) (domain HOST 0 1000)
4  (domain PORT 0 300; 000) (domain VLAN 0 60000)
5  (domain VSWITCH  0 1000)
6  (int T1 TENANT) (int T2 TENANT)
7  (int R1 Resource) (int R2 Resource)
8  (int VM1 INSTANCE) (int VM2 INSTANCE)
9  (int H1 HOST) (int H2 HOST)(int V VLAN)
10 (int T TENANT) (int P PORT)  (int vs VSWITCH)
11 //Relations Declarations and Audit data as their support
12 (relation BelongsTo 2 (supports (18037 10)(18038 10)( 18039 10)
13 (18040 10)(18038 11)(18042 11)(18043 11)(18044 11)(18045 11)
14 (18046 12)(18047 12)))
15 (relation HasRunningVM 2 (supports (6100 10)(6101 10)(6102 11)
16 (6103 11)(6104 11)(6105 11)))
17 (relation IsLocatedAt 2 (supports(((6089 11000)(6090 11000)
18 (6093 11000)(6101 11100)(6102 11100))
19 (relation DoesNotTrust 2 (supports(9 11)(9 13)(9 14)))
20 (relation IsAssignedPortVLAN 3 (supports (18028 6017 9)(18029 6018 9)
21 (18030 6019 10)(18031 6019 10)(18032 6020 10) ))
22 (relation HasPortVLAN 3 (supports(511 18030 6019)(511 18031 6019 10)
23 (512 18032 6020)(512 18033 6021)))
24 //Security properties expressed in terms of predicates over relations
```

```
25  (predicate (CommonOwnership T1 R1 T2 R2)
26  (and (BelongsTo T1 R1) (BelongsTo T2 R2 ) (= R1 R2) (not (= T1 T2))))
27  (predicate (coResidence T1 T2 VM1 VM2 H1 H2) (and (DoesNotTrust T1 T2)
28  (HasRunningVM VM1 T1)(HasRunningVM VM2 T2) (IsLocatedAt H1 VM1)
29  (IsLocatedAt H2 VM2) (=H1 H2)))
30  (predicate (portConsistency P V T)
31  (or (and (IsAssignedPoprtVLAN P V T)
32  (not(HasPortVLAN VS P V)))
33  (and (HasPortVLAN VS P V) (not(IsAssignedPoprtVLAN P V T)))))
34  \\The Body
35  (or (CommonOwnership T1 R1 T2 R2) (coResidence T1 T2 VM1 VM2 H1 H2)
36  (portConsistency P V T) )
```

3.5 Application to OpenStack

This section describes how we integrate our audit and compliance framework into OpenStack. First, we briefly present the OpenStack networking service (Neutron), the compute service (Nova) and Open vSwitch [21], the most popular virtual switch implementation. We then detail our auditing framework implementation and its integration in OpenStack along with the challenges that we faced and overcame.

3.5.1 Background

OpenStack [16] is an open-source cloud infrastructure management platform that is being used almost in half of private clouds and significant portions of the public clouds (see [1] for detailed statistics). The major components of OpenStack to control large collections of computing, storage and networking resources are respectively Nova, Swift and Neutron along with Keystone. Following is the brief description of Nova and Neutron:

Nova [16] This is the OpenStack project designed to provide massively scalable, on demand, self service access to compute resources. It is considered as the main part of an Infrastructure as a Service model.

Neutron [16] This OpenStack system provides tenants with capabilities to build rich networking topologies through the exposed API, relying on three object abstractions, namely, networks, subnets and routers. When leveraged with the Modular Layer 2 plug-in (ML2), Neutron enables supporting various layer 2 networking technologies. For our testbed we consider Open vSwitch as a network access mechanism and we maintain two types of network segments, namely, VLAN for communication inside of the same compute node, and VXLAN for inter compute nodes communications.

Open vSwitch [21]. Open vSwitch is an open source software switch designed to be used as a vSwitch in virtualized server environments. It forwards traffic between different virtual machines (VMs) on the same physical host and also forwards traffic between VMs and the physical network.

3.5.2 Integration to OpenStack

We focus mainly on three components in our implementation: the data collection engine, the data processing engine, and the compliance validation engine. The data collection engine involves several components of OpenStack e.g., Nova and Neutron for collecting audit data from databases and log files, different policy files and configuration files from the OpenStack ecosystem, and log files from various virtual networking components such as Open vSwitch to fully capture the configuration. The data is then converted into a consistent format and missing correlation is reconstructed. The results are used to generate the code for the validation engine based on Sugar input language. The compliance validation engine performs the verification of the properties by feeding the generated code to Sugar. Finally, Sugar provides the results on whether the properties hold or not. Figure 3.6 illustrates the

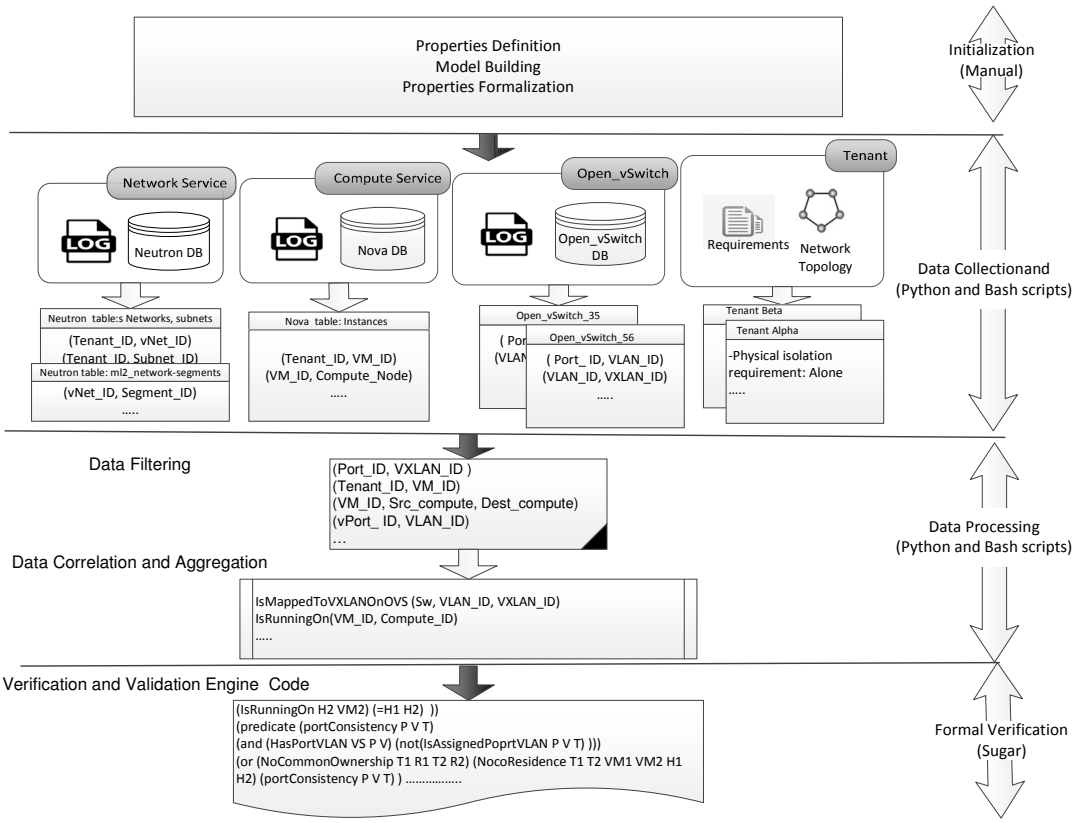


Figure 3.6: Our OpenStack-based auditing solution with the example of data collection, formating, correlation building and Sugar source generation

steps of our auditing process. In the following, we describe our implementation details along with the related challenges.

Data collection engine. We present hereafter different sources of data in OpenStack along with the current support for auditing offered by OpenStack and the virtual networking components. The main sources of audit data in OpenStack are logs, configuration files, and databases. Table 3.3 shows some sample data sources. The involved sources for auditing depend on the objective of the auditing task and the tackled properties. We use three different sources to audit configuration correctness of virtualized infrastructures:

- *OpenStack*. We rely on a collection of OpenStack databases, hosted in a MySQL server, that can be read using component-specific APIs such as Neutron APIs. For instance, in Nova database, table *Compute-node* contains information about the hosting

Relations	Sources of Data
BelongsTo	Table *Instances* in Nova database and *Routers, Subnets* and *Ports* in Neutron database, Neutron logs
DoesnotTrust	The tenant physical isolation requirement input
IsLocatedAt	Tables *Instances* in Nova database
IsAssignedPortVLAN	*Networks* in Nova database and *Ports* in Neutron database
HasPortVLAN	Open vSwitch instances located at various compute nodes
HasRunningVM	Table *Instances* in Nova database

Table 3.3: Sample data sources in OpenStack, Open vSwitch and tenants' requirements

machines such as the hypervisor's type and version, table *Instance* contains information about the project (tenant) and the hosting machine, table *Migration* contains migration events' related information such as the source-compute and the destination-compute. The Neutron database includes various information such as security groups and port mappings for different virtualization mechanisms.

- *Open vSwitch*. Flow tables and databases of Open vSwitch instances located in different compute nodes and in the controller node constitute another important source of audit data for checking whether there exist any discrepancies between the actual configuration and the OpenStack view.

- *Tenant policies*. We consider security policies expressed by the customers, such as physical isolation requirements. As expressing tenants' policies is out of the scope of this work, we assume that they are parsable XML files.

Data processing engine. Our data processing engine, which is implemented in Python, mainly retrieves necessary information from the collected data according to the targeted properties, recovers correlation from various sources, eliminates redundancies, converts it into appropriate formats, and finally generates the source code for Sugar.

- Firstly, for each property, our plug-in identifies the involved relations. The relations' support is either fetched directly from the collected data such as the support of the

relation *BelongsTo*, or recovered after correlation, as in the case of the relation *IsAssignedPortVLAN*.

- Secondly, our processing plug-in formats each group of data as an n-tuple, i.e., *(resource, tenant)*,*(port, vlan, tenant)*, etc.

- Finally, our plug-in uses the n-tuples to generate the portions of Sugar's source code, and append the code with the variable declarations, relationships and predicates for each security property (as discussed in Section 3.4). Different scripts are needed to generate Sugar source code for the verification of different properties.

Compliance Validation. The compliance validation engine is discussed in details in Section 3.4. In the following example, we discuss how our auditing framework can detect the violation of the no common ownership, no co-residence and port inconsistency security properties caused by the attack scenarios of our running example.

Example 3.6. *In this example, we describe how a violation of no common ownership, no co-residence and port-consistency properties may be caught by auditing.*

Firstly, our program collects data from different tables in the Nova and Neutron databases, and logs from different Open vSwitch instances. Then, the processing engine correlates and converts the collected data and represents it as tuples; for an example: (18038 10) (6100 11000) (512 6020 18033) where Port_84: 18038, Alpha: 10, VM_01: 6100, Open_vSwitch_56: 512, vPort_40: 18033 and VLAN_100: 6020. Additionally, the processing engine interprets each property and generates the associated Sugar source code (see Listing 3.1 for an excerpt of the code) using processed data and translated properties. Finally, Sugar is used to verify the security properties.

We show for each property how the violation is detected:

- *No common Ownership. The predicate CommonOwnership will evaluate to true if*

there exists a resource belonging to two different tenants. As Port_84 has been created by Beta, BelongsTo(Port_84, Beta) evaluates to true based on collected data from Neutron logs. Port_84 is defined on Alpha's router, hence, BelongsTo(Port_84, Alpha) evaluates to true based on collected data from Neutron database. Consequently, the predicate CommonOwnership evaluates to true. In this case, the output of sugar (SAT) is the solution of the problem, (r1 = 18038; r2 =18038; t1 =10; t2=11), which is actually the proof that Port_84 violates the no common ownership property.

- *No co-residence. In our example (see Figure 3.3), the supports HasRunningVM((VM_02, Alpha)(VM_03, Beta)), IsLocatedAt((VM_02, Compute_Node_96)(VM_03,Compute_Node_96) and DoesNotTrust(Alpha, Beta), where VM_02:6101, VM_03:6102, and Compute_Node_96:11100, make the predicate evaluate to true meaning that the no co-residence property has been violated.*

- *Port-consistency. The predicate PortConsistency evaluates to true if there exists a discrepancy between the OpenStack view of the virtualized infrastructure and the actual configuration. The support HasPortVLAN(Open_vSwitch_56, vPort_40, VLAN_100) makes the predicate evaluate to true, as long as there is no tuple such that IsAssignedPortVLAN (Port, VLAN_100, Alpha) where Port maps to vPort_40:18033.*

Challenges. Checking the configuration correctness in virtualized environment requires considering logs generated by virtualization technologies at various levels, and checking that mappings are properly maintained over different layers. Unfortunately, OpenStack does not maintain such overlay details.

At the OpenStack level, ports are directly mapped to VXLAN IDs, whereas at the Open-vSwitch level, ports are mapped to VLAN tags and mappings between the VLAN tags and VXLAN IDs are maintained. To overcome this limit, we devised a script that generates logs

from all the Open vSwitch instances. The script recovers mappings between VLAN tags and the VXLAN IDs from the flow tables using the *ovs-ofctl* command line tool. Then, it recovers mappings between ports and VLAN tags from the Open-vSwitch database using the *ovs-vsctl* command line utility.

Checking the correct configuration of overlay networks requires correlating information collected both from Open vSwitch instances running on top of various compute nodes and the controller node, and data recovered from OpenStack databases. To this end, we extended our data processing plug-in to deduce correlation between data. For example, we infer the relation $(port\ vlan\ tenant)$ from the available relations $(vlan\ vxlan)$ recovered from Open vSwitch and $(port\ vxlan\ tenant)$ recovered from the Nova and Neutron databases. In our settings, we consider a ratio of 30 ports per tenant, which leads to 300,000 entries in the relation $(port\ vxlan\ tenant)$ for 10,000 tenants. The number of entries is considerably larger than the number of tenants, because a tenant may have several ports and virtual networks. As a consequence, with the increasing number of tenants, the size of this relation grows and the complexity of the correlation step also increases proportionally. Note that the correlation is required for several of our listed properties.

An auditing solution becomes less effective if all needed audit evidences are not collected properly. Therefore, to be comprehensive in our data collection process, we firstly check fields of all varieties of log files available in OpenStack, all configuration files and all Nova and Neutron database tables. Through this process, we identify all possible types of data with their sources.

3.6 Experiments

In this section, we discuss the performance of our auditing solution by measuring the execution time, memory, and CPU consumption.

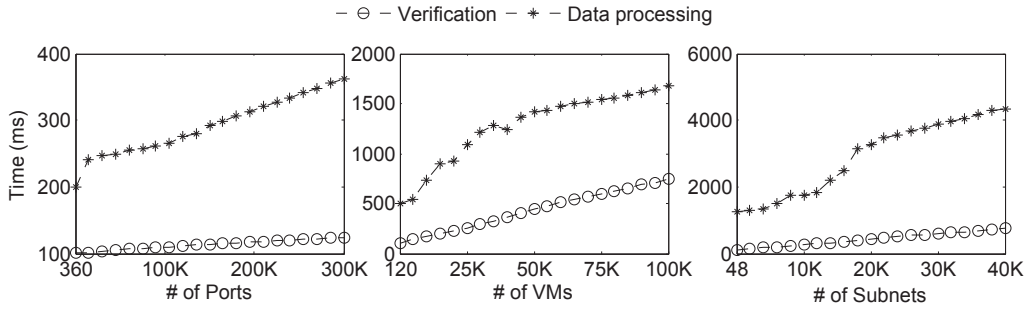
3.6.1 Experimental Setting

We deployed OpenStack with one controller node and three compute nodes, each having Intel i7 dual core CPU and 2GB memory running Ubuntu 14.04 server. Our OpenStack version is DevStack Juno (2014.2.2.dev3). We set up a testbed environment constituted of 10 tenants, 150 VMs and 17 routers. To stress the verification engine and assess the scalability of our approach, we furthermore simulated an environment with 10,000 tenants, 100,000 VMs, 40,000 subnets, 20,000 routers and 300,000 ports with a ratio of 10 VMs, 4 subnets, 2 routers and 30 ports per tenant. For the compliance verification, we use the V&V tool, Sugar V2.2.1 [84]. We conduct the experiments for 20 different audit trail datasets in total.

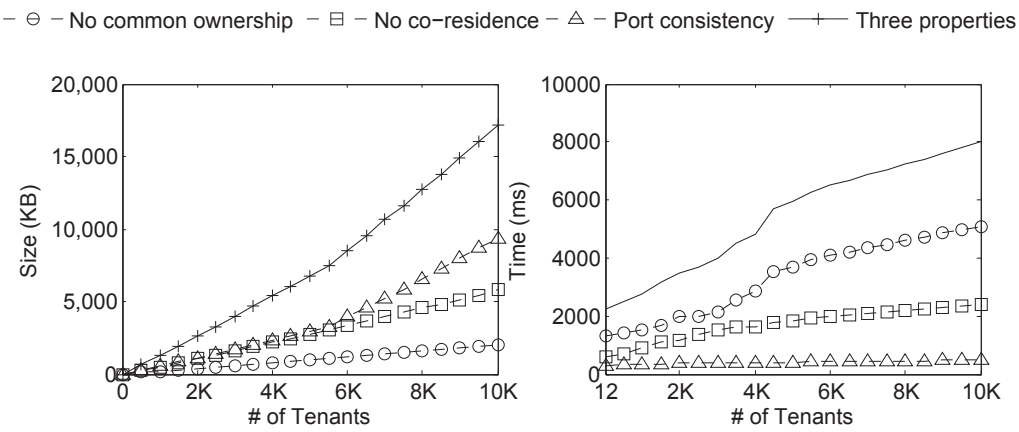
All data processing and V&V experiments are conducted on a PC with 3.40 GHz Intel Core i7 quad core CPU and 16 GB memory and we repeat each experiment $1,000$ times.

3.6.2 Results

The first set of our experiment (see Figure 3.7) demonstrates the time efficiency of our auditing solution. Figure 3.7(a) illustrates the time in milliseconds required for data processing and compliance verification steps for port consistency, no co-residence and no common-ownership properties. For each of the properties, we vary the most significant parameter (e.g., the number of ports, VMs and subnets for port consistency, no co-residence and no common ownership properties respectively) to assess the scalability of our auditing solution. Figure 3.7(b) (left) shows the size of the collected data in KB for auditing by varying the number of tenants. The collected data size reaches around 17MB for our largest dataset. We also estimate the time for collecting data as approximately 8 minutes for a fairly large cloud setup (10,000 tenants, 100,000 VMs, 300,000 ports, etc.). Note that data collection time heavily depends on the deployment options and the setup complexity. Moreover, the



(a) Time required for data processing and verification for the port consistency (left), no co-residence (middle) and no common ownership (right) by varying number of ports, VMs and subnets respectively.



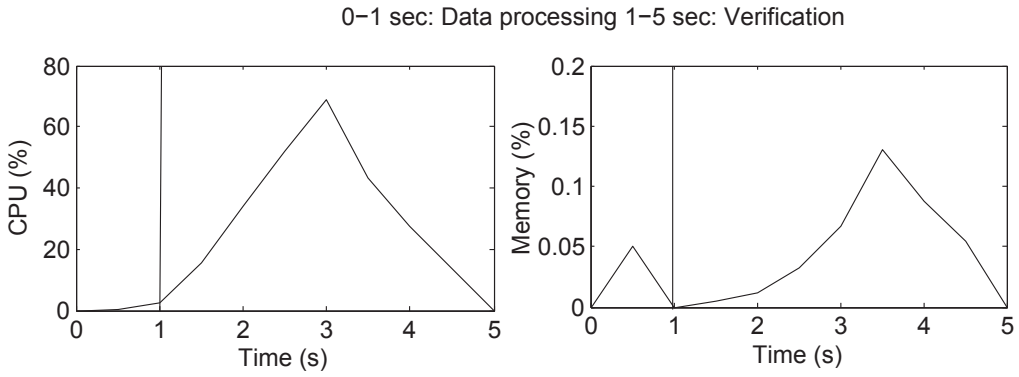
(b) Total size (left) of collected audit data and time required (right) for auditing port consistency, no co-residence, no common ownership and sequentially auditing three properties (worst case) by varying the number of tenants.

Figure 3.7: Execution time for each auditing step, total size of the collected audit data and total time for different properties using our auditing solution

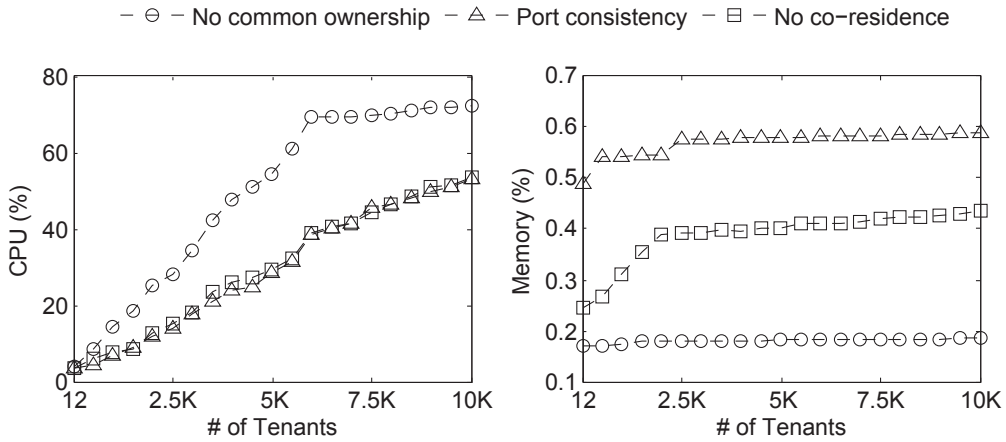
initial data collection step is performed only once for the auditing process (later on incremental collection will be performed at regular intervals), so the time may be considered reasonable. Figure 3.7(b) (right) shows the total execution time required for each property individually and in total. Auditing no common ownership property requires the longest time, because of the highest number of predicates used in the verification step; however, it finishes in less than 4 seconds. In total, the auditing of three properties completes within 8 seconds for the largest dataset, when properties are audited sequentially. However, since there is no interdependency between verifying different security properties, we can easily run parallel verification executions. The parallel execution of the verification step for different properties reduces the execution time to 4 seconds, the maximum verification time required among three security properties. Additionally, we can infer that the execution time is not a linear function of the number of security properties to be verified. Indeed, auditing more security properties would not lead to a significant increase in the execution time.

The objective of our second experiment (Figures 3.8(a)(left) and 3.8(b)(right)) is to measure the CPU usage (in %). In Figure 3.8(a)(left), we measure the peak CPU usage consumed by data processing and verification steps while auditing the no common ownership property. We notice that the average CPU usage is around 35% for the verification, whereas it is fairly negligible for the data processing step. According to Figure 3.8(b)(left), the CPU usage grows almost linearly with the number of tenants. However, the speed of increase varies depending on the property. It reaches a peak of over 70% for the no common ownership property for 10,000 tenants. This is due to the huge amount of tenant-specific resources (e.g., for 10,000 tenants the number of involved resources may reach $216,000$).

Our final experiment (Figures 3.8(a)(right) and 3.8(b)(left)) demonstrates the memory usage of our auditing solution. Figure 3.8(a)(right) shows that data processing step has a minor memory usage (with a peak of 0.05%), whereas the highest memory usage observed for the verification step for our largest setup is less than 0.19% of 16GB memory. Figure



(a) Peak CPU and memory usage for each step of our auditing solution over time when there are 10,000 tenants, 40,000 subnets, 100,000 VMs and 300,000 ports



(b) CPU (left) and memory (right) usage for each step of our auditing solution over time when there are 6,000 tenants, 24,000 subnets, 60,000 VMs and 180,000 ports

Figure 3.8: CPU and memory usage for each step and for different properties of our auditing solution over time

3.8(b) (right) shows that port consistency property has the lowest memory usage with a percentage of 0.2% whereas no common ownership has the highest memory usage, which is less than 0.6% for 10,000 tenants. Our observation from this experiment is that memory usage is related to the number of relations, variables and constraints involved to verify each property.

Discussion. In our experiments, we audited several security properties, e.g., no common ownership and port consistency, for up to $10,000$ tenants with a large set of various resources ($300,000$ ports, $100,000$ VMs, $40,000$ subnets) in less than 8 seconds. The auditing activity occurs upon request from the auditor (or in regular intervals when the auditor sets regular audits). Therefore, we consider the costs of our approach to be reasonable even for large data centers. Although we report results for a limited set of security properties related to virtualized cloud infrastructure, promising results show the potentiality of the use of formal methods for auditing. Particularly, we show that the time required for our auditing solution grows very slowly with the number of security properties. As seen in Figure 3.7(a), we anticipate that auditing a large list of security properties in practice would still be realistic. The cost generally increases almost linearly with the number of tenants.

Note that, we conduct our experiments in a single PC; if the security properties can be verified through concurrent and independent Sugar executions, we can easily parallelize this task by running several instances of Sugar on different VMs in the cloud environment. Thus the parallelization in the cloud allows to reduce the overall verification time to the maximum time for any individual security property.

3.7 Summary

In this work, we elaborated a generic model for virtualized infrastructures in the cloud. We identified a set of relevant structural security properties to audit and mapped them to different standards. Then, we presented a formal approach for auditing cloud virtualized infrastructures from the structural point of view. Particularly, we showed that our approach is able to detect topology inconsistencies that may occur between multiple control layers in the cloud. Our evaluation results show that formal methods can be successfully applied for large data centers with a reasonable overhead.

Chapter 4

ISOTOP: Auditing Virtual Networks Isolation Across Cloud Layers in OpenStack

4.1 Introduction

Despite the abundant benefits of the cloud, security and privacy concerns are still holding back its widespread adoption [87]. Particularly, multi-tenancy in cloud environments, supported by virtualization, allows optimal and cost-effective resource sharing among tenants that do not necessarily trust each other. Furthermore, the highly dynamic, elastic, and self-service nature of the cloud, introduces additional operational complexity that may prepare the floor for misconfigurations and vulnerabilities, leading to violations of baseline security and non-compliance with security standards (e.g., ISO 27002/27017 [73, 8] and CCM 3.0.1 [7]). Particularly, network isolation failures are among the foremost security concerns in the cloud [17, 18]. For instance, virtual machines (VMs) belonging to different corporations and trust levels may share the same set of resources, which opens up opportunities

for inter-tenant isolation breaches [66]. Consequently, cloud tenants may raise questions like: "How to make sure that all my virtual resources and private networks are properly isolated from other tenants' networks, especially my competitors? Are my vertical Network Segments (e.g., for finance, human resources, etc.) properly segregated from each other?".

Security auditing aims at verifying that the implemented mechanisms are actually providing the expected security features. However, auditing security without suitable automated tools could be practically infeasible due to the design complexity and the sheer size of the cloud as motivated in the following example.

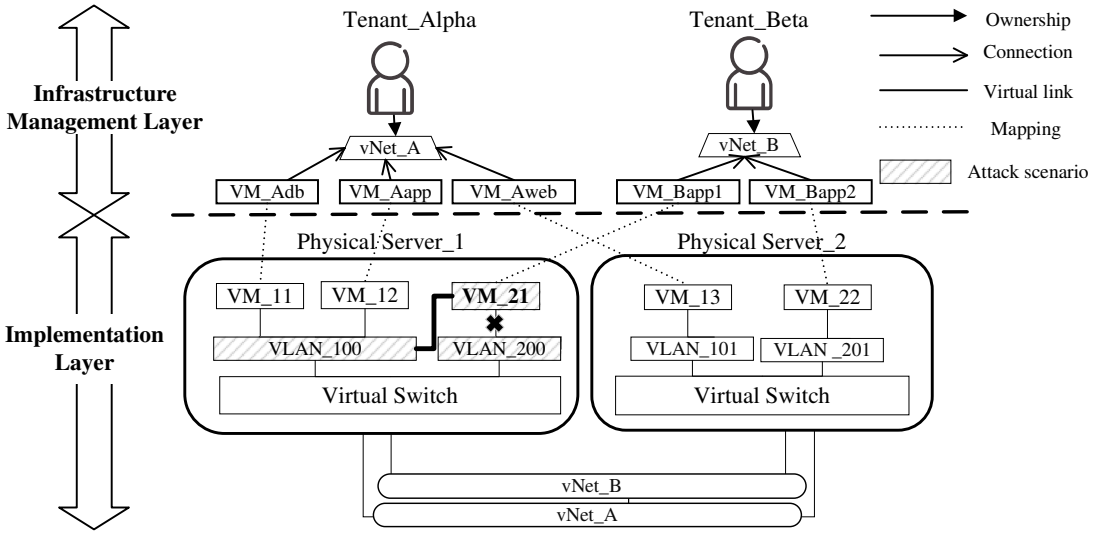


Figure 4.1: A two-layer view of a multi-tenant cloud virtualized infrastructure: The infrastructure management layer and the implementation layer

Motivating Example. Figure 4.1 illustrates a simplified view of an OpenStack [16] configuration example for virtualized multi-tenant cloud environments. Following a layered architecture [88], the cloud stack includes an infrastructure management layer responsible of provisioning, interconnecting, and decommissioning a set of virtual resources belonging to different tenants, at the implementation layer, across distributed physical resources. For instance, at the infrastructure management layer, virtual machines `VM_Adb`

and VM_Bapp1, are defined in separate Virtual Networks, vNet_A and vNet_B belonging to Tenant_Alpha and Tenant_Beta, respectively. At the implementation layer, these VMs are instantiated on Physical Server_1 as VM_11 and VM_21 and are interconnected to form those virtual networks. As the latter networks share the same physical substrate, network isolation mechanisms are defined at the management layer and configured at the implementation layer through network virtualization mechanisms to ensure their logical segregation. For instance, Virtual Local Area Network (VLAN) is used to isolate different virtual networks at the host level (more details are provided in Section 4.2.1). To audit isolation as defined in applicable standards, there exist several challenges.

- The gap between the high-level description of the requirements in the standards and the actual security properties hinders auditing automation. For instance, the requirement on segregation in networks in ISO 27017 [8] recommends *"separation of multi-tenant cloud service customer environments"*. Stated as such, these requirements do not detail exactly what data to be checked or how it should be verified.

- The layered nature of the cloud stack and the dependencies between layers make existing approaches that separately verify each single layer ineffective. Those layers maintain different but complementary views of the virtual infrastructure and current isolation mechanisms configurations. For instance, assume Tenant_Beta compromises the hypervisor on Physical Server_1 (e.g., by exploiting some vulnerabilities [78]) and succeeds to directly modify VLAN_200 associated with VM_21 to become VLAN_100 that is currently associated with VM_11 and VM_12 on Physical Server_1. This leads to a topology isolation breach as both VMs will become part of the same Layer 2 virtual network defined for vNet_A, opening the door for further attacks [89]. The verification of the management layer view cannot detect such a breach as VLAN tags are managed locally at the implementation layer. Additionally, verifying the implementation layer only without mapping the virtual

resources to their owners (maintained only at the management layer), would not allow a per-tenant identification of the breached resource. For example, the association between VM_Bapp1, vNet_B and their owner (Tenant_Beta) in the management layer view should be consistently mapped into the association between VM_21 in Physical Server_1 with VLAN_200 at the implementation level. This should be done for all tenants. Considering the implementation layer after the attack in Figure 4.1, VM_11, VM_12 and VM_21 in Physical Server_1 can be identified to be on the same VLAN, namely, VLAN_100. However, without considering that the corresponding VMs at the management layer are in different virtual networks and belong to different tenants, the breach cannot be properly detected.

- Correctly identifying the relevant data and their sources in the cloud for each security requirement increases the complexity of auditing. This can be amplified with the diversity and plurality of data sources located at different cloud stack layers. Furthermore, the data should not be collected only from different layers but also from different physical servers. In addition, their underlying semantics and relationships should be properly understood to be able to process it. The relation of this data and its semantics to the verified property constitutes a real challenge in automating cloud auditing.

In summary, taking into account the complexity factor and multi-layered nature of the cloud, the majority of existing approaches (e.g., [41, 37]) are not designed to handle cross-layer consistent isolation verification. Thus, in this work, we propose an automated cross-layer approach that tackles the above issues for auditing isolation requirements between virtual networks in a multi-tenant cloud. We focus on isolation at Layer 2 Virtual Networks and Overlay Networks, namely topology isolation, which is the basic building block for networks communication and segregation for upper network layers. To the best of our knowledge, this is the first effort on auditing cloud infrastructure isolation at layer 2 virtual

networks and overlay taking into account cross-layer consistency in the cloud stack. The following summarizes our main contributions:

- To fill the gap between standards and isolation verification, we devise a set of concrete security properties based on the literature and common knowledge on layer 2 virtual networks isolation and relate them to relevant requirements in security standards.

- To identify the relevant data for auditing network isolation and capture its underlying semantics across multiple layers, we elaborate a model capturing the cloud-stack layers and the verified network layers along with their inter-dependencies and isolation mechanisms. To the best of our knowledge, we are the first to propose such a model.

- We propose an off-line verification approach that spans the OpenStack implementation and management layers, which allows to evaluate the consistency of layer 2 virtual network isolation. We rely on the model defined above as input to our approach and a Constraint Satisfaction Problem (CSP) solver, namely, Sugar [84], as a back-end verification tool.

- We report real-life experience and challenges faced when integrating our auditing and compliance validation solution into OpenStack. We further conduct experiments to demonstrate the applicability of our approach.

4.2 Models

In this section, we provide a background on the network isolation mechanisms considered in this work, and we present the threat model followed by our model that captures tenants' virtual networks at the infrastructure management and implementation layers.

4.2.1 Preliminaries

In this work, we focus on layer 2 virtual networks deployed in cloud environments managed by OpenStack. We furthermore consider Open vSwitch (OVS)[1] for providing layer 2 network function to guest VMs at the host level [90].

In large scale OpenStack-based cloud infrastructures, layer 2 virtual networks are implemented on the same server using Virtual LANs (VLAN), and across the physical network through Virtual Extended LAN (VXLAN) as an overlay technology. The VXLAN technology is used to overcome the scale limitation of VLANs, which only allows for a maximum of 4,096 tags [18]. More specifically, on each physical server, disjoint VLAN tags are assigned to ports connecting VMs that are part of different isolated virtual networks. Furthermore, a unique VXLAN identifier is assigned per isolated virtual network in order to extend layer 2 virtual networks between different physical servers, thus forming an overlay network. When the traffic leaves a VM (or a physical server), the appropriate VLAN tag (or VXLAN identifier) is inserted into the traffic by configurable OVS forwarding rules to maintain proper layer 2 traffic isolation. The mapping between VLAN tags and VXLAN identifiers performed by the OVS rules ensures that the traffic is smoothly steered between sources and destinations deployed over different physical servers.

Example 4.1. *Figure 4.2 illustrates a more detailed view of layer 2 virtual networks implementation for the configuration showed in Figure 4.1. According to the latter figure,* `VM_11`*,* `VM_12` *and* `VM_13` *belong to* `Tenant_Alpha` *and are connected to* `vNet_A`*.* `VLAN_100` *is defined at* `Physical Server_1` *to enable isolated layer 2 communication between* `VM_11` *and* `VM_12`*, whereas* `VLAN_200` *is defined to isolate* `VM_21` *at the same physical server since the latter VM is connected to another virtual network (*`vNet_B`*). Similarly, at* `Physical Server_2`*, different VLAN tags, namely,* `VLAN_101` *and* `VLAN_201`*, are*

[1]Open vSwitch OVS is one of the mostly used OpenFlow-enabled Virtual Switches in more than 30% deployments, and is compatible with most hypervisors including Xen, KVM and VMware.

defined to isolate VM_13 *and* VM_22 *respectively since they are connected to different networks. Since* VM_11, VM_12 *and* VM_13 *are all connected to the same virtual network (see Figure 4.1) but deployed over two different physical servers, VXLAN is used as an overlay protocol to logically connect VMs across physical servers while ensuring isolation. To this end, two distinct VXLAN identifiers, namely,* VXLAN_0×100 *and* VXLAN_0×200*, are associated to* vNet_A *and* vNet_B*, respectively. Then, to achieve end to end isolation,* VXLAN_0×100 *is attached to* VLAN_100 *on* Physical Server_1 *and to* VLAN_101 *on* Physical Server_2*, while* VXLAN_0×200 *is attached to* VLAN_200 *on* Physical Server_1 *and to* VLAN_201 *on* Physical Server_2*. This would allow to isolate the virtual networks both at the host level (through different VLAN tags) and at the physical network level (through different VXLAN identifiers).*

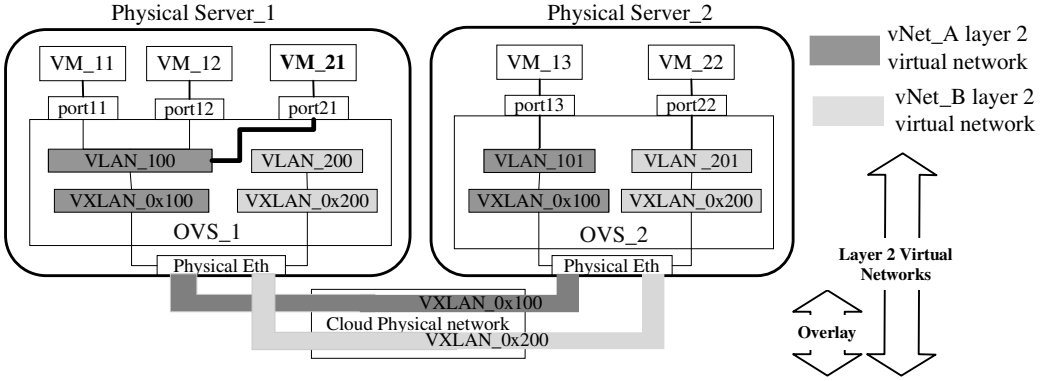


Figure 4.2: A detailed view of the implementation layer of Figure 4.1

4.2.2 Threat Model

We assume that the cloud infrastructure management system has implementation flaws and vulnerabilities, which can be potentially exploited by malicious entities leading to tenants' virtual infrastructures isolation failures. For instance, a reported vulnerability in OpenStack Neutron OSSA-2014-008 [91] allows a tenant to create a virtual port on another tenant's

virtual router without checking his identity. Exploiting such vulnerabilities leads to serious isolation breaches opening doors to more harmful attacks such as network sniffing. As another example, a malicious tenant can take advantage from the known cloud data centers configuration strategies to locate his victim inside the cloud [66]. In addition, he can compromise some host hypervisors to deliberately change network configurations at the implementation layer.

Our auditing approach focuses on verifying security compliance of OpenStack-managed cloud infrastructures with respect to predefined security properties related to virtual infrastructure isolation defined in relevant security standards or tenant specific requirements. Thus, our solution is not designed to replace intrusion detection systems or vulnerability analysis tools (e.g., vulnerability scanners). However, by verifying security properties, our solution may detect the effects and consequences of certain vulnerabilities exploit or threats on the configuration of the cloud under the following conditions: *a*) the vulnerability exploit or threat violates at least one of the security properties being audited, *b*) the violations generate logged events and configuration data, *c*) the corresponding traces of those violations in logs and configuration data are intact and not erased or tampered with, as the correctness of our audit results depends on the correct input data extracted from logs, databases, and devices.

The out of scope threats include attacks that do not violate the specified security properties, attacks not captured in the logs or databases, and attacks through which the attackers may remove or tamper with logged events. Existing techniques on trusted auditing may be applied to establish a chain of trust from TPM chips to auditing components, e.g., [71]).

We focus on layer 2 virtual network, and our work is complementary to existing solutions at other network layers (e.g., TenantGuard [92]). We assume the verification results do not disclose sensitive information about other tenants and regard potential privacy issues as a future work.

Finally, we focus on auditing structural properties such as the assignment of instances to physical hosts, the proper configuration of virtualization mechanisms, and consistency of the configurations in different layers of the cloud. Those properties mainly involve static configuration information that are already stored by the cloud system at the cloud management layer and the implementation layer. The verification of operational properties, which are related to the network forwarding functionality, are out of the scope of this work.

4.2.3 Virtualized Cloud Infrastructure Model

In this section, we present the two-layered model that we derive to capture information related to isolated virtual networks at both the infrastructure management and the implementation layers. This model was derived based on common knowledge and studied literature on implementation and management of isolated virtual networks [93]. For instance, to elaborate and validate the infrastructure management layer model, we analyzed the abstractions exposed by the most popular cloud platforms providing tenants the capability to build virtual private networks (e.g., AWS EC2- Virtual Private Cloud (VPC) [11], Google Cloud Platform (GCP) [12], Microsoft Azure [13], VMware virtual Cloud Director (vCD) [94] and OpenStack [16]). More details will be provided in Table 4.7 (Section 4.6). For the implementation model, we relied on performing extensive tests on OpenStack compute and network nodes, then we supported our understanding by exploring the literature [95, 18]. Finally, we validated our two-layer model with subject matter experts.

The model allows capturing the data to be audited at each layer, its underlying semantics and relation with isolation requirements. It also defines cross-layer mappings of data in different layers to capture consistency requirements.

Infrastructure Management Model. The upper model in Figure 4.3 captures the view from the cloud infrastructure management system perspective. This layer manages virtual resources such VMs, routers, and virtual networks (represented as entities) as well as their

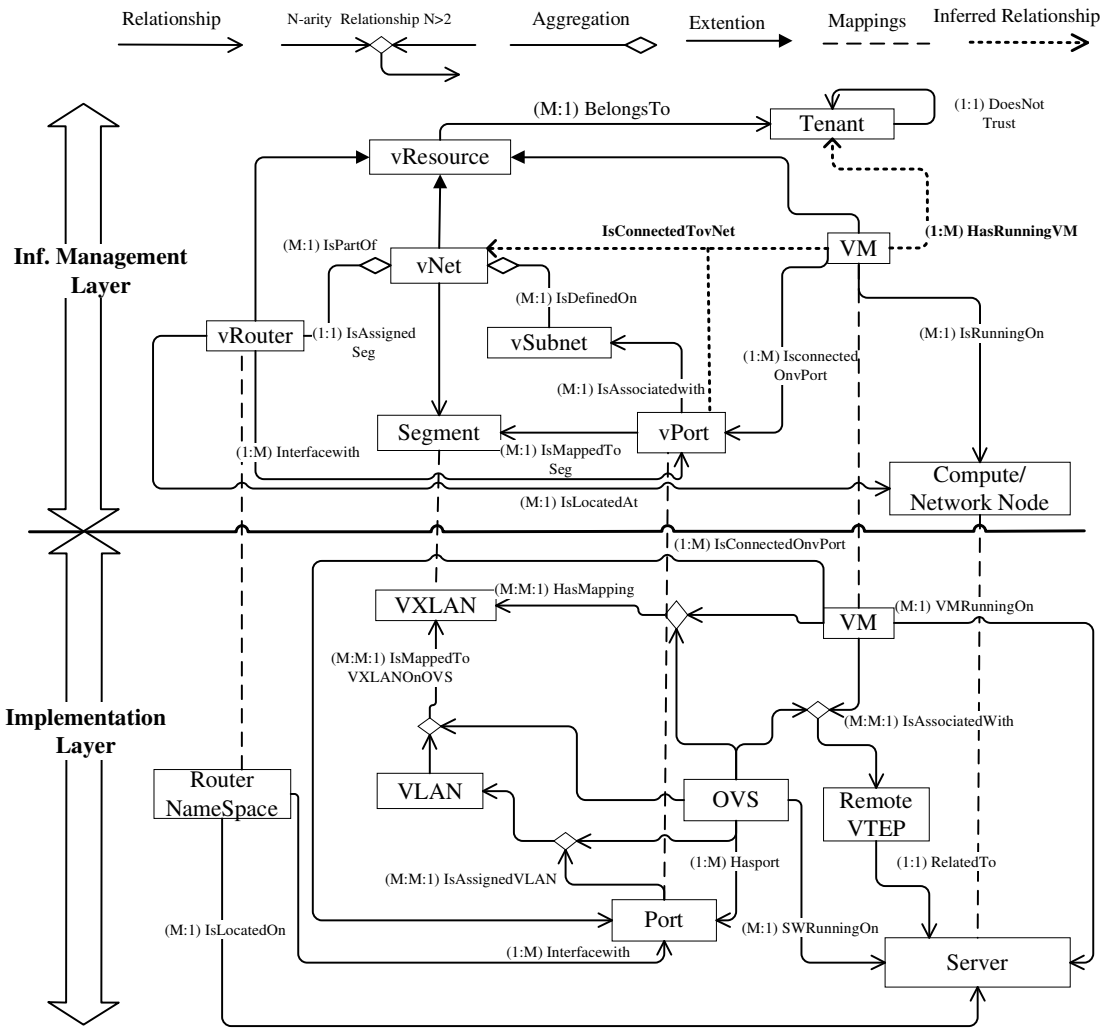


Figure 4.3: Two-layered model for isolated multi-tenant virtualized infrastructures in the cloud: Generic model for the infrastructure management layer (upper model) mapped into an implementation-specific model of the infrastructure layer (lower model)

ownership relation (represented as relationships) with respect to tenants. Once connected together, these resources form the tenants' virtual infrastructures. Some entities, for instance `Tenant`, are only maintained at the management layer and have no counterpart at the lower layer. Other entities exist across layers (e.g., VMs and ports), however, one-to-one mappings should be maintained. These mappings allow inferring missing relationships between layers and help checking consistency between the cloud stack layers. Isolation between different virtual networks at this layer is defined using a segmentation mechanism, modeled as entity `Segment`. A segment should be unique for all elements of the same virtual infrastructure.

Example 4.2. *Ownership is modeled using the* `BelongsTo` *relationship in Figure 4.3 between* `Tenant` *and* `vResource`*. The related cardinality constraint* `(M:1)`*, expresses that, following the directed edge, a given* `vResource` *can only belong to a single (i.e.,* 1*)* `Tenant`*, but, a* `Tenant` *can own multiple (i.e., M) virtual resources. The* `isAssignedSeg` *relationship and its cardinality constraint* `(1:1)` *relating* `Segment` *to* `vNet` *allows having a unique segment per network. Relationships* `isConnectToVnet` *and* `HasRunningVM` *are of special interest to us and thus they are depicted in the model even though they can be inferred from other relationships.*

Implementation Model. The lower model in Figure 4.3 captures a typical OpenStack implementation of the infrastructure management view using well-known layer 2 isolation technologies, VXLAN and VLAN. The model can capture other layer 2 isolation mechanisms such as Generic Routing Encapsulation (GRE) by replacing the entity VXLAN with entity GRE. Some entities and relationships in this model represent the implementation of their counterparts at the management model. For instance, VXLAN combined with VLAN are implementation of entity `Segment`. Other entities such as virtual networking devices Open vSwitch (OVS) and Virtual Tunneling End Point (VTEP) are specific to the implementation layer as they do not exist at the infrastructure management model. They play

the vital role in connecting VM instances to their hosting machines and to their virtual networks across different servers. Indeed, VTEPs are overlay-aware interfaces responsible for the encapsulation of packets with the right tunnel header depending on the destination VM and its current hosting server.

Example 4.3. *At the lower model in Figure 4.3, the ternary relationship* `isAssignedVLAN` *with cardinality* `(M:M:1)` *means that each single* `port` *in a given* `OVS` *can be assigned at most one* `VLAN` *but multiple* `ports` *can be assigned the same* `VLAN`*. To capture isolation at overlay networks spanning over different servers, the ternary relationship* `isMappedtoVXLAN` *states that each VLAN in each OVS is mapped to a unique VXLAN. The unicity between a specific port and a VLAN in an OVS as well as the unicity of the mapping of a VLAN to a VXLAN in a given OVS, are inherited from the unicity of the mapping of a segment to a virtual network. The two ternary relationships* `hasMapping` *and* `isAssociatedWith` *are used to model VTEPs information existing over different physical servers. Several relations have similar semantics in both models, however, we use different names for clarity. For instance,* `VMRunningOn` *at the implementation layer corresponds to* `isRunningOn` *at the management layer.*

Entities and relationships defined in these models will be used in our approach to automate the verification of isolation between tenants' virtual infrastructures. They will be essentially used to express system data and the relations among them in the form of instances of these models. Also, they will be used to express properties related to isolation as will be presented in next section.

4.3 Methodology

In this section, we detail our approach for auditing compliance of virtual layer 2 networks with respect to a multi-tenant cloud.

4.3.1 Overview

Figure 4.4 presents an overview of our approach. Our main idea is to use the derived two-layered model (Section 4.2) to capture the implementation of the multi-tenant virtual infrastructure along with its specification. We then verify the implementation against its specification to detect violation of the properties.

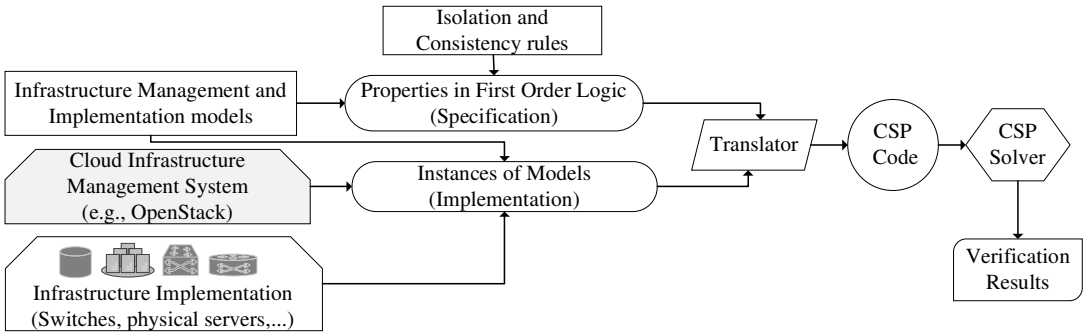


Figure 4.4: An overview of our verification approach

To be able to automatically process the model as the specification support for the virtual infrastructure, we first express it in First Order Logic (FOL) [85]. We encode entities and relationships in both models into a set of FOL expressions, namely, variables and relations. We also express isolation and consistency rules as FOL predicates based on the FOL expressions derived from the model. This process is performed offline and only once.

To obtain the implementation of the system, we collect real data from different layers (cloud management and cloud infrastructure) and use the model entities and relationships definitions to build an instance of the model representing the current state of the system. As we aim at detecting violations, we represent relationships between real data as instances of FOL n-ary relations without restricting instances to meet cardinality constraints. This will be detailed later on in this section. As a back-end verification mechanism, we rely on the off-the-shelf CSP solver Sugar. The latter allows formulation of many complex problems in terms of variables defined over finite domains and constraints. Its generic goal is to find a vector of values (a.k.a. assignment) that satisfies all constraints expressed

over the variables. If all constraints are satisfied, the solver returns SAT, otherwise, it returns UNSAT. In the case of a SAT result, a solution to the problem, which is a specific assignment of values to the variables that satisfies the constraints, is provided. One of the key advantages of using constraint solving is to enable uniformly specifying systems data and properties in a clean formalism and covering a wide range of properties [86]. Furthermore, the latter allows to identify the data violating the verified properties as it will be explained in Section 4.3.3.

4.3.2 Cloud Auditing Properties

Among the goals of this work is to establish a bridge between high-level security standards and low-level implementation as well as to enable verification automation. Therefore, this section describes a set of concrete security properties related to layer 2 virtual network and overlay network isolation in a multi-tenant environment. In this work, we focus on the verification of structural properties gathered from the literature and the subject matter. To have a more concrete example of layer 2 virtual network isolation mechanisms, we refer to VLAN and VXLAN as examples of well-established technologies.

Table 4.1 presents an excerpt of the security properties mapped to relevant domains and control classes in security standards, namely, CCM [7] (Infrastructure and virtualization security segmentation domain), ISO27017 [8] (Segregation in networks section) and NIST800 [81] (System and communications protection, System and information integrity security controls). Those properties either check topology isolation based on individual cloud layers (i.e., infrastructure management level or implementation level), or they check topology consistency based on information gathered from both layers at the same time. In the following, we discuss examples illustrating how those properties are related to isolation and consistency, and how they can be violated.

Topology Isolation. This property ensures that virtualization mechanisms are properly

configured and provide adequate logical isolation between virtual networks. By using isolated virtual topologies, traffic belonging to different virtual networks would travel on logically separated paths, thus ensuring traffic isolation. The following example illustrates a topology isolation violation using an instance of our model presented in Section 4.2.

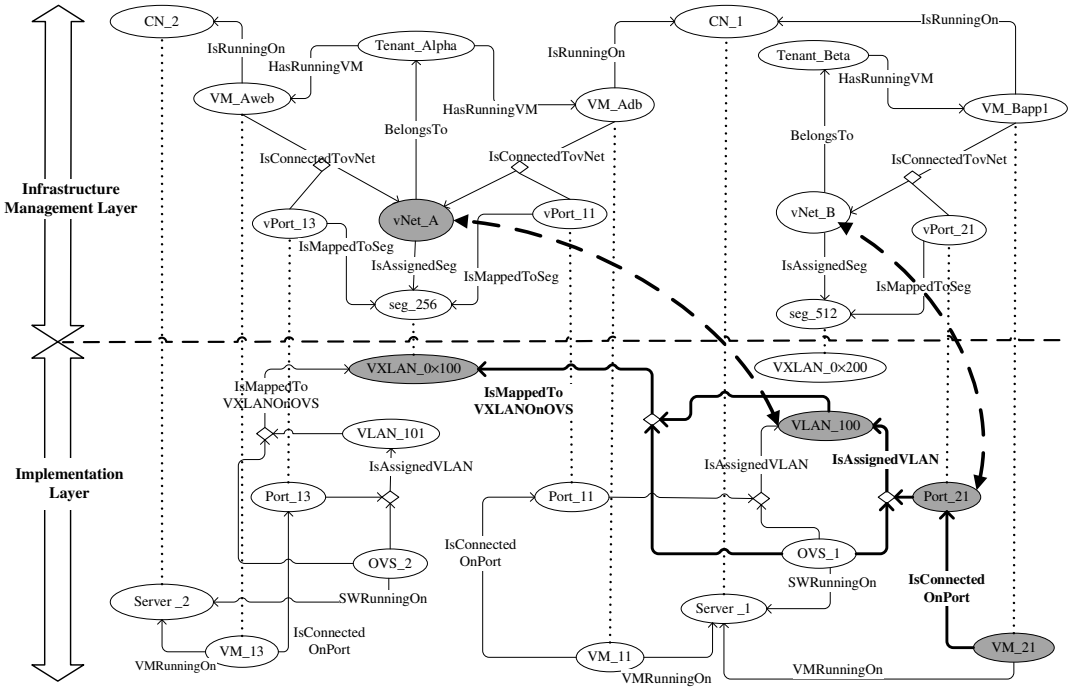


Figure 4.5: Subsets of data and its relations at the could infrastructure implementation and management layers showing isolation violation. At the implementation level, VM_21 is connected on Port_21, that is assigned VLAN_100 as a consequence of the attack. Since VLAN_100 is mapped to VXLAN 0×100, which is mapped to seg_256 at the infrastructure management layer and the latter segment is assigned to vNet_A of Tenant_Alpha, VM_21 belonging to Tenant_Beta is now on the same network segment as VMs in vNet_A

Example 4.4. *Figure 4.5 captures a subset of the data, at different layers, that is relevant to virtual networks* vNet_A *and* vNet_B *corresponding to the deployment illustrated in Figure 4.1 and Figure 4.2. The upper part of the figure shows a subset of the data managed by the infrastructure management layer and on the lower part, the subset of data managed by the implementation layer. Nodes represent data instances,*

while the directed arrows represent relations between these data instances. For example, at the infrastructure management layer, the relationship `IsConnectedTovNet` *relates three instances of data* `VM_Adb`*,* `vNet_A`*, and* `vPort_11`*, and means that* `VM_Adb` *is connected to* `vNet_A` *on virtual port* `vPort_11`*. A cross-layer mapping, shown as small dotted undirected arrows, between some of the data instances at different layers is used to relate management-defined data to its implementation counterpart. For instance,* `VM_Adb` *and* `vPort_11` *have each a one-to-one cross-layer mapping to* `VM_11` *and* `Port_11`*, respectively, while no data entity at the implementation layer could be directly mapped to* `vNet_A` *at the management layer. The latter can be indirectly mapped to* `VXLAN_0x100` *at the implementation layer via the segment* `seg_256`*. More precisely,* `vNet_A` *is implemented using* `VXLAN_0x100` *and a set of corresponding VLANs, namely,* `VLAN_100` *and* `VLAN_101` *(via* `IsMappedToVXLANonOVS`*), which are assigned to* `Port_11`*,* `Port_13`*, and* `Port_21` *(via* `IsAssignedVLAN`*).*

This instance of the layered-model allows capturing topology isolation breaches and identifying which networks, VMs, and tenants are in this situation. Indeed `VM_21` *is found to be on the same virtual layer 2 segment as* `VM_11` *and* `VM_13`*. There are two types of isolation breaches and they are illustrated as follows:*

- Intra-server topology isolation breach. *At the implementation layer,* `VM_21` *is connected on port* `Port_21` *(via relationship* `IsConnectedonPort`*), which is assigned* `VLAN_100` *(via relationship* `IsAssignedVLAN`*) in the open vSwitch* `OVS_1`*. Additionally, since* `Port_11` *connecting* `VM_11` *is also assigned* `VLAN_100` *on the same switch, both* `VM_11` *and* `VM_21` *connected via these ports are located on the same virtual network segment* `VLAN_100` *(which corresponds to vNet_A at the infrastructure management level) leading to an isolation breach. Since both VMs are in the same server, namely,* `Server_1`*, it is said to be an intra-server topology isolation at virtual layer 2. Noteworthy, without the correct mapping between* `VM_11`

and VM_21 *at the implementation layer to their respective counterparts* VM_Adb *and* VM_Bapp1 *as well as the ownership information (i.e., these VMs belong to different tenants and are connected on different virtual networks) at the management layer, we cannot conclude on the existence of this breach by only considering data from the implementation layer.*

- Inter-server topology isolation breach. *At the implementation layer,* VLAN_100 *that is assigned to ports* Port_21 *and* Port_11 *is mapped to* VXLAN_0x100 *via relationship* IsMAppedToVXLANonOVS *(which corresponds again to vNet_A at the infrastructure management level). However, this VXLAN identifier is also related to another VLAN_tag, namely,* VLAN_101*, which is assigned to port* Port_13 *connecting* VM_13 *on* Server_2*. This is an inter-server topology isolation breach, since* VM_13 *and* VM_21 *are running on different servers (*Physical Server_2 *and* Physical Server_1*).*

Topology Consistency. Topology consistency consists of checking whether the topology view in the cloud infrastructure management system, consistently matches the actual implemented topology, and the other way around, while considering different tenants' boundaries.

Example 4.5. *(Port consistency) Assume that a malicious insider deliberately created a port* Port_40 *directly on* OVS_1 *without passing by the cloud infrastructure management system and tagged it with* VLAN_100*, which is already assigned to* Tenant_Alpha*. This allows the malicious insider to sniff tenant's Alpha traffic on* VLAN_100 *via* Port_40*, which clearly leads to the violation of network isolation property.*

Category	Standard			Property		Level	
	CCM	ISO27017	NIST800	Name	Description	Mgmt.	Impl.
Topology isolation	•	•	•	Mappings unicity Virtual Networks-Segments (P1)	Virtual networks and segments should be mapped one-to-one	×	
				Mappings unicity Ports-Segments (P2)	vPorts should be mapped to unique segments	×	
				Correct association Ports-Virtual Networks (P3)	VMs should be attached to the virtual networks they are connected to through the right vPorts	×	
				Mapping unicity Ports-VLANs (P4)	Ports should be mapped to unique VLANs		×
				Mapping unicity VLANs-VXLANs (P5)	VLANs and VXLANs should be mapped one-to-one on a given server		×
				Overlay tunnels isolation (P6)	In each VTEP end, VMs are associated to their physical location and to the VXLAN assigned to the networks they are attached to		×
Topology consistency	•	•	•	VM location consistency (P7)	Consistency between VMs' locations at the implementation level and at the management level	×	×
				Ports consistency (P8)	Consistency between vPorts in the implementation level and their counterparts in the management level	×	×
				Virtual links consistency (P9)	VMs should be connected to the VLANs and VXLANs in the implementation level that correspond to the virtual networks they are attached to at the management level	×	×

Table 4.1: Excerpt of security properties

4.3.3 Verification Approach

In order to systematically verify isolation and consistency properties over the model, we need to transform the model and its instances as well as the requirements into FOL expressions that can be automatically processed. In the following, we present how we express the model, the data, and the properties in FOL.

Model and Data Representation

Entities in the model are encoded into FOL variables where their domains would encompass all instances defined by the system data. Each n-ary relationship is encoded into a FOL n-ary relation over the related variables, where the instance of a given relation is the set of tuples corresponding entities-instances as defined by the relationship.

For instance, in the model instance of Figure 4.5, the relationship `IsMappedToVXLANOnOVS` is translated into the following FOL relation instances capturing the actual implementation setup showing the mapping of a VLAN into a VXLAN on a given OVS instance.

- `IsMapedToVXLANOnOVS(OVS_2, VLAN_101, VXLAN_0×100)`
- `IsMapedToVXLANOnOVS(OVS_1, VLAN_100, VXLAN_0×100)`

Table 4.2 shows the main FOL relations defined in our model. These relations are required for expressing properties, which are formed as predicates as it will be presented next.

Properties Expressions

Security properties presented in Table 4.1 can be expressed as FOL predicates over FOL relations defined in Table 4.2.

Relations	Def. at	Evaluate to *True* if
$IsRunningOn(vm, cn)$	Mgmt.	The instance *vm* is located at the compute node *cn*
$IsMappedToSeg(vp, seg)$	Mgmt.	The virtual port *vp* is mapped to the segment *seg*
$IsAssignedSeg(vNet, seg)$	Mgmt.	The virtual network *vNet* is assigned the segment *seg*
$IsConnectedTovNet(vm, vNet, vp)$	Mgmt.	*vm* is connect to *vNet* on the virtual port*vp*
$HasPort(sw, p)$	Impl.	The virtual switch *sw* has a port*p*
$IsAssignedVLAN(sw, p, vlan)$	Impl.	The port*p* on switch *sw* is assigned the VLAN *vlan*
$IsMappedToVXLANOnOVS(sw, vlan, vxlan)$	Impl.	*vlan* is mapped to *vxlan* on the virtual switch *sw*
$SwRunningOn(sw, s)$	Impl.	The switch *sw* is running on the server *s*
$VMRunningOn(vm, s)$	Impl.	The VM *vm* is running on the server *s*
$IsConnectedOnPort(vm, sw, p)$	Impl.	The VM *vm* is connected on port *p* belonging to the switch *sw*
$HasMapping(ovs, vm, vxlan)$	Impl.	The VM *vm* is associated to *vxlan* on a remote switch *ovs*
$IsAssociatedWith(ovs, vm, vtep)$	Impl.	The VM *vm* is associated to the remote VTEP *vtep* on *ovs*
$IsRelatedTo(vtep, s)$	Impl.	the VTEP *vtep* is defined on the server *s*

Table 4.2: Model relations encoded in FOL

Properties	FOL Expressions
Mappings unicity Virtual Networks and Segments (P1)	$\forall vNet1, vNet2 \in vNET, \forall seg1, seg2 \in Segment : [IsAssignedSeg(vNet1, seg1) \wedge IsAssignedSeg(vNet2, seg2) \wedge \neg(vNet1 = vNet2) \rightarrow \neg(seg1 = seg2)] \wedge [IsAssignedSeg(vNet1, seg1) \wedge IsAssignedSeg(vNet2, seg2) \wedge \neg(seg1 = seg2) \rightarrow \neg(vNet1 = vNet2)]$
Mappings unicity Ports-Segments (P2)	$\forall seg1, seg2 \in Segment, \forall vp \in vPORT : IsMappedToSeg(vp, seg1) \wedge IsMappedToSeg(vp, seg2) \rightarrow (seg1 = seg2)$
Correct association Ports-Virtual Networks (P3)	$\forall vm \in VM, \forall vNet \in vNET, \forall seg1, seg2 \in Segment, \forall vp \in vPort : IsConnectedTovNet(vm, vNet, vp) \wedge IsAssignedSeg(vNet, seg1) \wedge IsMappedToSeg(vp, seg2) \rightarrow (seg1 = seg2)$

Table 4.3: Isolation properties at the infrastructure management level in FOL

Properties	FOL Expressions
Mapping unicity Ports-VLANs (P4)	$\forall sw \in OVS, \forall p \in Port, \forall vlan1, vlan2 \in VLAN : HasPort(sw,p) \wedge$ $IsAssignedVLAN(sw,p,vlan1) \wedge IsAssignedVLAN(sw,p,vlan2) \rightarrow$ $(vlan1 = vlan2)$
Mapping unicity VLANs-VXLANs (P5)	$\forall vxlan1, vxlan2 \in VXLAN, \forall vlan \in VLAN, \forall sw \in OVS,$ $\forall p \in PORT : (IsAssignedVLAN(sw,p,vlan) \wedge IsMappedToVXLANOnOVS(sw,vlan,vxlan1)$ $\wedge IsMappedToVXLANOnOVS(sw,vlan,vxlan2) \rightarrow (vxlan1 = vxlan2)$
Overlay tunnels isolation (P6)	$\forall vm \in VM, \forall sw1, sw2 \in OVS, \forall p \in PORT, \forall vxlan1, vxlan2 \in VXLAN, \forall s1, s2 \in Server,$ $\forall vtep \in RemoteVTEP, \forall vlan \in VLAN : HasPort(sw1,p) \wedge SWRunningOn(sw1,s1) \wedge$ $IsConnectedOnPort(vm,sw1,p) \wedge IsAssignedVLAN(sw1,p,vlan) \wedge$ $IsMappedToVXLANOnOVS(sw1,vlan,vxlan1) \wedge IsAssociatedWith(sw2,vm,vtep)$ $\wedge HasMapping(sw2,vm,vxlan2) \wedge IsRelatedTo(vtep,s2) \rightarrow (s1 = s2) \wedge (vxlan1 = vxlan2)$

Table 4.4: Isolation properties at the implementation level in FOL

Properties	FOL Expressions
VM location consistency (P7)	$\forall vm1 \in VM, \forall cn \in COMPUTEN : IsRunningOn((vm1, cn) \rightarrow$ $\exists vm2 \in iVM, \exists s \in SERVER : VMRunningOn(vm2, s) \wedge (vm1 = vm2) \wedge (cn = s)$
Ports consistency (P8)	$\forall vNet \in vNET, \forall seg \in Segment, \forall vp \in vPORT : IsAssignedSeg(vNet, seg)$ $\wedge IsMappedToSeg(vp, seg) \rightarrow$ $[\exists sw \in OVS, \exists vxlan \in VXLAN, \exists vlan \in VLAN, \exists p \in PORT : IsAssignedVLAN(sw, p, vlan)$ $\wedge IsMappedToVXLANOnOVS(sw, vlan, vxlan) \wedge (seg = vxlan) \wedge (vp = p)]$
Virtual links consistency (P9)	$\forall vm1 \in iVM, \forall vxlan \in VXLAN, \forall sw \in OVS, \forall vlan \in VLAN, \forall p \in PORT :$ $IsConnectedOnPort(vm1, sw, p) \wedge$ $IsAssignedVLAN(sw, p, vlan) \wedge IsMappedToVXLANOnOVS(sw, vlan, vxlan) \rightarrow$ $[\exists vm2 \in vVM, \exists vNet \in vNET, \exists seg \in Segment, \exists vp \in vPORT :$ $IsConnectedTovNet(vm2, vNet, vp) \wedge (vm1 = vm2) \wedge$ $IsAssignedSeg(vNet, seg) \wedge (seg = vxlan)]$

Table 4.5: Topology consistency properties in FOL

Table 4.3 shows FOL predicates for the isolation properties at the infrastructure management model. Table 4.4 presents FOL predicates for the isolation properties at the implementation model. Table 4.5 summarizes the expressions of consistency-related properties.

Isolation Verification

As discussed before (Section 4.3.3), model instances are built based on the collected data and they are encoded as tuples of data representing relations' instances. On another hand, properties are encoded as predicates to specify the conditions that these relations' instances should meet.

To verify the security properties, we use both properties' predicates and relations' instances to formulate the CSP constraints to be fed into the CSP solver. Since CSP solvers provide solutions only in case the constraint is satisfied (SAT), we define constraints using the negative form of the FOL predicates presented in Tables 4.3, 4.4 and 4.5. Hence, the solution provided by the CSP solver gives the relations' instances for which the negative form of the property is satisfied, meaning that a violation has occurred.

To better explain how the CSP solver allows to obtain the violation evidence, we provide hereafter an example of the verification of the inter-server isolation property provided in Example 4.4.

Example 4.6. *We assume that VM location consistency and port consistency properties were verified to be met by the configuration. From the infrastructure management level, we recover the virtual networks connecting each VM and their corresponding segment. This is captured through the following relation instances:*

- `IsConnectedTovNet((VM_Bapp1, vNet_B, vPort_21), (VM_Adb, vNet_A, vPort_11), (VM_Aweb, vNet_A, vPort_13))`
- `IsAssignedSeg((vNet_B, seg_512),(vNet_A, seg_256)))`

From the implementation level, we recover the OVS and the ports connecting VMs in addition to their assigned VLAN tags and VXLAN identifiers captured through the following relation instances:

- IsConnectedOnPort((VM_21, OVS_1, Port_21),(VM_11, OVS_1, Port_11), (VM_13, OVS_2, Port_13))
- IsAssignedVLAN((OVS_1, Port_21, vlan_100),(OVS_1, Port_11, vlan_100), (OVS_2, Port_13, vlan_101))
- IsMappedToVXLANOnOVS((OVS_1, vlan_100, vxlan_0×100),(OVS_2, vlan_101, vxlan_0×100))

We would like to verify that the VXLAN identifier assigned to a virtual network at the implementation level is equal to the segment assigned to this same network at the infrastructure management level (after conversion to decimal), which is expressed by virtual link consistency property (P9). To find whether there exist relations' tuples that falsify this property (¬P9), we first formulate the CSP instance using the negative form of the corresponding predicate, which corresponds to the following predicate:

$$\begin{aligned}\neg\mathtt{P11} = \exists \mathtt{vm1} \in \mathtt{iVM}, \exists \mathtt{vxlan} \in \mathtt{VXLAN}, \exists \mathtt{sw} \in \mathtt{OVS}, \exists \mathtt{vlan} \in \mathtt{VLAN}, \exists \mathtt{p} \in \mathtt{PORT}, \\ \forall \mathtt{vm2} \in \mathtt{vVM}, \forall \mathtt{vNet} \in \mathtt{vNET}, \forall \mathtt{seg} \in \mathtt{Segment}, \forall \mathtt{vp} \in \mathtt{vPORT}: \\ \mathtt{IsConnectedOnPort}(\mathtt{vm1}, \mathtt{sw}, \mathtt{p}) \wedge \mathtt{IsAssignedVLAN}(\mathtt{sw}, \mathtt{p}, \mathtt{vlan}) \wedge \\ \mathtt{IsMappedToVXLANOnOVS}(\mathtt{sw}, \mathtt{vlan}, \mathtt{vxlan}) \wedge \\ \neg\mathtt{IsConnectedTovNet}(\mathtt{vm2}, \mathtt{vNet}, \mathtt{vp}) \vee \neg(\mathtt{vm1} = \mathtt{vm2}) \vee \\ \neg\mathtt{IsAssignedSeg}(\mathtt{vNet}, \mathtt{seg}) \vee \neg(\mathtt{seg} = \mathtt{vxlan})\end{aligned} \tag{4}$$

By verifying predicate 4 over all the aforementioned relations' instances, the solver

finds an assignment such that the above predicate becomes true, which means that the property P9 is violated. The predicate instance that caused the violation can be written as follows:

$$\begin{aligned} &\texttt{IsConnectedOnPort}(\texttt{VM\_21}, \texttt{OVS\_1}, \texttt{Port\_21}) \wedge \\ &\texttt{IsAssignedVLAN}(\texttt{OVS\_1}, \texttt{Port\_21}, \texttt{vlan\_100}) \wedge \\ &\texttt{IsMappedToVXLANOnOVS}(\texttt{OVS\_1}, \texttt{vlan\_100}, \texttt{vxlan\_0} \times 100) \wedge \\ \neg &\texttt{IsConnectedTovNet}(\texttt{VM\_Bapp1}, \texttt{vNet\_B}, \texttt{vPort\_21}) \vee \neg(\texttt{VM\_21} = \texttt{VM\_Bapp1}) \vee \\ \neg &\texttt{IsAssignedSeg}(\texttt{vNet\_B}, \texttt{seg\_512}) \vee \neg(\texttt{seg\_512} = \texttt{vxlan\_0} \times 100) \end{aligned} \tag{5}$$

Since `seg` *is equal to* `512` *and the decimal value of* `VXLAN0_×100`*, namely,* `vxlan`*, is* `256`*, then the equality* `seg=vxlan` *will be evaluated to false and* `¬(seg=vxlan)` *will be evaluated to true, which makes the assignment in predicate 5 satisfying the constraint. This set of tuples provides the evidence about what values breached the security property P9. Note that as VM consistency and port consistency properties were assumed to be verified, the equality between* `VM_Bapp1` *and* `VM_21` *holds (based on their identifiers that could be their MAC addresses for instance).*

In the following section, we present our auditing solution integrated into OpenStack and show details on how we use the CSP solver Sugar as a back-end verification engine.

4.4 Implementation

In this section, we first provide a high-level architecture of our system. We then briefly review the most relevant OpenStack services and OVS. Finally, we detail our implementation and its integration into OpenStack and Congress [38], an open-source framework

implementing policy as a service for OpenStack.

4.4.1 Architecture

Figure 4.6 illustrates a high-level architecture of our auditing system. It has three main components: data collection and processing module, compliance verification module and the dashboard and reporting module. Our solution interacts mainly with the cloud infrastructure management system (e.g., OpenStack) and elements in the data center infrastructure to collect various types of audit data. It also interacts with the cloud tenant to obtain the tenant requirements and to provide the tenant with the audit results. The properties extractor intercepts tenants' requirements (expressed as high level properties) and identifies the corresponding low level and concrete properties that can be directly checked on the collected and processed data. As expressing and processing tenants' policies is out of the scope of this work, we assume that they are parsable XML files.

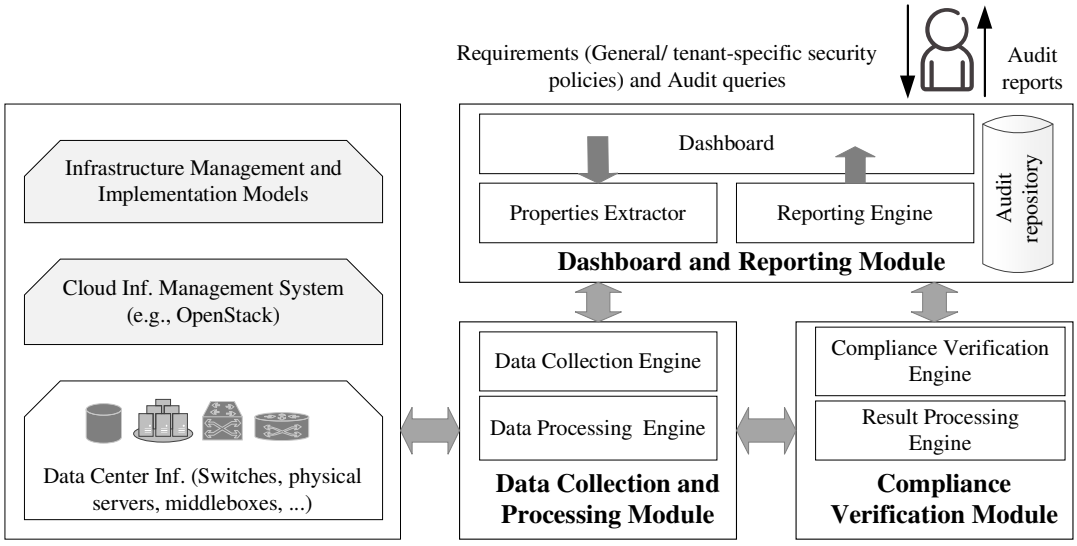


Figure 4.6: A high-level architecture of our cloud auditing solution

The data collection and processing module is composed of the collection engine and the processing engine. The collection engine is responsible for collecting the required audit data in a batch mode. The role of the processing engine is to filter, format, aggregate,

and correlate this data. The required audit data may be distributed throughout the cloud and in different formats. The processing engine pre-processes the data in order to provide specific information needed to verify given properties. Furthermore, the processing engine recovers the formalized form of the concrete properties that need to be audited. The last processing step is to generate the code for compliance verification using both the processed data and the formalized properties. The generated code depends on the selected back-end verification engine.

The compliance verification module is responsible for performing the actual verification of the audited properties and the detection of violations, if any. Triggered by an audit request, the compliance verification module invokes the back-end verification engine. In case of violation, the verification engine provides details on the breach, which are then intercepted and interpreted by the result processing engine.

If a security audit property fails, evidence can be obtained from the output of the verification back-end. Once the outcome of the compliance verification is ready, audit results and evidences are stored in the audit repository database and made accessible to the audit reporting engine. Several potential formal verification engines can serve our needs, and the actual choice may depend on the property being verified.

4.4.2 Background

As we are interested in auditing the infrastructure virtualization and network segregation, we first investigated OpenStack documentation to learn which services are involved in the creation and maintenance of the virtual infrastructure and networking. We found that Nova and Neutron services in OpenStack are responsible for managing networking at the management layer. We also investigated the implementation-level, and found that OVS instances running in different compute nodes are the main components that implement the virtual infrastructure. Following is a brief description of Nova, Neutron and OVS:

Nova [16] This is the OpenStack project designed to provide massively scalable, on demand, self-service access to compute resources. It is considered as the main part of an IaaS model.

Neutron [16] This OpenStack project provides tenants with capabilities to build networking topologies through the exposed API, relying on three object abstractions, namely, networks, subnets and routers. When leveraged with the Modular Layer 2 plug-in (ML2), Neutron enables supporting various layer 2 networking technologies. In many existing deployments, OVS is used with OpenStack to manage the network connectivity between tenants' VMs.

In our settings, an OVS defines two interconnected bridges, the integration bridge (`br-int`) and the tunneling bridge (`br-tun`). VMs are connected via a virtual interface (tap device)[2] to `br-int`. The latter acts as a normal layer 2 learning switch. It connects VMs attached to a given network to ports tagged with the corresponding VLAN, which ensures traffic segregation inside the same compute node.

Each tenant's network is assigned a unique VXLAN identifier over the whole infrastructure. The *br-tun* is endowed with OpenFlow rules [96] that map each internal VLAN-tag to the corresponding VXLAN identifier and vice versa. For egress traffic, the OpenFlow rules strip the VLAN-tag and set the corresponding VXLAN identifier in order to transmit packets over the physical network. Conversely, for ingress traffic, OpenFlow rules strip the VXLAN identifier from the received traffic and set the corresponding VLAN-tag.

4.4.3 Integration Into OpenStack

We mainly focus on four components in our implementation: the data collection engine, the data processing engine, the compliance verification engine and the dashboard and reporting

[2]This direct connection is an abstraction of a chain of one-to-one connections from the virtual interface to the *br-int*. In fact, the tap device is connected to the Linux bridge *qbr*, which is in turn connected to the `br-int`.

engine. In the following, we describe our implementation details.

Data collection engine. The data collection engine involves several components of OpenStack, e.g., Nova and Neutron for collecting audit data from databases and log files, different policy files and configuration files from the OpenStack ecosystem, and configurations from various virtual networking components such as OVS instances in all physical servers to fully capture the configuration and virtual networks state. We present hereafter different sources of data along with the current support for auditing offered by OpenStack and the virtual networking components. Table 4.6 shows some sample data sources. We use different sources including OpenFlow tables extracted from OVS instances in every compute node, and Nova and Neutron databases:

- *OpenStack*. We rely on a collection of OpenStack databases, that can be read using component-specific APIs. For instance, in Nova database, table *Instance* contains information about the project (tenant) and the hosting machine, table *Migration* contains migration events' related information such as the source-compute and the destination-compute. The Neutron database includes various information such as port mappings for different virtualization mechanisms.

- *OVS*. OpenFlow tables and internal OVS databases in different compute nodes constitute another important source of audit data for checking whether there exists any discrepancy between the actual distributed configuration at the implementation layer and the OpenStack view.

For the sake of comprehensiveness in the data collection process, we firstly check fields of a variety of log files available in OpenStack, different configuration files and all Nova and Neutron database tables. We also debug configurations of all OVS instances distributed over the compute nodes using various OVS's utilities. Mainly, we recovered ports' configurations (e.g., ports and their corresponding VLAN tags) from the integration bridges

Relations	Sources of Data
IsRunningOn	Table *Instances* in Nova database
IsAssignedSeg	Table *ml2_network_segments* in Neutron database
IsMappedToSeg	Table *neworkconnections* in Neutron database
IsConnectedTovNet	Table *Instances* in Nova database
HasPort	OVS instances located at various compute nodes, *br_int* configuration
IsAssignedVLAN	OVS instances located at various compute nodes, *br_int* configuration
IsMappedToVXLANOnOVS	OVS instances located at various compute nodes, *br_tun* OpenFlow tables
VMRunningOn	OVS instances located at various compute nodes, *br_int* configuration
SWRunningOn	The infrastructure deployment
IsConnectedOnPort	OVS instances located at various compute nodes, *br_int* configuration
HasMapping	OVS instances located at various compute nodes
IsAssociatedWith	OVS instances located at various compute nodes
IsRelatedTo	OVS instances located at various compute nodes

Table 4.6: Sample data sources in OpenStack and Open vSwitch

using the utility `ovs-vsctl show`, and we extracted VLAN-VXLAN mappings form the tunneling bridges' OpenFlow tables using `ovs-ofctl dump-flows`. The tunneling bridge maintains a chain of OpenFlow tables for handling ingress and egress traffic. In order to recover the appropriate data, we identify the pertinent tables where to collect the VLAN-VXLAN mappings from. Through this process, we identify all possible types of data, their sources and their relevance to the audited properties.

Data processing engine. The data processing engine, which is implemented in Python and Bash scripts, mainly retrieves necessary information from the collected data according to the targeted properties, recovers correlation from various sources, eliminates redundancies, converts it into appropriate formats, and finally generates the source code for Sugar.

- Firstly, based on the properties, our plug-in identifies the involved relations. The relations' instances are either fetched directly from the collected data such as the

support of the relation `BelongsTo`, or recovered after correlation, as in the case of the relation `IsConnectedTovNet`.

- Secondly, our processing plug-in formats each group of data as an n-tuple, i.e., `(resource, tenant)`, `(ovs, port, vlan)`, etc.

- Finally, our plug-in uses the n-tuples to generate the portions of Sugar's source code, and append the code with the variable declarations, relationships and predicates for each security property.

Checking consistent topology isolation in virtualized environments requires considering configurations generated by virtualization technologies at various levels, and checking that mappings are properly maintained over different layers. OpenStack maintains tenants' provisioned resources but does not maintain overlay details of the actual implementation. Conversely, current virtualization technologies do not allow mapping VMs, networks and traffic details to their owners. Therefore, we map virtual topology details at the implementation level to the corresponding tenant's network to check whether isolation is achieved at this level. Here are examples of mappings to provide per-tenant evidences for resources and layer 2 virtual network isolation. Figure 4.7 relates relations of property *P*9 along with some of their data support to their respective data sources.

- At the OpenStack level, tenants' VMs are connected to networks through subnets and virtual ports. Therefore, we correlate data collected from *Insatances* Nova table to recover a direct connection between VMs and their connecting networks at the centralized view through the relation `IsConnectTovNet`. We also keep track of their owners.

- At the virtualization layer, networks are identified only through their VXLAN identifiers. We map each network's segment identifier recovered from OpenStack (Neutron

Database) to the VXLAN identifier collected from OVS instances (`br_tun` OpenFlow tables) to be able to map each established flow to the corresponding networks and tenants. Furthermore, for each physical server, we assign VMs to the ports that they are connected to through the relation `IsConnectOnPort`, and we assign ports to their respective VLAN-tags through the relation `IsAssignedVLAN` from the configurations details recovered from `br-int` configuration in OVS.

- At the OpenStack level, ports are directly mapped to segment identifiers, whereas at the OVS level, ports are mapped to VLAN-tags and mappings between the VLAN-tags and VXLAN identifiers are maintained in OpenFlow tables distributed over multiple OVS instances. To overcome this limit, we devised a script that recovers mappings between VLAN-tags and the VXLAN identifiers from the flow tables in `br-tun` using the `ovs-ofctl` command line tool. Then, it recovers mappings between ports and VLAN-tags from the Open-vSwitch database using the `ovs-vsctl` command line utility.

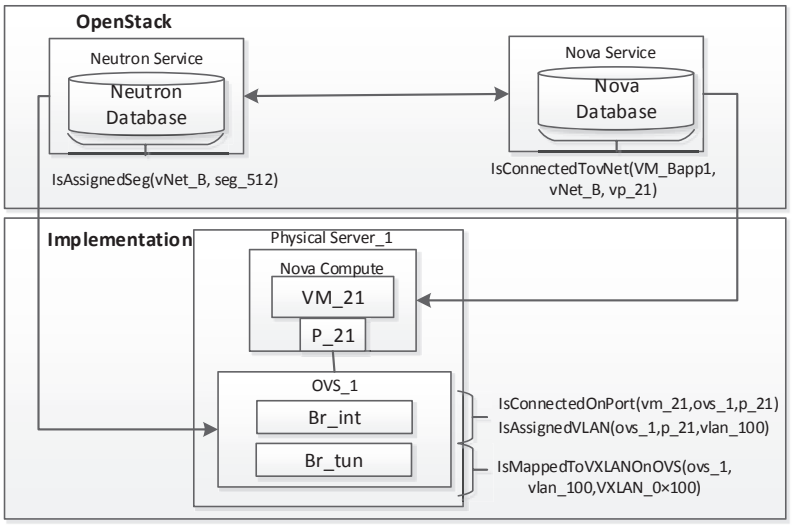


Figure 4.7: Mapping of relations involved in property *P*9 to their data sources

Depending on the properties to be checked, our data processing engine encodes the

involved instances of the virtualized infrastructure model as CSP variables with their domains definitions, where instances are values within the corresponding domain. The CSP code mainly consists of four parts:

- *Variable and domain declaration*. We define different entities and their respective domains. For example, *TENANT* is defined as a finite domain ranging over integer such that *(domain* `TENANT 0 max_tenant` is a declaration of a domain of tenants, where the values are between 0 and `max_tenant`.

- *Relation declaration*. We define relations over variables and provide their supports (instances) from the audit data. Relations between entities and their instances are encoded as relation constraints and their supports, respectively. For example, `HasRunningVM` is encoded as a relation, with a support as follows: `(relation HasRunningVM 2 (supports (vm1,t1) (vm2,t2)))`, where the support of this relation (e.g., $(vm1, t1)$) will be fetched and pre-processed in the data processing step.

- *Constraint declaration*. We define the negation of each property in terms of predicates over the involved relations to obtain a counter-example in case of a violation.

- *Body*. We combine different predicates based on the properties to verify using Boolean operators.

Compliance Verification. The compliance verification engine performs the verification of the properties by feeding the generated code to Sugar. Finally, Sugar provides the results on whether the properties hold or not. It also provides evidence in case of non-compliance.

Example 4.7. *In this example, we discuss how our auditing framework can detect the violation of the virtual links inconsistency caused by the inter-compute node isolation breach described in Example 4.4.*

Firstly, our program collects data from different sources. Then, the processing engine correlates and converts the collected data and represents it as tuples; for an example: `(18045 6100 21) (6100 512)` *reflect the current configuration at the infrastructure management level, and* `(18045 1 21) (1 21 100) (1 100 256)` *correspond to a given network's configuration at the implementation level, where* `VM_Bapp1: 18045, VM_21: 18045, vNet_B: 6100, seg_512: 512, vPort_21: 21, OVS_1: 1, Port_21: 21, VLAN_100: 100, vxlan_1×100: 256`. *Additionally, the processing engine interprets each property and generates the associated Sugar source code (see Listing 4.1 for an excerpt of the code) using processed data and translated properties. Finally, Sugar is used to verify the security properties.*

The predicate P9 for verifying virtual link consistency evaluates to true if there exists a discrepancy between the network `VM_Bapp1` *is connected to according to the infrastructure management view, and the layer 2 virtual network* `VM_Bapp1` *is effectively connected to at the implementation level. In our case, the predicate evaluates to true since* `vxlan0×100≠seg_512` *(as detailed in Example 4.6), meaning that* `VM_Bapp1` *is connected on the wrong layer 2 virtual network.*

Listing 4.1: Sugar Source Code

```
1  //Declaration
2  (domain iVM 0 100000)(domain OVS 0 400)(domain PORT 0 100000)
3  (domain VLAN 0 10000) (domain VXLAN 0 10000)(doamin vVM 0 100000)
4  (domain VNET  0 10000) (domain SEGMENT 0 10000)(domain VPORT 0 100000)
5  (int vm1 iVM) (int vm2 vVM)(int sw OVS) (int p PORT)(int vlan VLAN)
6  (int vxlan VXLAN)(int vnet VNET) (int seg SEGMENT) (int vp VPORT)
7  //Relations declarations and audit data as their support from the
8  infrastructure manangement level
9  (relation IsConnectedTovNet 3  (supports (18045 6100 21)
10 (18037 6150 7895)(18038 6120 2566) ( 18039 6230 554)
11 (18040 6230 4771)))
```

```
(relation IsAssignedSeg 2 (supports (6150 356)(6120 485)(6230 265)
(6100 512)(6285 584)(6284 257)))
//Relations declarations and audit data as their support from the
implementation level
(relation IsConnectedOnPort 3 (supports(((18045 1 21)(18037 96 23)
(18046 65 32)(18040 68 8569)(18047 78954))
(relation IsAssignedVLAN 3 (supports(92 13 41)(92 14 42)(85 38 11)))
(relation IsMAppedToVXLANOnOVS 3 (supports (1 100 256)(92 6018 9)
(92 6019 10)))
//Security properties expressed in terms of predicates over relation
 constraints
(predicate (P  vm1 vm2 vnet seg vxlan  sw p vp)
(and(IsConnectedOnPort vm1 sw p)(IsAssignedVLAN sw p vlan)
(IsMappedToVXLANOnOVS sw vlan vxlan)(IsConnectedTovNet vm2 vnet vp)
(IsAssignedSeg vnet seg)(eq vm1 vm2)(not(eq seg vxlan))))
//The body
(P  vm1 vm2 vnet seg vxlan  sw p vp)
```

Understanding Violations Through Evidences. As explained in Section 4.3.3, we define constraints using the negative form of properties' predicates. Thus, if a solution satisfying the constraint is provided by the CSP solver, then the latter solution is a set of variable values that make the negation of the predicates evaluate to true. Those values indicate the relation instances (system data) that are at the origin of the violation, however, they might be unintelligible to the end users. Therefore, we replace the variables' numerical values by their high-level identifiers, which would help admins identify the root cause of the violation and fix it eventually.

Example 4.8. *From Example 4.7, the CSP solver concludes that the negative form of the property is satisfied, which indicates the existence of a violation. Furthermore, the CSP solver outputs the following variable values as an evidence:* `vm1=18045`, `vm2=18045`, `vnet=6100`, `seg=512`, `vxlan=100`, `sw=1`, `p=21`, `vp=21`. *To make the evidence*

easier to interpret, we replace the value `6100` *of the variable vnet by* `vNet_B`*, the value* `18045` *of the variable* `vm2` *by* `VM_Bapp1` *and the value* `21` *of the variable* `vp` *by* `vp_21`*. Using this information, the admin will conclude that* `VM_Bapp1` *is connected to another* `Tenant_Alpha`*'s layer 2 virtual network at the implementation level identified through* `VXLAN_0×100`.

Dashboard and Reporting Engine. We further implement the web interface (i.e., dashboard) in PHP to place verification requests and display verification reports. In the dashboard, tenant admins are initially allowed to select different standards (e.g., ISO 27017, CCM V3.0.1, NIST 800-53, etc.). Afterwards, security properties under the selected standards can be chosen. Once the verification request is placed, the summarized verification results are shown in the verification report page. The details of any violation with a list of evidences are also provided.

4.4.4 Integration Into OpenStack Congress

To demonstrate the service agnostic nature of our framework, we further integrate our system with the OpenStack Congress service [38]. Congress implements policy as a service in OpenStack in order to provide governance and compliance for dynamic infrastructures. Congress can integrate third party verification tools using a data source driver mechanism [38]. Using Congress policy language that is based on Datalog, we define several tenant specific security policies. Then, we use our processed data to detect those security properties for multiple tenants. The outputs of the data processing engine is provided as input for Congress to be asserted by the policy engine. This allows integrating compliance status for some policies whose verification is not yet supported by Congress.

4.5 Experiments

In this section, we evaluate the scalability of our approach by measuring the response time of the verification task as well as the CPU and memory consumption for different cloud sizes and in different scenarios (a breach violating some properties or no breach).

4.5.1 Experimental Setting

We set up a real environment including 5 tenants, 10 virtual networks each having 2 subnets, 10 routers and 100 VMs. We utilize OpenStack Mitaka with one controller and three compute nodes running Ubuntu 14.04 LTS. The controller is empowered with two Intel Xeon E3-1271 CPU and 4GB of memory. Each compute node benefits from one CPU and 2GB of memory. To further stress the verification engine and assess the scalability of our solution, we generated a simulated environment including up to 6k virtual networks and 60K VMs with the ratio of 10 VMs per virtual network. As a back-end verification tool, we use the CSP solver Sugar V2.2.1 [84]. All the verification experiments are run on an Amazon EC2 C4.Large Ubuntu 16.04 machine (2 vCPU and 3.75GB of memory).

4.5.2 Results

We consider for the experiments three properties from table 4.1, where each is selected from one of the three categories defined therein:

- *Mapping unicity virtual networks-segments* (P1), which is a topology isolation property checked at the infrastructure management level.
- *Mapping unicity VLANs-VXLANs* (P5), which is a topology isolation property checked at the implementation level.

- *Virtual links consistency* (P9), which checks that a VM is connected to the right VXLAN at the implementation level.

In the first set of experiments, we design two configuration scenarios to study different response times in two possible cases: presence of violations and absence of violations. This is because the verification of these two scenarios is expected to have different response times due to the time required to find the evidence of the violation.

In the first scenario, we implement in our environment a configuration of the virtual infrastructure where none of the studied properties are violated. In the second scenario, we implement the topology isolation attack described in Example 4.4. For the latter scenario, as generally, a fast yes or no answer on the compliance status of the system is required by the auditor, we only consider the response time to report evidence for the first breach. Note that we do not report the average response time to find all compliance breaches as this depends on the number of breaches, their percentage to the total input size and their distribution in the audit information. Meanwhile, as the real life scenarios can dramatically vary from one environment to another, we cannot use any average number, percentage or distribution of compliance breaches applying to all possible use cases. Therefore, we present in Figure 4.8, the verification time for no security breach detected (left side chart) and the verification time to report non-compliance and provide evidence for the first security breach (right side chart) for different datasets varying from 5K up to 60K VMs. Note that, we implement the attack scenario of topology isolation described in Example 4.4 by randomly modifying some VLAN ports and VLAN to VXLAN mappings.

As indicated in the left chart of Figure 4.8, the time required for verifying P1 and P5, where there is no breach, is 0.6s and 4.5s, respectively, for the largest dataset of 60K VMs. The verification time for those properties increases linearly and smoothly when the size of the cloud infrastructure increases and there is no breach. However, the verification time for property P9 is 102s for 30k VMs and 581s for 60k VMs. The difference in response

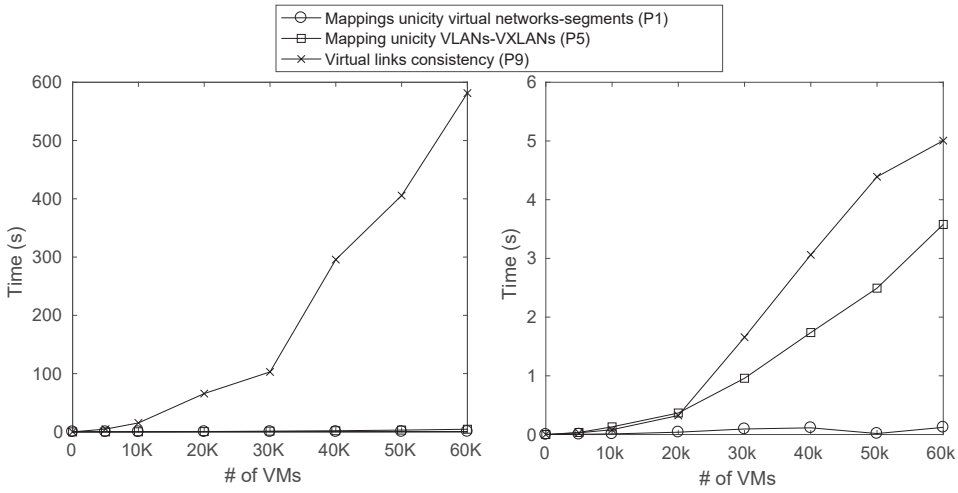


Figure 4.8: Verification time as a function of the number of VMs for properties P1, P5, and P9: (left side) time to report no breach of compliance, and (right side) time to find the first breach and build evidence of non-compliance

time for P9 is justified as the latter is more complex than other properties and involves more relations and thus larger input data. Later in this section, we will show how one can decrease the response time for the verification of P9 to get more acceptable boundaries.

According to Figure 4.8 (right side chart), the time required to find the first breach and build the supporting evidence for each one of the three properties remains under 5s for the largest dataset, which is two orders of magnitude smaller than the time required to assert compliance for the entire system. The time required to find the first breach, depends on several factors such as the predicates affected by the breach and the location of the breach in the input file. However, the latter response time is always shorter than the time required for asserting the compliance of the system.

The left side chart of Figure 4.9 reports CPU consumption percentage as a function of the datasets' size, up to 60k VMs. For the largest dataset, the peak CPU usage reaches 50% for P9 and does not exceed 25% for P1. Also, the highest memory usage observed does not exceed 8% for P9 verification (see the right chart of Figure 4.9), and 3.3% for the largest dataset for P5. It is worthy to note that these amounts of CPU/memory usage are not monopolized during the whole verification time and they represent the peek usage. We

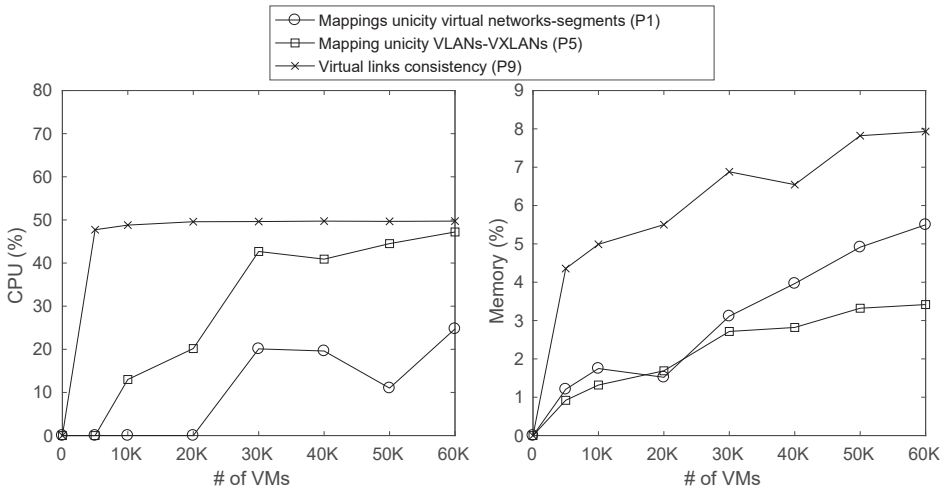


Figure 4.9: CPU (left side) and memory (right side) usage to verify no-compliance breach for properties P1, P5 and P9

therefore remark the low cost on CPU and memory for our approach.

In our second set of experiments, since Sugar supports several SAT solvers, we run Sugar with different SAT solvers to investigate which option provides a better response time, particularly for property P9. According to Figure 4.10, Treengling solver provides the longest response time with 900s for a 30k VMs dataset, whereas Minisat provides the best response time with 102s. All previously reported verification results in the other experiments were obtained using Minisat.

In our third set of experiments, we investigate the parameters that affect the response time, particularly in the case of complex security properties such as P9. To this end, we consistently split the data supports for the relations `IsConnectedToVnet` and `IsAssignedSeg` of P9 over multiple CSP files (up to 16 files), and repeated the supports for the relations `IsConnectedOnPort`, `IsAssignedVLAN` and `IsMappedToVXLANOnOVS` to maintain data interdependency.

Figure 4.11 reports the response times for the parallel verification of different CSP sub-instances of P9 using multiple processing nodes for the largest dataset (60K VMs). By splitting the data support into two CSP files, the verification time already decreases from

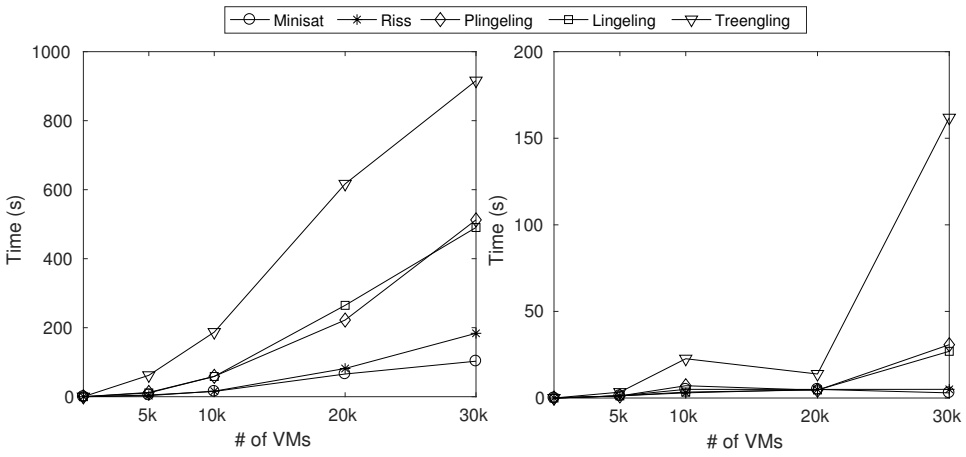


Figure 4.10: Verification time using different SAT solvers for P9 as a function of the number of VMs: (left side) time to report no breach of compliance, and (right side) time to find the first breach and build evidence of non-compliance

581s to 168s (i.e., a factor of improvement of 71%), whereas it decreases up to 4.6s when splitting the data over 16 CSP sub-instance files.

Based on this last experiment, we can conclude that splitting the input data for the same property to be verified using parallel instances of CSP solvers can improve the response time. However, this should be performed while considering the dependency between different relations and their supports in the predicate to be solved.

Based on those results, we conclude that our solution provides acceptable response time for auditing security isolation in the cloud, particularly, in the case of off-line auditing. While the verification of simple properties is scalable for large cloud virtual infrastructures, response time for complex properties involving large input data can induce more delays that can be still acceptable for auditing after the fact. However, response time for those properties can be considerably improved by splitting their CSP instance into sub-instances involving smaller amounts of data to be checked in parallel. Note that our analysis holds for the specific scenario where security properties are expressed as constraints defined as logical operations over relations, which is only a subset of possible constraints that can be offered by the CSP solver Sugar (the complete set of constraints supported by Sugar can

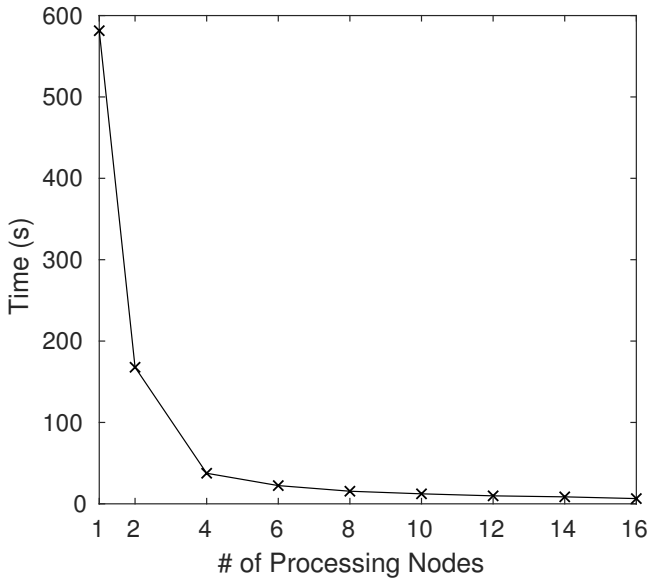


Figure 4.11: Verification time as function of the number of processing nodes for P9 for a dataset of 60k VMs, where each processing node verifies a separate CSP sub-instance of P9

be found in [97]). Expressing new security properties with other kinds of constraints may require performance to be reassessed through new experiments.

4.6 Discussion

The experimental results presented in the previous section show that CSP solvers can be used for off-line auditing verification with acceptable response time and scalability in case of moderate size of data. Our results also show that for properties handling larger datasets, we need to decompose the verification of the properties over smaller chunks of data to improve the response time. Additionally, we explore a parallel processing approach to improve the response time for very large datasets. Note that the response time can be further improved to achieve on-line auditing by improving the performance of the CSP-solving phase [98], which is an interesting future direction.

Model Entities	OpenStack	AWS-EC2-VPC	GCP	Microsoft Azure	VMware vCD
VM	Instance	EC2 instance	VM instance	Azure VM	VM
vNet	Network	Virtual private cloud	Auto mode vpc Custom mode vpc	Virtual Network	Network
vSubnet	Subnet	Subnet	Subnet	Subnet	Subnet
vRouter	Router	Routing tables	Routes	BGP and user-defined routes	Distributed logical routers
vPort	Port	-	-	NIC	Port/port-group
Segment	Network ID	VPC ID	VPC ID	Virtual network ID	Network ID

Table 4.7: Mapping virtual infrastructure model entities into different cloud platforms

The abstract views offered by different cloud platforms to tenants are quite similar to what we propose at the cloud infrastructure management view of our model. For instance, both Amazon AWS EC2-VPC (Virtual Private Cloud) [11], Google Cloud Platform (GCP) [12], Microsoft Azure [13] and VMware virtual Cloud Director (vCD) [94] provide tenants with the capability to create virtual network components as software abstractions, enabling to provision virtual networks. Therefore, our model can capture the main virtual components that are common to most of the IaaS management systems with minor changes. Table 4.7 maps the entities of our infrastructure management view model to their counterparts in the cloud platforms cited above.

Eucalyptus [99] is an open source IaaS management system. The Eucalyptus virtual private cloud (VPC) is implemented with MidoNet [100], an open-source network virtualization platform. In the same fashion as OpenStack Neutron, Eucalyptus MidoNet supports virtualization mechanisms such as VLAN and VXLAN to implement large scale layer 2 virtual networks spanning over the cloud infrastructure. Therefore, our implementation layer model can be applied to Eucalyptus implementations with minor changes.

However, implementation details may significantly vary between different platforms. Furthermore, cloud providers typically do not disclose their implementation details to their customers. Therefore, the implementation layer of our model along with the extracted properties might need to be revised according to the implementation details of each cloud deployment if those are provided. However, this needs to be done only once before initializing the compliance auditing process.

Our current solution is designed for the specific OpenStack virtual layer 2 implementation mainly relying on VLAN and VXLAN as well-established network virtualization technologies, and OVS as a widely used virtual switch implementation. However, as we use high-level abstractions to represent virtual layer 2 connectivity and tunneling technologies, we believe that our approach remains applicable in case of other overlay technologies

such as GRE. In small to medium clouds, where VLAN tags are sufficient to implement all layer 2 virtual networks on top of the physical network, our implementation model is simplified and the security properties related to the mapping between VLAN and VXLAN can be skipped.

Among the main advantages of using a CSP solver for the verification is that it allows to integrate new audit properties with a minor effort. In our case, including a new property consists of expressing it in FOL and identifying the audit data it should be checked against. These properties can be modified at any stage of the cloud life cycle and their verification or not can be decided depending on the cloud deployment offering (e.g., public or private cloud).

In this work, we extracted a set of security properties from specific domains in relevant cloud security standards that are mainly related to infrastructure virtualization and tenants' networks isolation (e.g., Infrastructure Virtualization Systems domain from CCM, and Segregation in Networks section from ISO27017). Thus, our list of implemented security properties is not meant to exhaustively cover the entire security standards. Covering other security control classes for the standards requires extracting new sets of security properties to be modeled and formalized. However, as we handle general concepts for modeling different virtual resources, we believe that our approach can be generalized to other security properties to support the entire security standards.

Finally, through this work, we show the applicability and the benefit of our formal approach in verifying security properties while providing evidences to assist admins finding the root causes of violations. As discussed in this section, we believe our high-level abstractions-based model can be easily mapped to different cloud platforms. However, the model needs to be adapted to support those different cloud platforms' implementation details, and augmented to support new security properties.

4.7 Summary

Auditing compliance of the cloud with respect to security standards faces several challenges. In this work, we proposed an automated off-line auditing approach while focusing on verifying network isolation between tenants' virtual networks in OpenStack-managed cloud at layer 2 and overlay. As shown in this work, the layered nature of the cloud stack and the dependencies between layers make existing approaches that separately verify each single layer ineffective. To this end, we devised a model that captures for each cloud-stack layer, namely the infrastructure management and the implementation layers, the virtual network entities along with their inter-dependencies and their isolation mechanisms. The model helped in identifying the relevant data for auditing network isolation and capturing its underlying semantics across multiple layers. Furthermore, we devised a set of concrete security properties related to consistent network isolation on virtual layer 2 and overlay networks to fill the gap between the standards and the low level data. To provide a reliable and evidence-based auditing, we encoded properties and data as a set of constraints satisfaction problems and used an off-the-shelf CSP solver to identify compliance breaches. Our approach furthermore pinpoints the roots of breaches enabling remediation. Additionally, we reported real-life experience and challenges faced when trying to integrate auditing and compliance verification into OpenStack. We further conducted experiments to demonstrate the applicability of our approach. Our evaluation results show that formal methods can be successfully applied for large data centers with a reasonable overhead.

Chapter 5

QuantiC: Distance Metrics for Evaluating Multi-Tenancy Threats in Public Cloud

5.1 Introduction

Multi-tenancy of the cloud is a double edged sword. On one side, the economic gain fulfilled through resource sharing constitutes one of the most appealing cloud advantages that attract prospective customers. On the other side, the security challenges driven by multi-tenancy and the associated risks [101] constitute some of the main concerns that are holding back the migration of critical applications to cloud.

In fact, the proximity with the victim can be exploited by malicious cloud users to mount several attacks. In Table 5.1, we roughly classify those attacks into two categories according to the required proximity (the list of attacks is not meant to be exhaustive; other, including future or unknown, attacks may also fit into those categories). When an attacker shares the same host with the targeted victim, (s)he can launch type I attacks (e.g., side

channel attacks[15]), whereas type II attacks (e.g., power attack [102]) can be mounted when resources are shared with the victim at higher levels of the cloud infrastructure, (e.g., rack-level). Successful attacks may affect security properties of both victim's virtual machines (VMs) and their generated network flows at various levels of the hierarchy. As an example, recent works have demonstrated the feasibility of real-life attacks conducted in commercial clouds including Amazon EC2, aiming at forcing malicious VMs to be placed within a specific zone, which could be a host, a rack or a larger scale area inside the cloud data center [28, 29].

Today's cloud service providers (CSPs) are well aware of such multi-tenancy-related threats, and they are often under obligation to protect their tenants against such threats, either as part of the service level agreements or to demonstrate compliance with security standards (e.g., CCM 3.0.1 [7]). Nonetheless, addressing multi-tenancy threats remains a challenging issue. First of all, completely avoiding multi-tenancy is certainly impractical since it reduces the financial benefit, which is an important factor to cloud adoption. Alternatively, enabling resource sharing naturally implies a degree of exposure to multi-tenancy threats. A mid-way solution for the CSP would be to balance between the security implications and the economic benefits of resource sharing. In this respect, evaluating multi-tenancy threats based on the proximity between tenants sharing the same cloud constitutes a valuable means towards reaching an optimum trade-off between tolerated risks and costs according to negotiated contracts.

Particularly, existing approaches (e.g., [63, 103]) propose metrics to evaluate the overall cloud security risk based on vulnerabilities in cloud deployments (a detailed review of the related work is given in Section 2.3.2). Nonetheless, none of them provides the potential impact at tenant-level according to the degree of resource sharing. Furthermore, those works focus only on the multi-tenancy threat related to type I attacks, while evaluating the threat of type II multi-tenancy attacks has not been tackled yet.

Multi-Tenancy Attacks		Cloud Inf. Levels		Targeted Resources		Targeted Sec. Prop.		
		Host only	Different Levels	Compute	Network	C	I	A
Type I	Side channel attacks [15]	•		•		•		
	Host-based DoS attack [104]	•		•	•			•
	SDN-based freeloading attack [105]	•			•	•	•	
Type II	Power attacks [102]		•	•	•			•
	Bandwidth attack [106]		•		•			•
	Resource abuse [107]		•		•			•

Table 5.1: Multi-tenancy attacks, their scopes, targeted resources and the affected security properties, namely, confidentiality (C), integrity (I) and availability (A)

To the best of our knowledge, this is the first work that proposes multi-level metrics to quantify the distance between tenants' virtual infrastructures in an SDN-based cloud, as a means to evaluate the multi-tenancy threats related to both type I and type II attacks and assess the corresponding risk per tenant. Specifically, the main contributions of this work are as follows.

- We devise a multi-level model capturing tenants' virtual infrastructures deployment inside SDN-based cloud.
- We propose novel metrics, namely, physical, compute and network distances, to quantify the multi-tenancy threat in an SDN-based cloud.
- We present three case studies based on both a real cloud and fictitious clouds. The first and second case studies show how our metrics correlate with the two types of multi-tenancy attacks. In the third case study, we implement our metrics in OpenStack and show how they can be used to define the CSP's compliance with tenants' distance requirements.

5.2 Models

In the following, we discuss our threat model, and present the running example and the cloud infrastructure model.

5.2.1 Threat Model

In this study, we assume that tenants do not have any prior knowledge on the identities of other tenants hosted inside the same cloud. Our in-scope attacks include any multi-tenancy attacks that require an adversary to share resources with the victim tenant at multiple levels of the cloud data center. Any attacks involving administrator privileges are out of scope.

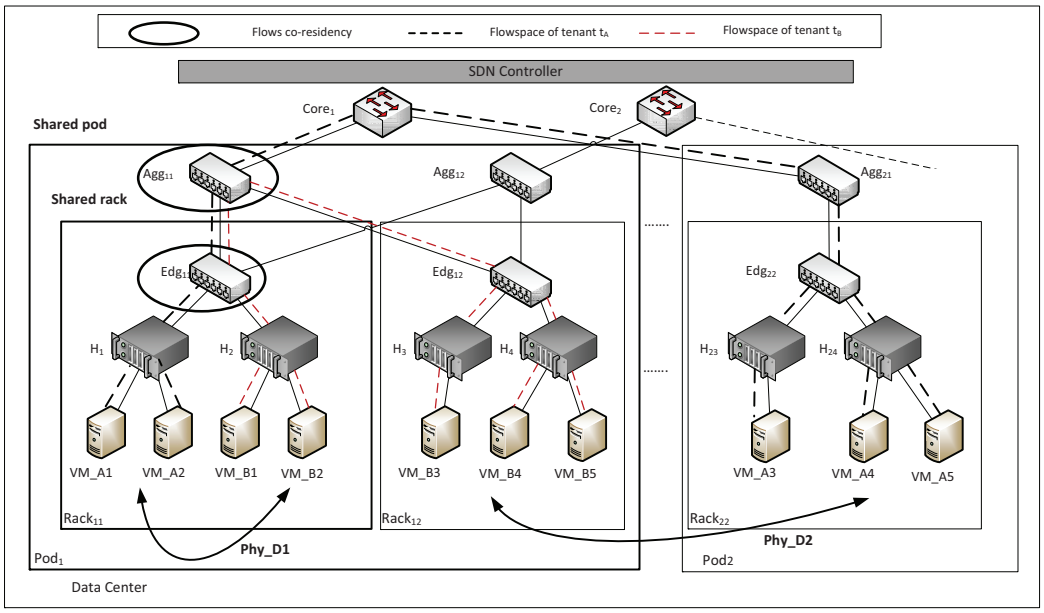


Figure 5.1: An example demonstrating the physical distance between tenants' virtual infrastructures, where *VM_A1*,.., *VM_A5* belong to tenant t_A and *VM_B1*,.., *VM_B5* belong to tenant t_B

Consequently, we assume the information collected from the cloud infrastructure management system to calculate our metrics are trusted.

Our metrics are meant for evaluating the multi-tenancy threats against the in-scope attacks, and they are not designed to detect such attacks, identify the malicious tenant, or pinpoint the vulnerabilities. In fact, our metrics can be applied without any prior knowledge of the attacker's identity (unlike [61]). Thus, our metrics are complementary to other attack-specific security solutions, e.g., attack detection and vulnerability analysis.

5.2.2 Running Example

In Figure 5.1, tenant t_A shares the same data center with many other tenants (to better illustrate the case, we consider an exemplary tenant t_B). Assume the CSP wants to evaluate the impact of potential type I and type II multi-tenancy attacks depicted in Table 5.1 against t_A. Based on the deployment in Figure 5.1, the CSP can make the following observations:

- None of t_A's VMs are co-located with t_B at the host-level, therefore, it is unlikely for t_B to perform type I attacks (e.g., side channel attacks [15]) against t_A's VMs, or to abuse their network flows (e.g., freeloading attack [105]).

- Although launching type I attacks is out of t_B's reach, a closer look reveals that t_A is still under the risk of type II attacks that take advantage of the shared infrastructure at higher levels without requiring host-level co-residency. For example, t_B can perform power attack [102] at $Rack_{11}$ using *VM_B1* and *VM_B2* to disturb services running at *VM_A1* and *VM_A2* located at the same rack. This attack also disturbs the communication of *VM_A1* and *VM_A2* with *VM_A3*, *VM_A4* and *VM_A5* located at $Rack_{22}$.

- Furthermore, *VM_A3*, *VM_A4* and *VM_A5*, that are located in a different rack and pod than t_B, are less exposed to type II attacks since their physical distance with respect to t_B is larger than the physical distance of *VM_A1* and *VM_A2* with respect to the same tenant ($Phy\_D2 > Phy\_D1$).

The above observations intuitively show the correlation between measuring distances between tenants' virtual infrastructures and evaluating the degree of exposure to multi-tenancy threats at different levels of the shared cloud infrastructure.

5.2.3 Multi-Level Cloud Infrastructure Model

To measure the distance between tenants, we derive an entity-relationship model that captures tenants' virtual infrastructure elements, the cloud infrastructure elements and their relationships. Figure 5.2 illustrates such a model. The cloud physical infrastructure includes servers and switches that are hierarchically structured in different management zones shown as aggregated nodes (e.g., several hosts can be aggregated into a rack zone).

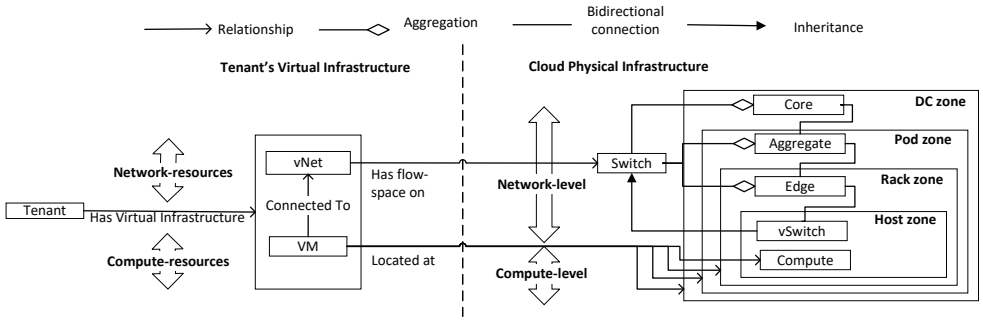


Figure 5.2: Multi-level cloud infrastructure model capturing tenants' virtual infrastructures, the physical infrastructure and their mapping. Note that the presented three-tiered network hierarchy is shared by most cloud data center topologies [108]

A *Tenant*'s virtual infrastructure consists of a set of *VMs* and their connecting virtual networks (*vNet*). Tenants' VMs are located at *compute* services running inside hosts. VMs are connected to vNets that are typically implemented using *flowspaces* constituted of a set of OpenFlow rules [96] segregated with flow tags[1]. These rules are configured in some physical and virtual switches in different levels of the hierarchy to enable the communication between VMs. We use FS_{vNet} to denote the cloud-wide flowspace of *vNet*, FS^{i}_{vNet} to denote the flowspace of *vNet* at *Level_i*, and $FS^{sw_{ij}}_{vNet}$ to denote a flowspace in a given switch sw_{ij} at *Level_i*.

On the right side of Figure 5.2, we define four physical *levels* (*Level_0* to *Level_3*) where tenants' virtual infrastructures (depicted on the left side of Figure 5.2) might be located. As detailed later in Section 5.3, we use those levels to define our distance metrics. In the following, we provide the formal definition for the multi-level cloud infrastructure model.

Definition 1 (Multi-Level Cloud Infrastructure Model)**.** *We define the cloud infrastructure*

[1]A flow tag is a special match field in OpenFlow rules that enables to segregate flow rules belonging to different virtual networks

model as an array $\overrightarrow{CInf}$ *of dimension four, where* $CInf[i].zone$ *and* $CInf[i].switch$ *are respectively the sets of zones and switches at Level_i* ($0 \leq i \leq 3$).

Example 5.1. *Figure 5.3 illustrates an instance of the aforementioned multi-level cloud infrastructure model (Figure 5.2) capturing the example of Figure 5.1. In Figure 5.3, an*

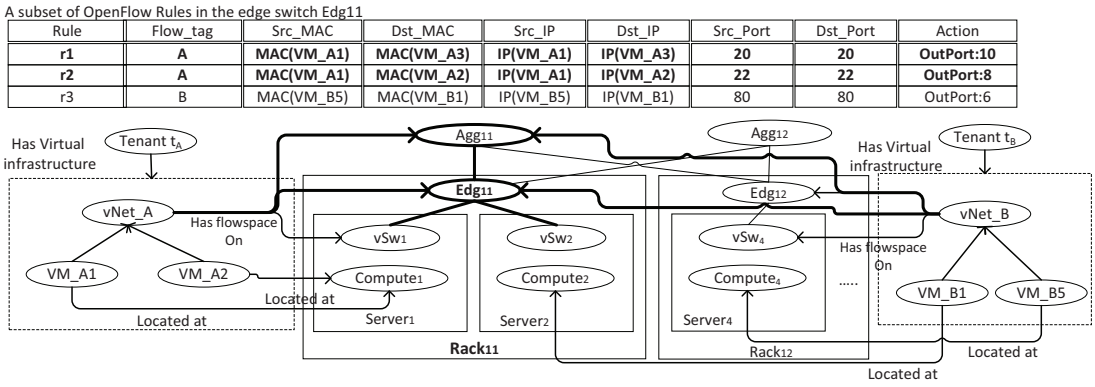


Figure 5.3: An instance of the multi-level cloud infrastructure model capturing a subset of the deployment of Figure 5.1

excerpt of the OpenFlow table in Edg_{11} *shows the co-residency of the flow rules belonging to vNet_A (i.e., r1 and r2) and vNet_B (i.e., r3). Specifically, VM_A1 and VM_A2 of* t_A *located at* $Rack_{11}$ *communicate with VM_A3, VM_A4 and VM_A5 (not shown for space limitation) located at* $Rack_{22}$ *through vNet_A. Similarly, VM_B1 located at* $Rack_{11}$ *communicates with VM_B5 located at* $Rack_{12}$ *through vNet_B. Those communications are made possible through flowspaces installed inside* Edg_{11}, Agg_{11} *and other switches in the topology depending on the location of the communicating VMs. Since VM_A1 and VM_A2 of* t_A *co-reside with VM_B1 at* $Rack_{11}$, *the flowspaces governing their flows will inevitably share* Edg_{11} *at the rack-level and possibly* Agg_{11} *at the pod-level.*

5.3 Multi-Tenancy Distance Metrics

We first define the multi-level physical distance between a pair of tenants to capture their symmetric distance based on the level of physical resource sharing, then we refine this distance along the compute and network dimensions to quantify their asymmetric distances based on their virtual infrastructures' deployment.

5.3.1 Physical Distance

The physical distance captures the symmetric relationship between a pair of tenants in terms of the levels of shared resources. We define this distance between two tenants' virtual infrastructures (VMs and their flowspaces) as a four-dimensional vector D_ϕ, where $D^i_\phi = 0$ (resp. $D^i_\phi = 1$) means *Level_i* is (not) shared. We provide an illustrative example followed by the formal definition.

Example 5.2. *In Figure 5.3, VMs of tenant t_A do not co-locate in the same hosts at Level_0 with the VMs of tenant t_B, their physical distance at Level_0 is therefore $D^0_\phi = 1$. However, VM_A1 and VM_A2 share $Rack_{11}$ at Level_1 with VM_B1 and VM_B2, and since management zones are nested, it follows that all the upper levels of the cloud infrastructure are also shared. Additionally, the flowspaces associated with vNet_A and vNet_B share Edg_{11} at Level_1 and Agg_{11} at Level_2. Thus, the physical distance between the two tenants can be quantified using the vector* $(1,0,0,0)$.

Let t and t' be two tenants hosted at the cloud data center. The virtual infrastructure belonging to tenant t (resp. tenant t') is composed of a set of VMs, VM_s (resp. VM'_s) connected to *vNet* (resp. *vNet'*), where FS^i_{vNet} (resp. $FS^i_{vNet'}$) is the associated flowspace at a given *Level_i* ($0 \leq i \leq 3$). We define the set of shared zones between t and t' at *Level_i* to be the set of zones that are simultaneously accommodating at least one VM belonging to tenant t and one VM belonging to tenant t'. We denote it $sz_i\{VM_s, VM'_s\}$. We similarly

define the set of shared switches between t and t' at $Level_i$ to be the set of switches on which is installed at least one OpenFlow rule r from each of t and t' flowspsaces. We denote it $ss_i\left\{FS^i_{vNet}, FS^i_{vNet'}\right\}$. We define the symmetric physical distance between the pair of tenants $\{t,t'\}$ as follows:

Definition 2 (Physical Distance)**.** *Let $sz_i\{VM_s, VM'_s\}$ and $ss_i\left\{FS^i_{vNet}, FS^i_{vNet'}\right\}$ respectively the sets of shared zones and switches between t and t' at Level_i. Then, their physical distance is given by the four dimensional vector $D_\phi\{t,t'\}$, where the values of its elements D^i_ϕ are computed as follows:*

$$D^i_\phi\left\{t,t'\right\} = \begin{cases} 1 & \text{if } sz_i\{VM_s, VM'_s\} = \emptyset \text{ and } ss_i\left\{FS^i_{vNet}, FS^i_{vNet'}\right\} = \emptyset \\ 0 & \text{Otherwise} \end{cases}$$

5.3.2 Compute Distance

The compute distance is an asymmetric distance that captures the degree of exposure of a tenant t's VMs to another tenant t'.

Example 5.3. *From Example 5.2 we have $D_\phi\{t_A, t_B\} = (1,0,0,0)$. VM_A1 and VM_A2 of t_A share $Rack_{11}$ with t_B's VMs, while VM_A3, VM_A4 and VM_A5 share the cloud infrastructure with t_B at Level_3 only, which corresponds to the data center. Consequently, the compute distance for t_A with respect to t_B at Level_1 and Level_2 is the fraction of VMs that do not share the same racks and pods, which is $3/5$. Hence, the multi-level compute distance for tenant t_A with respect to t_B is $(1, 3/5, 3/5, 0)$.*

More formally, we define the average compute distance of tenant t with respect to tenant t' according to the number of shared zones as follows (VM^z_s is the set of VMs located at zone z):

$$D^i_{\varsigma}(VM_s, VM'_s) = \begin{cases} D^i_{\phi}\{t, t'\} & \text{if } sz_i\{VM_s, VM'_s\} = \emptyset \\ \frac{\sum_{z \in CInf[i].zone \setminus sz_i\{VM_s, VM'_s\}} |VM^z_s \cap VM_s|}{|VM_s| \times |sz_i\{VM_s, VM'_s\}|} & Otherwise \end{cases}$$

We consider the average distance because the more the shared zones the higher the risk related to multi-tenancy attacks would be, as will be discussed in Section 5.4.1. Note that when all tenants' VMs are deployed inside the same data center, D^3_{ς} is always equal to zero. flowspaces

5.3.3 Network Distance

By analogy to the compute distance, the network distance is also an asymmetric distance that captures the degree of exposure of a specific tenant's network resources with respect to another tenant.

Example 5.4. *The OpenFlow rules depicted in Figure 5.3 have six match fields, source/destination MAC, source/destination IP and source/destination port, in addition to the flow-tag. The bit sequence composing those match fields can be either a wildcard or an exact-match, i.e., fixed to zero or to one, where rules with more wildcarded bits define larger flows. Since sharing more flows with other tenants increases the risk of network isolation breaches (e.g., freeloading attacks [105]) and unavailability (e.g., bandwidth attack [106]), we quantify the network distance of vNet_A with respect to vNet_B based on the size of flowspaces that are not sharing the same switches. As illustrated in Figure 5.3, a case of co-residency for the flowspaces of vNet_A and vNet_B is reported at Level_1 in Edg_{11}. In the latter switch, both flow rules r1 and r2 have all the match fields as exact match, meaning that each rule handles a flow composed of one packet only. Since not all flowspaces can be shown for space limitation, we assume that the flow size of vNet_A at Edg_{11} is equal*

to 10, and that its total flow size at Level_1 is 16. Then, the network distance at this level is $D^1_\eta = (16-10)/16$. *Additionally, if we assume that all vNet_A flowspaces are shared with vNet_B at both Level_2 and Level_3, then the network distance vector for vNet_A with respect to vNet_B would be equal to* $(1, 6/16, 0, 0)$.

Let ω be the length in terms of bits of an OpenFlow rule match sequence. Similarly to [32], we abstract away from the meaning associated with each OpenFlow rule's header match field, and consider a match sequence to be a sequence of bits defined over $\{0,1,*\}^\omega$, where * is the wildcard symbol. Let ψ be the number of exact match bits of an OpenFlow rule *r*, where $\psi \leq \omega$, and let *sizeof(_)* be a function that measures the flow size of the OpenFlow rules. The flow size of *r* is equal to $sizeof(r) = 2^{\omega-\psi}$. Particularly, the flow size defined by a rule where all bits in the match sequence are exact match, is equal to $sizeof(r) = 2^0 = 1$ (as $\psi = \omega$). The size of all flowspaces for a given virtual network at a specific level can be computed by aggregating the size of all OpenFlow rules associated with it (for simplicity, we assume that OpenFlow rules do not overlap). This is given by $size(FS^i_{vNet}) = \sum_{r \in FS^i_{vNet}} sizeof(r)$. We define the average network distance between the flowspaces of *vNet* and *vNet'* at a given *Level_i* as:

$$D^i_\eta(FS^i_{vNet}, FS^i_{vNet'}) = \begin{cases} 1 & \text{if } ss_i\left\{FS^{\,i}_{vNet}, FS^{\,i}_{vNet'}\right\} = \emptyset \\ \dfrac{size(\bigcup_{s \in CInf[i].switch \setminus ss_i\left\{FS^{\,i}_{vNet}, FS^{\,i}_{vNet'}\right\}} FS^s_{vNet})}{size(FS^i_{vNet}(t)) \times |ss_i\left\{FS^{\,i}_{vNet}, FS^{\,i}_{vNet'}\right\}|} & otherwise \end{cases}$$

5.4 Case Studies

In this section, we illustrate through case studies the applicability of our distances with both fictitious and real clouds. We also present a quantitative auditing approach based on

our metrics.

5.4.1 Case Study 1 (Correlation with Multi-Tenancy Attacks)

We consider the fictitious cloud data center illustrated in Figure 5.4, which is constituted of four pods, eight racks (two racks per pod) and 96 physical servers (12 servers per rack). This data center is shared by several tenants. For illustrative purposes, we consider four tenants, namely, t_A, t_B, t_C and t_D.

In the following, we show how our physical distance correlates with type I and type II multi-tenancy attacks (see Table 5.1). The rows of matrix $D_\phi(t_A)$ hereafter report the physical distance of t_A with respect to tenants, t_B (first row), t_C (second row) and t_D (third row) based on the deployment of Figure 5.4, where each column represents a physical level of the cloud infrastructure.

$$D_\phi(t_A) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

The following shows how larger physical distances reduce the multi-tenancy threats. Assume t_B, t_C and t_D are malicious and want to take advantage of the multi-tenancy situation to launch type I or type II attacks (see Table 5.1) against t_A. Based on Table 5.1, we can discuss the required distance and potential impact for each category of attacks as follows.

- Type I attacks require co-residency with the targeted victim at the *same host* (e.g., side channel attacks [15]). As $D^0_\phi\{t_A, t_B\} = 0$, the only potential risk of this type of attacks is limited to tenant t_B.

- Type II attacks do not necessarily require co-residency at the host-level to succeed. However, the following reasoning shows that the larger the physical distance is, the

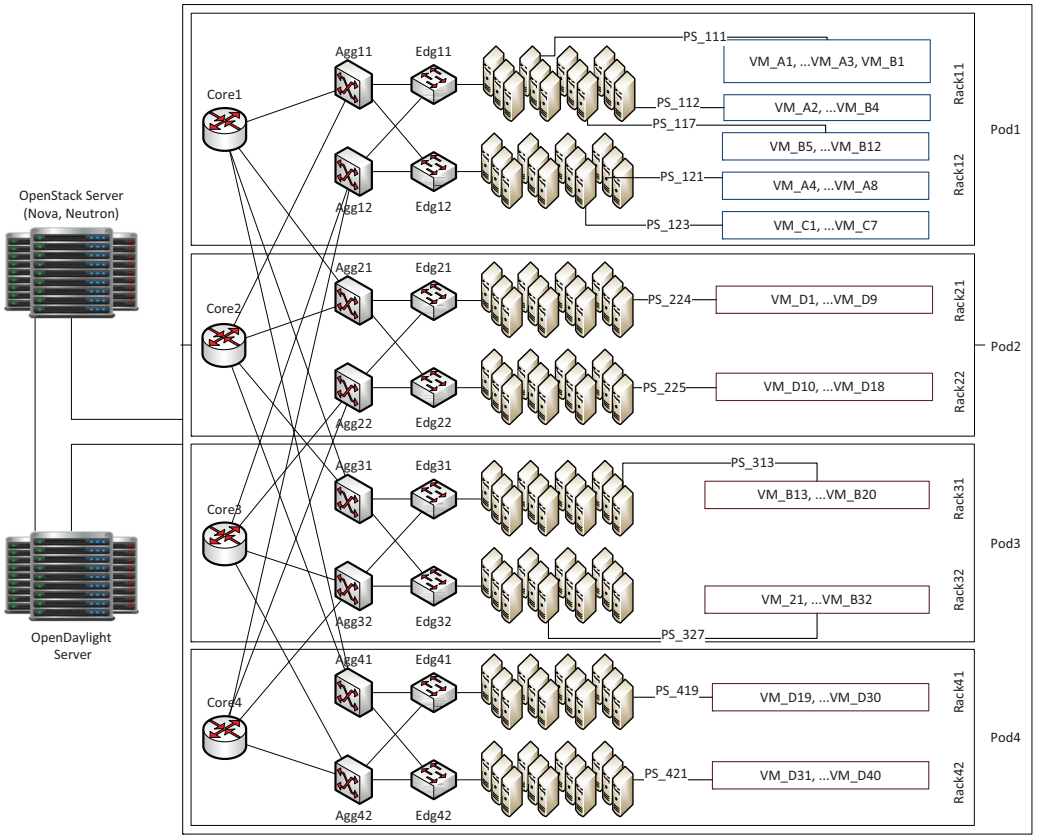


Figure 5.4: An illustrative case study of a cloud data center topology. Physical servers are named *PS_xyz*, where *x* is the index of the pod, *y* is the index of the rack, and *z* is the index of the physical server

less the risk related to those attacks would be. We consider power attack [102] as an example and similar reasoning can be applied to other type II attacks (e.g., bandwidth attacks [106]).

The power attack exploits the power over-subscription vulnerability, which consists of overloading a power supply with more workloads than it supports with the assumption that workloads will never reach their peak simultaneously. If the attacker succeeds to place many VMs inside a zone (server, rack or a larger zone) alimented with the same power facility, then he can generate simultaneous power spikes, which would lead to power outage when the power consumption exceeds the power capacity for that specific zone. However, the larger the zone attacker is targeting, the more controlled VMs need to be deployed to increase the power consumption, since smaller zones converge faster to their peak power[2]. Based on that and considering $D_{\phi}(t_A)$, we can infer the following:

- If t_B or t_C launch their attack against $Rack_{11}$, this would be enough for them to cause damage to all the resources of t_A (VMs and their flows) that are located at this rack zone, since both tenants share the same rack as the victim.
- However, it is more difficult for t_D to affect t_A resources since this would require him to launch this attack at the data center scale (as no racks or pods are shared), which would require much more effort than for t_B or t_C.

To show the correlation between the physical distance and the effort required to launch power attack, we simulated the cloud architecture described in [110], with a number of tenants' workloads following an exponential distribution [107]. Power is defined per units, where each unit power supports one VM. We assume each host has the capability to accommodate eight VMs, and the power consumption at higher levels is obtained by summing up

[2]It has been reported in [109] that racks reach 96% of their peak power, while pods and data centers do not exceed respectively 86% and 72% of their peak power

the power consumption of aggregated lower levels. Figure 5.5 reports the effort required by an attacker at each level of the cloud infrastructure in terms of the number of deployed VMs and their consumed power.

We observe that launching power attack at *Level_0* requires the lowest effort, while launching the attack at the data center scale requires consuming four orders of magnitude more energy, which is achieved by deploying more VMs. From this analysis, we can conclude that larger physical distances reduce the multi-tenancy risk for power attack. In the next case study, we show with real cloud data, the need for refined distance metrics to capture the impact of potential multi-tenancy attacks.

5.4.2 Case Study 2 (Real Cloud Data Center)

This case study is based on a real community cloud hosted at a major telecommunication company. We collect data from part of this cloud composed of 22 hosts organized into two racks as depicted in Figure 5.6. We perform our study on a dataset composed of 372 VMs belonging to 37 tenants. The focus of this case study is to show the complex co-residency relationships between tenants in real world cloud, and therefore, the need for metrics to measure distances between tenants' resources. For illustration, we randomly choose three tenants, t_1, t_2 and t_3. Note that the dimension of our distances is equal to three for this hierarchy, since the latter is only composed of hosts, access and aggregate layers.

Table 5.2 reports the number of VMs of tenants t_1, t_2 and t_3 inside each physical host of the considered part of the cloud data center. One can notice that tenants' VMs are scattered over multiple physical nodes in both racks. Specifically, t_1 has VMs co-residing with both t_2 and t_3's VMs in many different locations. Consequently, the flowspace of t_1's virtual network co-resides with the flowspaces of t_2 and t_3 virtual networks at different physical switches, in addition to the virtual switches running at the physical servers. Due to lack of space, we only discuss the compute distance. The matrix $D_\varsigma(t_1)$ reports the compute

distance of t_1 with respect to t_2 (first row) and t_3 (second row) based on the deployment in Table 5.2.

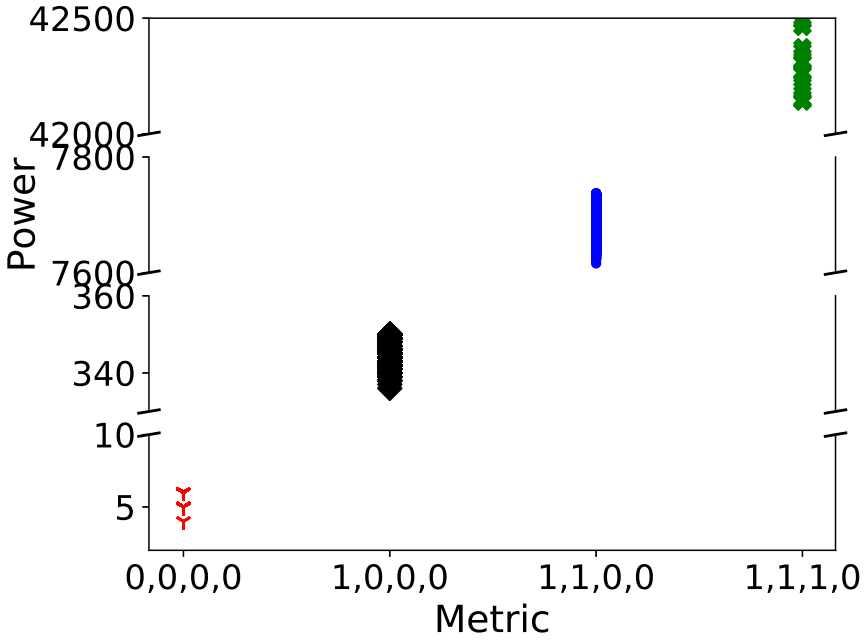
$$D_{\varsigma}(t_1) = \begin{pmatrix} 0.005 & 0.5 & 0 \\ 0.049 & 0.5 & 0 \end{pmatrix}$$

We can infer the following from the compute distances:

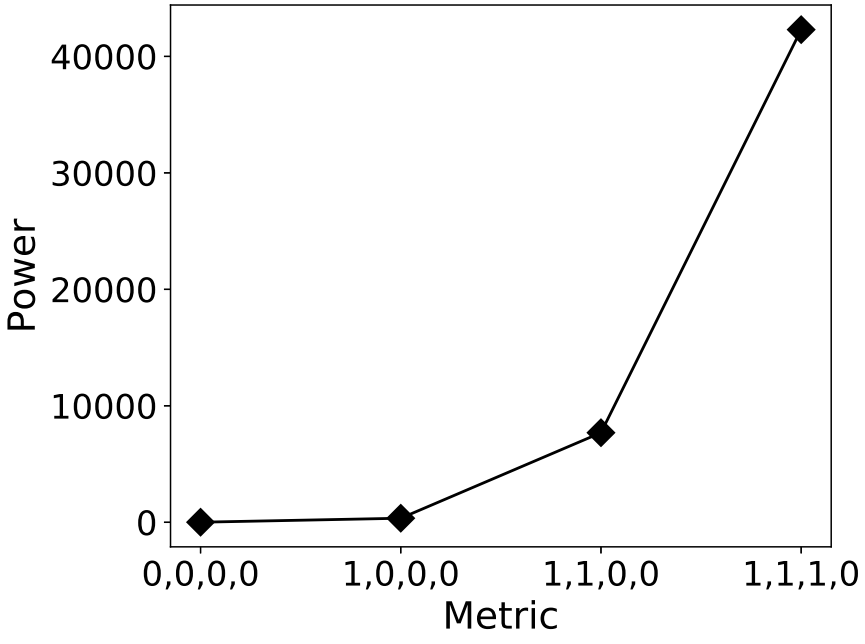
- Both t_2 and t_3 can perform type I attacks against t_1 since both are co-residing with the victim at some physical hosts (*Level_0*). However, t_1 has more VMs sharing the same hosts as t_2, and hence has smaller distance with respect to t_2 than t_3 ($0.005 < 0.049$). Therefore, the impact of t_2 attack on t_1 VMs will be higher than the impact of t_3 attack. Note that similar reasoning can be applied on the network distances.

- Both t_2 and t_3 can perform type II attacks either at the rack-level or at the pod-level as they have many VMs deployed over $Rack_1$ and $Rack_2$. Since the distance of t_1 with respect to t_2 is equal to his distance with respect to t_3 both at the rack-level ($D^1_{\varsigma} = 0.5$) and at the pod-level ($D^2_{\varsigma} = 0$), attacks from the two tenants will have similar impact on t_1.

We further evaluate through simulations how the compute distance changes while increasing the cloud data center's workload and size. As illustrated in Figure 5.7, our compute distance at *Level_0* captures the expected increase in the degree of resource sharing while increasing the total number of data center's VMs (see Figure 5.7(a)), and the decrease in resource sharing while increasing the data center's size (see Figure 5.7(b)), which shows the effectiveness of our metric.



(a)



(b)

Figure 5.5: (a) Attacker's requirements, and (b) average attacker's requirement in terms of power consumption to disrupt services of a victim at different levels of the cloud infrastructure. The X axis corresponds to possible physical distance metric values

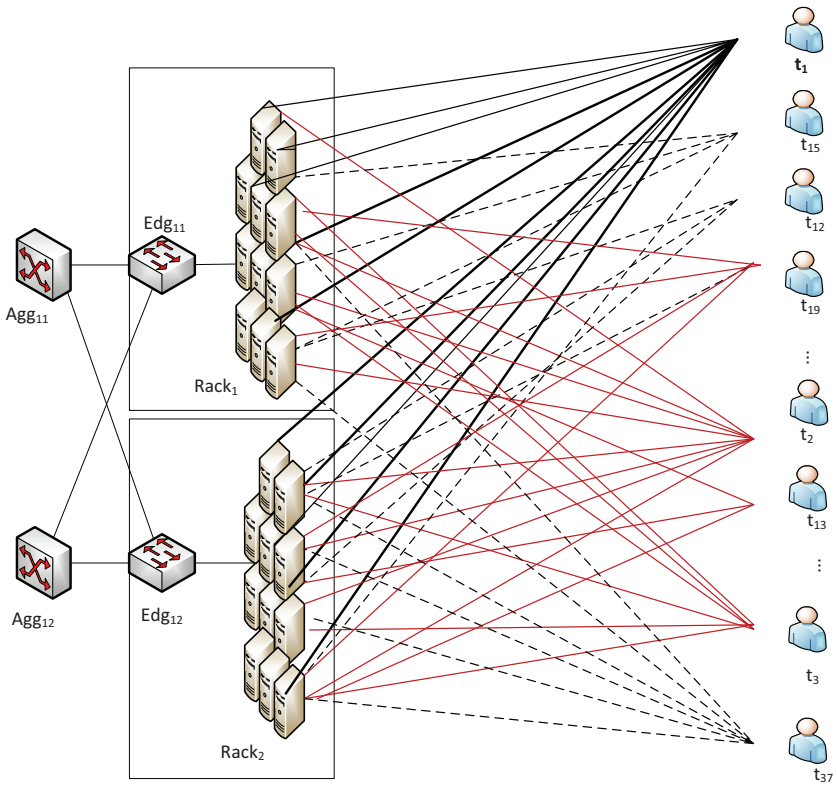
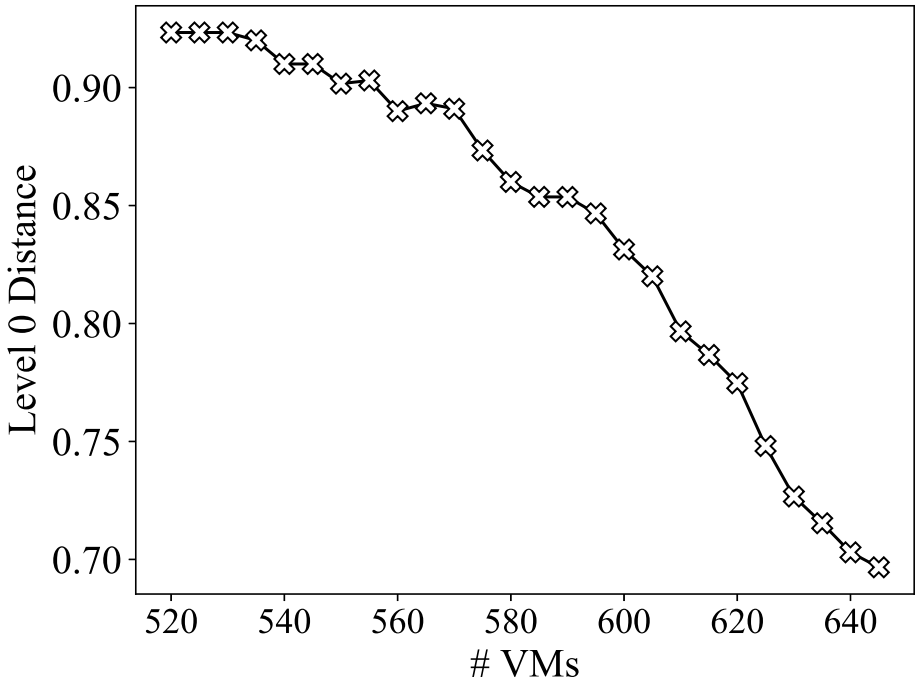
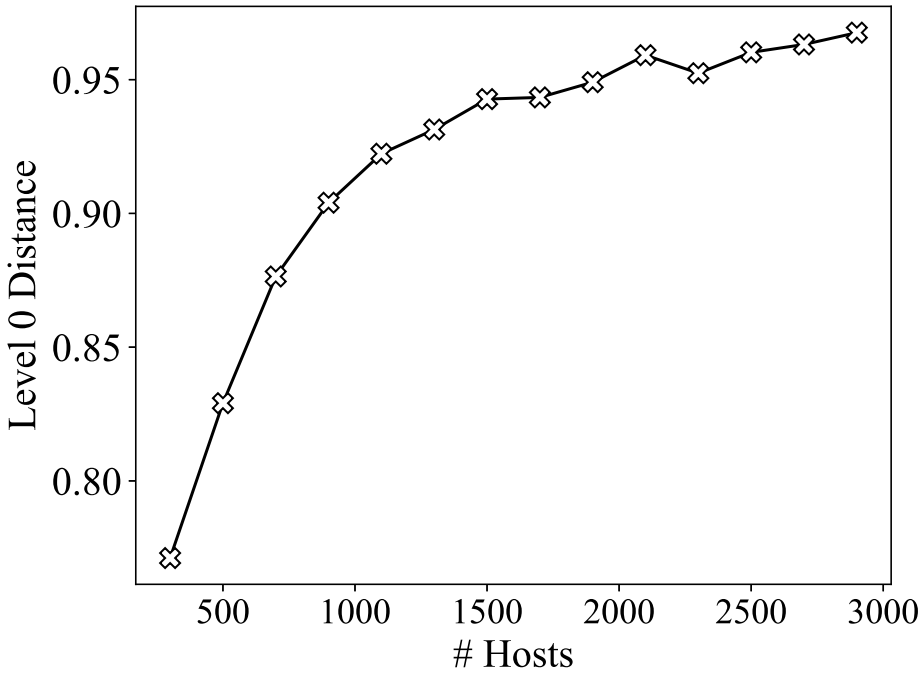


Figure 5.6: Part of a real cloud data center topology constituted of 22 physical servers organized into two racks hosting 372 VMs belonging to 37 tenants



(a)



(b)

Figure 5.7: Compute distance at *Level_0* (a) while increasing the number of data center's VMs, and (b) while increasing the number of data center's hosts

Racks	$Rack_1$											$Rack_2$										
Hosts	*S1*	*S2*	*S3*	*S4*	*S5*	*S6*	*S7*	*S8*	*S9*	*S10*	*S11*	*S12*	*S13*	*S14*	*S15*	*S16*	*S17*	*S18*	*S19*	*S20*	*S21*	*S22*
t_1	**0**	**1**	**0**	**0**	**0**	**4**	**6**	**4**	**4**	**8**	**9**	**4**	**10**	**4**	**6**	**16**	**1**	**4**	**8**	**7**	**2**	**0**
t_2	4	0	0	2	0	4	6	12	8	5	4	2	3	6	6	6	1	10	2	0	0	1
t_3	0	0	0	0	0	0	2	1	1	1	1	0	5	0	1	0	0	0	0	2	0	0

Table 5.2: Number of VMs of tenants t_1, t_2 and t_3 insider each physical host in the considered part of the cloud data center

5.4.3 Case Study 3 (Quantitative Auditing)

In this case study, we show how our metrics can be used to quantitatively audit the compliance of deployed virtual infrastructures against tenants' requirements in terms of the distance. As a continuity of the case study in Section 5.4.1, we assume that tenant t_A's security team is aware of the multi-tenancy attacks and specifies accordingly a compute distance requirement for his own VMs against other tenants as $D_{\varsigma}(t_A) = (1, 1, 0.5, 0)$.

To evaluate the compliance deviation, the CSP first measures the distances for the current cloud deployment, then he checks the measured distances against the required one to evaluate the deviations. In the following, matrices $M_{\varsigma}(t_A)$ and $\triangle D_{\varsigma}(t_A)$ respectively report measured distances and deviations for t_A with respect to tenants t_B, t_C and t_D (represented respectively by the first, second and third row in matrices) based on the cloud configuration in Figure 5.4 and the required compute distance $D\varsigma(t_A)$. The obtained deviation matrix reports how much the current cloud implementation has deviated from the required specification, where higher values correspond to more deviations and consequently reduced distances.

$$M_{\varsigma}(t_A) = \begin{pmatrix} 0.625 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \triangle D_{\varsigma}(t_A) = \begin{pmatrix} 0.375 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

We integrated the described auditing approach into OpenStack [111], one of the most commonly used infrastructure management platforms. Algorithm 1 describes the compliance deviation evaluation procedures based on the required distances. First, the procedure *Per_Tenant_Implemented_Distance* measures the implemented distances based on data collected mainly from Nova[3] database for the compute distances, and on the OpenDaylight[4]

[3]OpenStack Nova [111] is a project designed to provide on-demand access to compute resources
[4]OpenDayLight is an open source SDN controller

[68] database for the network distances. Then, the procedure *Per_Tenant_Deviation* evaluates the deviation with respect to different tenants accommodated by the same data center. Finally, a matrix is generated to report deviations at different cloud levels. Note that if tenant t_A has multiple outsourced virtual infrastructures, he can specify distance-based policies with multiple rules according to the sensitivity-level of different workloads.

Algorithm 1 Compliance Deviation Evaluation

procedure GLOBAL_DEVIATION($D(t)$)
  for each tenant t' belonging to the data center **do**
    $M(t,t')$=Per_Tenant_Implemented_Distance(t, t')
    $\triangle D(t,t')$=Per_Tenant_Deviation($D(t)$, $M(t,t')$)
  Return($\triangle D(t)$)
procedure PER_TENANT_DEVIATION($D(t)$, $M(t,t')$)
  for $i = 0 \quad to \quad 3$ **do**
    $\triangle D[i] = 0$
  if $M[0] < D[0]$ **then**
    $\triangle D[0] = D[0] - M[0]$
  Return ($\triangle D(t,t')$)

To evaluate our quantified auditing approach, we simulate the K-ary tree data center topology [112] with 40 core switches, and deploy the virtual infrastructures of 20 tenants. We assign tenants' VMs to servers in a round robin fashion and build their connections in switches at different levels.

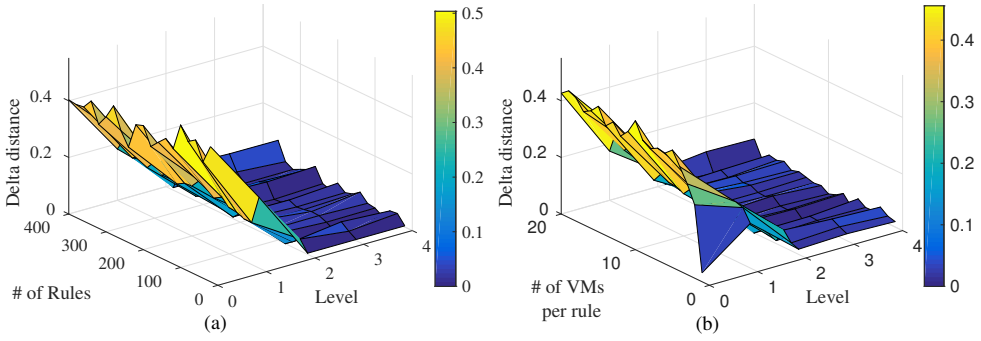


Figure 5.8: Changes in the deviation vectors (a) while varying the number of rules, and (b) while varying the number of VMs per rule

In Figure 5.8(a), we fix the number of VMs per rule to 20 and vary the number of rules, whereas in Figure 5.8(b), we fix the number of rules to eight and vary the number of VMs per rule. In both figures, we can notice that the most significant deviations (delta distances) are recorded for *Level_0* (up to 0.45), which correspond to the host-level. This is due to the higher security threats related to host-level co-residency (type I attacks), leading t_A to set higher distances at *Level_0* compared to other levels. Therefore, deviations from those distance requirements drastically decimate the overall security with respect to the distance. As for *Level_1* and beyond, the deviation average does not exceed 0.1. This stems from the less significant security threats at higher levels leading t_A to relax the requested distances to reduce costs. Note that our approach is flexible to accommodate different tenants' security needs as they could specify their distances at deployment time.

5.4.4 Discussions

Based on the presented case studies, we can conclude that the physical distance correlates with the degree of difficulty for multi-tenancy attacks, while the compute and network distances provide the potential impact of those attacks according to the degree of resource sharing at each level. Therefore, our distance metrics can be applied for evaluating the preliminary tenant pair-wise multi-tenancy risk incurred by a given cloud deployment. To this end, the CSP first defines a diagonal probability matrix P, where each element p_{ii} corresponds to the likelihood of different types of multi-tenancy attacks at *Level_i*. Those probabilities can be defined using existing approaches as presented in [63]. Then, the multi-tenancy risk for a given tenant t with respect to another tenant t' will be given by the weighted norm of tenant t's distance with respect to tenant t'. This can be expressed as $Risk(t,t') = ||D(t,t')||_P = \sqrt{D(t,t')^\top \times P \times D(t,t')}$. Since potential attackers' identity is not known a priory, the overall multi-tenancy risk for a tenant t can be defined as the average of tenant pair-wise risks given by $Risk(t) = \frac{\sum_{t' \in T \setminus \{t\}} Risk(t,t')}{|T|-1}$, where T is the set of

tenants of the cloud data center. Note that the multi-tenancy risk can be defined at both compute and network levels.

Due to the dynamic nature of the cloud, calculated metric values might be quickly invalidated by various management operations such as VM migration. By integrating our metrics into the cloud infrastructure management platform (e.g., OpenStack [111]), the CSP can monitor those operations and evaluate our distance metrics at runtime to continuously control the co-residency threats. Additionally, in the current version of our metrics, we assume that all VMs are equally sensitive, which might not be the case for some applications (e.g., three-tier applications). We leave the study of those directions as part of future work.

5.5 Summary

In this work, we proposed the physical, compute and network distance metrics to quantify proximity between tenants inside cloud deployments. We showed through different case studies and through integration into OpenStack the effectiveness and applicability of those metrics to evaluate multi-tenancy threats. We believe our suite of metrics can be extended to evaluate other threats in cloud. Therefore, it should be considered as a first step toward a more general tool-set for threat evaluation in cloud environments.

Chapter 6

ProxiMet: Security Metrics for Evaluating and Mitigating Co-residency Threats in Public Cloud

6.1 Introduction

Multi-tenancy allows a cloud service provider (CSP) to serve multiple customers using the same physical resources to achieve the desired cost effectiveness. On the other hand, multi-tenancy is also a double-edged sword as it significantly expands the attack surface of cloud tenants by exposing their most valuable or sensitive assets to other tenants sharing the same physical resources. Existing works have demonstrated real-life attacks for forcing attackers' virtual machines (VMs) to co-reside with targeted VMs, either inside the same host or at higher proximity levels (e.g., the same rack [28]), in commercial public clouds including Amazon EC2 (even after the network management has been hardened through Virtual Private Networks), Google GCE, and Microsoft Azure [28, 29, 113]. Once the

malicious tenant achieves co-residency with the victim, he/she can launch various cross-tenant attacks, such as side-channel attacks [66, 24, 104], host-based DoS attacks [104], resource freeing attacks [114], and power attack [102].

Today's cloud providers are well aware of such co-residency-related threats, and they are often under obligation to protect their tenants against such threats through isolation [115, 116], either as part of the service level agreements (SLAs) or to demonstrate compliance with security standards (e.g., ISO 27002/27017 [73, 8] and CCM 3.0.1 [7]). Nonetheless, addressing co-residency threats remains a challenging issue in that completely avoiding multi-tenancy is impractical since it defeats the cost-effectiveness purpose of cloud computing. Instead, cloud providers must balance security with cost effectiveness through resource sharing as a partial remediation. However, to achieve an optimal trade-off among those factors, a prerequisite is to be able to evaluate the co-residency threats of clouds, i.e., to answer the question: *To which extent a tenants' virtual infrastructure deployment is exposed to potential co-residency threats?*

As demonstrated in Table 6.1, many existing works can provide a partial answer to the above question. However, we can also see those works largely focus on detecting specific co-residency attacks through monitoring certain metrics about resource usage. While the proposed metrics are effective for detecting such attacks at run-time, applying them to evaluate and mitigate the co-residency threats of clouds as a preventive solution has two major limitations. First, since the metrics are designed to detect attacks as they happen, the cloud provider cannot apply the metrics proactively to evaluate or mitigate such threats before they actually happen. Second, as can be seen in the table, those metrics are very specific for each attack, and consequently, the cloud provider must deal with a larger number of such metrics, if he/she wants to cover most of the known attacks; even if the cloud provider is willing to do so, it still may not work for future, unknown attacks which might involve other metrics than these, or those attacks that are stealthy in nature [117]. There also exist

some generic security metrics [61, 62] which can cover multiple attacks, but those mostly rely on the fact that the attacker is known (a detailed review of the related work will be given in Section 2.3.2).

In this work, we present *ProxiMet*, a suite of novel security metrics to quantify the proximity between cloud tenants' virtual infrastructures. Our key observation is that, as demonstrated in Table 6.1, although specific co-residency attacks may involve very different resources and thus require different metrics of resource usage for detection (e.g., CPU usage for a side channel attack and power consumption for a power attack), they all share similar prerequisites in terms of co-residency, i.e., attackers must first gain sufficient co-residency with a victim. Based on this observation, we first extract such common co-residency prerequisites from co-residency attacks along two dimensions, namely, the *co-residency extent*, and the *co-residency intensity*, which captures two different aspects of co-residency threats. Second, we define metrics to evaluate those co-residency dimensions based on the proximity between tenants' virtual infrastructures according to their cloud deployment. Third, we show the usefulness of our metrics through a case study based on data collected from a real cloud. We further assess the effectiveness of our security metrics through simulations using CloudSim based on two well-known cloud VM-placement policies [61]. The main contributions of this work are as follows:

- We examine various co-residency attacks and extract their common co-residency prerequisites in order to define our proximity metrics.
- We show through a real cloud-based case study the effectiveness of our metrics and how they enable the control and mitigation of the co-residency threat level through cloud management operations.
- We further conduct extensive simulations to show the relationship between our metrics and co-residency attack types.

Attacks	Attack-Specific Metrics							
	IO intensity [25]	Bit-rate/ Error rate distribution [118]	Number of instructions per second [104]	CPU overhead [119]	CPU time [114]	Throughput [19]	Power consumption [102]	Available bandwidth [106]
Last-level cache [15]	•							
Hammer attack [24]	•							
L2 cache exploration [118]		•						
Whispers [120]		•						
Host-based DoS attack [104]			•					
CPU consumption attack [119]				•				
Resource-freeing attack [114]					•			
Power attack [102]				•		•	•	
CIDoS attack [121]		•						
Bandwidth saturation [106]								•

Table 6.1: An excerpt of metrics used in detecting several co-residency-based attacks. The symbol • means the metric can be used in the detection of the attack

6.2 Preliminaries

This section gives our cloud infrastructure model, threat model, and running example.

6.2.1 Multi-Level Cloud Infrastructure Model

In the following, we model cloud tenants' virtual infrastructures and their deployment inside the physical infrastructure. A virtual infrastructure consists of virtual resources including VMs (deployed in physical hosts) and virtual networks (deployed in infrastructure switches). The assignment of VMs to physical hosts is usually decided based on specific placement policies. Figure 6.1 illustrates an entity-relationship diagram that captures our model, where nodes represent physical and virtual resources and arrows depict their relationships (e.g., mapping or association). The cloud physical infrastructure consists of nested management zones shown as aggregated nodes (e.g., several hosts can be aggregated into a rack zone, and multiple rack zones can be aggregated into a pod zone). We use the terms single-node zone and multi-node zone, to refer to zones at the host level (servers), and zones at higher levels of the hierarchy, respectively. Our model captures the tree-based hierarchical network topologies (e.g., basic-tree, fat-tree and clos networks) [122], currently in-use in several data centers' designs [112], [123], and our model can be adapted to other topologies [122].

6.2.2 Threat Model

As in [29], we assume a malicious tenant has the same privilege of a regular tenant to access the interface for launching and terminating his/her own VMs. As in [29, 28], we also assume a malicious tenant can infer the VM-placement policy used in the cloud and consequently craft special launch strategies to increase his/her chances of co-residency with the victim. Our in-scope attacks include any cross-tenant attacks that require an adversary

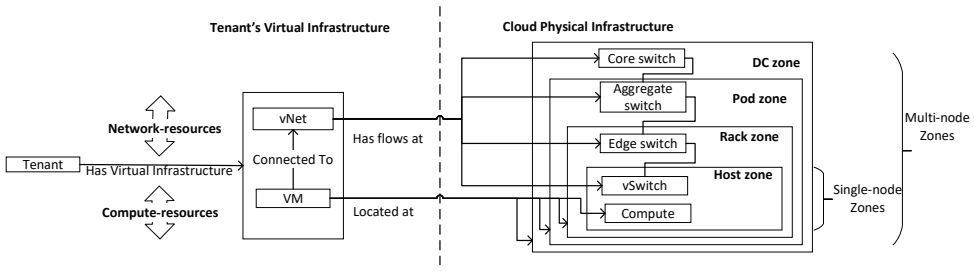


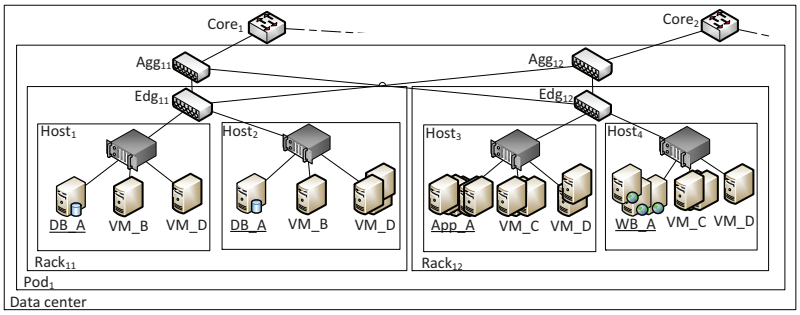
Figure 6.1: A multi-level model capturing tenants' virtual infrastructures, the cloud physical resources, and their relationships

to co-reside certain resources with the resources of a victim tenant. Any attacks that involve administrator privileges or target the cloud infrastructure or provider are out of the scope. Consequently, we assume the information collected from the cloud infrastructure management system to calculate our metrics are trusted.

Our metrics are meant for evaluating the general security posture of clouds against the in-scope attacks, and they are not designed to detect such attacks, identify the malicious tenant, or pinpoint the vulnerabilities. In fact, our metrics are to be applied before the attacks actually happen (unlike [25]), and without any prior knowledge of the attacker's identity (unlike [61]). Thus, our metrics are complementary to other attack-specific security solutions, e.g., attack detection and vulnerability analysis.

6.2.3 Running Example

To build intuitions, we discuss our running example shown in Figure 6.2. In this cloud deployment, we assume *Tenant_A* has a three tier application composed of two database servers (*DB_A* in hosts 1 and 2), five application servers (*App_A* in host 3), and three web servers (*WB_A* in host 4). The right-side table of Figure 6.2 shows the total number of VMs belonging to all four tenants (including *Tenant_A*) in each host.



The number of VMs over hosts:

Hosts	A	B	C	D	Total
$Host_1$	1	1		1	3
$Host_2$	1	1		3	5
$Host_3$	5		15	2	22
$Host_4$	3		20	1	24
Total	10	2	35	7	54

Figure 6.2: An example showing the deployment of the virtual infrastructures of tenants A, B, C and D inside the data center

Assume the cloud provider would like to evaluate the level of co-residency threats of *Tenant_A*'s virtual infrastructure, without knowing which of the three tenants (*Tenant_B*, *Tenant_C*, and *Tenant_D*) would be malicious, and which attacks (as shown in Table 6.1) would be used. At the same time, the co-residency status shown in the figure would certainly determine (as a necessary but not sufficient condition) whether any such attack may succeed. For example, *Tenant_C* cannot launch any side-channel attack on the database servers of *Tenant_A* (since *Tenant_C* is not co-residing with *Tenant_A* on hosts 1 and 2), whereas he/she can potentially stage server-level or rack-level power attacks [102] or host-based DoS attacks [104] against the web and application servers, since he/she has a large number of VMs co-residing with *Tenant_A* on hosts 3 and 4. Clearly, it is not straightforward to design security metrics that can effectively evaluate the co-residency threat levels according to all possible attack scenarios, which we will pursue in the remainder of this chapter.

6.3 Methodology

In this section, we first extract some common aspects of co-residency attacks then we define our proximity metrics.

6.3.1 Extracting Common Aspects of Co-Residency Attacks

The co-residency attacks shown in Table 6.1 may look very different at first glance as they employ different techniques and have different objectives. However, those attacks have in common one prerequisite, i.e., *the malicious tenant must first co-reside with the victim*. Therefore, we start by examining how such a prerequisite applies to each attack in more details. In Table 6.2, we classify the attacks into two categories (the list of attacks is not meant to be exhaustive; other, including future or unknown, attacks may also fit into those categories), and we discuss their co-residency prerequisite as follows.

Type I Attacks. In Table 6.2, the first seven attacks are very different in nature, however, they have commonalities with respect to co-residency prerequisites. For instance, last-level cache attack, hammer attack and L2 cache exploration attack are all side/covert channel attacks whose objective is either to steal sensitive information from the victim's resources to breach confidentiality, or to establish illicit communication paths exploiting co-residency. Host-based DoS attacks attempt to compromise victim's hosts availability by deploying well-tuned controlled VMs over those hosts, while resource freeing attack objective is to use shared hosts' resources on the victim's expense. Another example of type I attacks is the hyperjacking attack, where a single malicious VM constitutes the only prerequisite to exploit the hypervisor (e.g. CVE-2015-3456 [124]) and take control over co-hosted VMs. The common aspect of all those attacks is to place *at least one* malicious VM inside the same host with the victim's VMs. Through those attacks, the adversary may target either *a fraction* of the victim's resources (e.g., database or storage servers), or *all* of them, which

Attack type	Attack	Common aspects			
		Extent		Intensity	
		Fractional	Complete	At least one	As many as possible
Type I	Last-level cache [15]	✓	✓	✓	
	Hammer attack [24]	✓	✓	✓	
	L2 cache exploration [118]	✓	✓	✓	
	Whispers [120]	✓	✓	✓	
	Host-based DoS attack [104]	✓	✓	✓	
	CPU consumption attack [119]	✓	✓	✓	
	Resource freeing attack [114]	✓	✓	✓	
Type II	Power attack [102]	✓			✓
	CIDoS attack [121]	✓			✓
	Bandwidth saturation attack [106]	✓			✓

Table 6.2: Co-residency attacks and their common aspects of co-residency prerequisite

is referred to as complete coverage [125, 66].

Example 6.1 (CPU consumption attack [119])**.** *CPU consumption attack is a co-residency-based attack that exploits hypervisors' vulnerabilities. Under this attack, one malicious VM can increase the CPU cycles usage on a host from the legitimate limit (40%) up to 85%. An attacker whose objective is to disturb the right functioning of a specific victim's application and increase her expenses (since customers are charged based on the amount of time their VMs are running) will place at least one VM in each host accommodating the victim's resources in order to maximize the overall victim's charged costs. Therefore, a necessary condition for this attack to succeed is to co-reside with the largest number of victim's VMs. It follows that its percentage of success is determined by the degree of coverage attacker achieves with respect to his victim's resources. We refer to this coverage-level as extent since it defines the degree of damage that could be caused to the victim assuming that all her resources are equally important.*

***Type II Attacks*.** This category includes attacks exploiting vulnerabilities in the cloud management strategies (e.g., resource over-subscription [126, 102]). Examples are power attack [102] and bandwidth saturation attack [106]. These attacks aim at affecting the availability of a given zone of the cloud infrastructure, which could be at the host-level, rack-level or higher levels in the cloud hierarchy (as described in Section 6.2). To succeed and maximize the effect, those attacks require *as many malicious resources as possible* to be placed inside the targeted zone. Furthermore, they typically target a fraction of their victim's resources located at a specific zone since larger targets, e.g., the data center, are significantly more difficult to compromise.

Example 6.2 (Power attack [102])**.** *Power attack exploits the power over-subscription vulnerability, which consists of overloading a power supply with more workloads than it supports with the assumption that workloads will never reach their peak simultaneously. If the*

attacker succeeds to place many VMs inside a zone alimented with the same power facility, then he can generate simultaneous power spikes, which would lead to power outage when the power consumption exceeds the power capacity for that specific zone. It has been shown in [102] that one malicious VM can increase the power consumption of the hosting machine with up to 95 watt while a benign VM causes a power increase of 45 watt only. Obviously, the more the number of controlled VMs attacker could place in the targeted zone, the higher the risk related to those attacks would be. The number of VMs needed will be depending though on the power supply capacity. Indeed, the larger the rated capacity of a zone's circuit breaker, the more VMs the attacker needs to place in it to cause its outage. Additionally, the larger the zone attacker is targeting, the more controlled VMs need to be deployed to increase the power consumption, since smaller zones converge faster to their peak power[1].

As demonstrated through above discussions, those seemingly different attacks indeed share the common prerequisite of co-residency, and such prerequisite may be characterized along two diagonal dimensions as follows.

- *Co-residency Extent.* This aspect of the co-residency prerequisite reflects the level of coverage the attacker wants to achieve with respect to the victim's resources. In Table 6.2, the extent columns show whether each attack may target a fraction (e.g., type II attacks) of the victim's resources or all of them (e.g., type I attacks).

- *Co-residency Intensity.* This aspect of the co-residency prerequisite reflects the amount of malicious resources the attacker needs to place at a given zone for the attack to succeed. In Table 6.2, the intensity columns show whether each attack may require at least one (type I attacks), or as many as possible (type II attacks) malicious VMs to co-reside with the victim.

[1] It has been reported in [109] that racks reach 96% of their peak power, while pods and data centers do not exceed respectively 86% and 72% of their peak power.

6.3.2 Proximity Metrics

To quantify the common aspects of co-residency prerequisites depicted in Table 6.2 and discussed in Section 6.3.1, we define two metrics, namely, *the co-residency extent*, and *the co-residency intensity*. The first metric is designed to capture the extent aspect, and the last metric captures the intensity aspect of co-residency prerequisites. In addition, we propose a generic metric, namely, *the multi-tenancy attack surface*, which does not directly map to those common aspects, but evaluates the co-residency threat from a more general perspective as will be discussed later in this section.

For notations, let t and z be a tenant and a zone in the cloud data center *DC*, where z could be either a single-node zone (i.e., a host) or a multi-node zone (a rack, a pod, or the data center as a whole). Let R_t be the set of resources belonging to a tenant t. We denote by z_t the set of zones in which tenant t has at least one resource deployed. We use the notation $|\mathrm{R}_t|$ for the number of resources of tenant t. Table 6.3 summarizes our notations along with their description.

Co-residency Extent. This metric evaluates the extent aspect of co-residency attacks. To this end, we first calculate the resource distribution to capture how many resources of a given tenant t (the victim) are sharing zones with another tenant t', the potential attacker, by summing up t's resource distribution values over all the zones z that are shared between t and t' ($z \in z_t \cap z_{t'}$). This can be expressed as $\textit{pairwise-extent}(t,t') = \sum_{z \in z_t \cap z_{t'}} RD(t,z)$.

Based on this, we define the co-residency extent, as the highest level of pairwise extent with respect to all the tenants having at least one zone shared with tenant t. This is expressed as follows:

$$\textit{Co-residency-extent(t)} = max_{t' \in T \setminus \{t\}} \textit{pairwise-extent}(t,t')$$

This metric reports the upper bound threat-level related to type I co-residency attacks

Var.	Description
T	The set of tenants inside the data center *DC*
Z	The set of zones inside the data center *DC*
T_z	The set of tenants having at least one of their virtual resources in z
R_t	The set of virtual resources belonging to tenant t
$R_{t,z}$	The set of virtual resources of tenant t in zone z
R_z	The set of all virtual resources in zone z
ZC(t)	*Zone coverage* is the number of zones accommodating tenant t's resources. We denote $ZC(t) = \|z_t\|$. Depending on the zone type, we call $ZC_{host}(t)$, $ZC_{Rack}(t)$ and $ZC_{pod}(t)$ the host-level coverage, rack-level coverage, and pod-level coverage, respectively.
$RD(t,z)$	*Resource distribution* is the ratio of resources belonging to t located at z. We denote it $RD(t,z) = \frac{\|R_{t,z}\|}{\|R_t\|}$
RA(t,z)	*Resource abundance* is the fraction of resources belonging to t in z over all resources in that zone (regardless of tenants they belong to). More formally $RA(t,z) = \frac{\|R_{t,z}\|}{\|R_z\|}$
$ZS(t,z)$	*Zone sharing* is the ratio of tenants effectively co-residing with t in z (excluding t itself). More formally, $ZS(t,z) = \frac{\|T_{z_t}\|-1}{\|T\|-1}$

Table 6.3: Summary of the notation used in Proximet

according to the current cloud deployment.

Example 6.3. *We evaluate the co-residency extent for Tenant_A according to the deployment depicted in Figure 6.2. Table 6.4 reports the pairwise extent for Tenant_A with respect to other tenants. From this table, we can conclude that the co-residency extent for Tenant_A is equal to* 1. *We can see that the co-residency extent coincides with Tenant_D, who shares all the hosts with Tenant_A. Hence, the metric provides an upper bound to the threat of a co-residency attack whose prerequisite is to maximize the co-residency extent (e.g., Hammer attack [24]).*

Co-residency Intensity. We define this metric to evaluate the intensity aspect of type II co-residency attack. For a given tenant t, which is the victim, and a given tenant t', the potential attacker, we first measure the resource abundance of t' at each zone z that is shared with

	$Host_1$	$Host_2$	$Host_3$	$Host_4$	**Per-tenant coverage**
Tenant_B	0.1	0.1	0	0	0.2
Tenant_C	0	0	0.5	0.3	0.8
Tenant_D	0.1	0.1	0.5	0.3	**1**

Table 6.4: Host-level pairwise extent with respect to *Tenant_A*

	$Host_1$	$Host_2$	$Host_3$	$Host_4$	**Per-tenant intensity**
Tenant_B	0.33	0.2	0	0	0.13
Tenant_C	0	0	0.68	0.83	**0.37**
Tenant_D	0.33	0.6	0.09	0.04	0.26

Table 6.5: Host-level pairwise intensity with respect to *Tenant_A*

t, we sum up the resource abundance values over all shared zones ($z \in z_t \cap z_{t'}$), then, we normalize the obtained value by the zone coverage of tenant t, the potential victim. This is expressed formally as $\textit{pairwise-intensity}(t,t') = \frac{\sum_{z \in z_t \cap z_{t'}} RA(t',z)}{ZC(t)}$.

Similarly to the previously defined metric, as we are evaluating potential threats, we define the co-residency intensity, which provides an upper bound to the threat of type II co-residency attacks with maximum attacker's resources intensity as a prerequisite, to be the highest level of resource intensity with respect to all the tenants having at least one zone shared with tenant t. Thus, the *co-residency-intensity* is expressed as follows:

$$\textit{Co-residency-intensity}(t) = max_{t' \in T \setminus \{t\}} \textit{pairwise-intensity}(t,t')$$

Example 6.4. *In this example, we evaluate the co-residency intensity for Tenant_A according to the deployment of Figure 6.2. In Table 6.5, column six reports the pairwise intensity. We can see that the co-residency intensity for Tenant_A is equal to* 0.37*, which corresponds to Tenant_C who has the largest host-level VM abundance at $Host_3$ and $Host_4$ inside $Rack_{12}$. This depicts the worst case scenario when there exists an adversary trying to launch a power attack [102] or a bandwidth saturation attack [106].*

Multi-Tenancy Attack Surface. This metric is designed to provide an insight about

	$Host_1$	$Host_2$	$Host_3$	$Host_4$
Resource-distribution	0.1	0.1	0.5	0.3
Host-sharing	0.1	0.1	0.1	0.1
Per-Host resource attack surface	0.01	0.01	0.05	0.03

Table 6.6: Per-host attack surface with respect to VMs of *Tenant_A*

the overall security posture of a virtual infrastructure's deployment with respect to co-residency without necessarily correlating with one of the common aspects to co-residency attacks. We consider that each tenant t's resource is a potential entry point to compromise the tenant's virtual infrastructure, and each tenant t' sharing the same zone is a potential attacker. Hence, we define the multi-tenancy attack surface to measure the attackability against tenant t's resources at a given zone z by combining his resource distribution value at that zone, with his zone sharing value inside the same zone, which is expressed as *Per-zone-attack-surface(t,z)* $= RD(t,z) \times ZS(t,z)$.

To obtain the attack surface related to all of tenant t's deployed virtual resources, we sum up the per-zone attack surface for tenant t over all the zones where the latter exists ($z \in |z_t|$). This can be expressed as:

$$\textit{Attack-Surface(t)} = \sum_{z \in |z_t|} \textit{Per-zone-attack-surface(t,z)}$$

Example 6.5. *According to the deployment of Figure 6.2, and assuming that* $T = 20$*, we calculate the per-host attack surface for Tenant_A. To this end, we first calculate the per-host resource distribution for Tenant_A's VMs, and it's host sharing (reported in rows two and three of Table 6.6, respectively). Then, we evaluate the per-host VM attack surface as reported in row four of Table 6.6. Finally, the overall multi-tenancy attack surface is obtained by summing up the per-host attack surface values, which is equal to* 0.1 *(not shown in the table).*

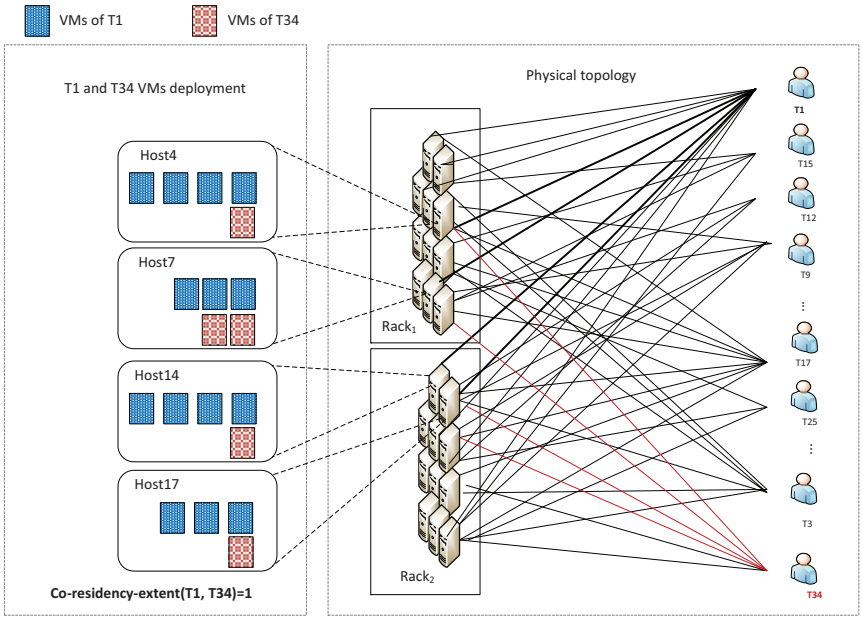


Figure 6.3: Subset of a real cloud data center: (right) distribution of tenants' VMs on a subset of physical hosts and racks, and (left) zoom on the VMs deployment of two tenants ($T1$ and $T34$) inside the physical hosts

6.4 Case Study (Real Cloud Data Center)

The objective of this case study is to show the applicability of our metrics and how they can be used to mitigate the co-residency threats in real world cloud deployments.

This case study is based on a real community cloud hosted at a major telecommunication company. For privacy and security concerns, we collect and anonymize data from part of this cloud composed of 22 physical machines organized into two racks as depicted on the right side of Figure 6.3.

We perform our study on four different datasets (as subsets of the aforementioned cloud data), where each dataset captures VMs deployment in the considered portion of the cloud during one day. Table 6.7 reports the total number of tenants and VMs, and the average number of tenants and VMs per host. Additionally, Figure 6.4 reports the distribution of the number of tenants with respect to the number of VMs they own in different datasets.

Dataset	DS1	DS2	DS3	DS4
# Tenants	33	31	32	21
# VMs	212	301	372	290
Average # Tenants per host	6.45	7.27	7.63	4.77
Average # VMs per host	9.63	13.68	16.9	13.18

Table 6.7: Total number of tenants (with at least one running VM) and VMs, and the average number of tenants and VMs per host for each dataset

6.4.1 Evaluation of our Metrics on a Real Cloud

In this section, we demonstrate the applicability of our metrics on a real cloud. Table 6.8 summarizes the average values of the co-residency extent and the co-residency intensity in the studied datasets. The highest average of both metrics is recorded for dataset DS3, which has the largest total number of VMs and the largest average of VMs per host. More specifically, Figure 6.5 depicts the distribution of our metrics' values for different tenants computed in the four datasets described in Table 6.7. We can see that overall, the co-residency extent metric tends to have higher values for most of the tenants in all datasets, whereas the co-residency intensity tends to take relatively smaller values except for few cases where it could reach large values.

It can be noted that the large values of the co-residency extent metric in all datasets are mainly due to several factors such as the relatively small number of VMs owned by most of the tenants, and the small number of hosts. For instance, as most of the tenants have a number of VMs varying between 1 and 10 VMs (as illustrated in Figure 6.4), hosted in a relatively small number of hosts, the chance of co-residency increases, which results in a relatively high co-residency extent for most of these tenants. The lower values of the co-residency intensity metric can provide insights about the employed VM placement policy (e.g., least policy), which tends to spread the VMs over multiple hosts for resiliency and load balancing purposes. Although those results provide some hints about the relationship between the number of VMs per tenant, the size of the data center, the VM placement policy, and the metrics' values, we delay drawing conclusions until studying the general

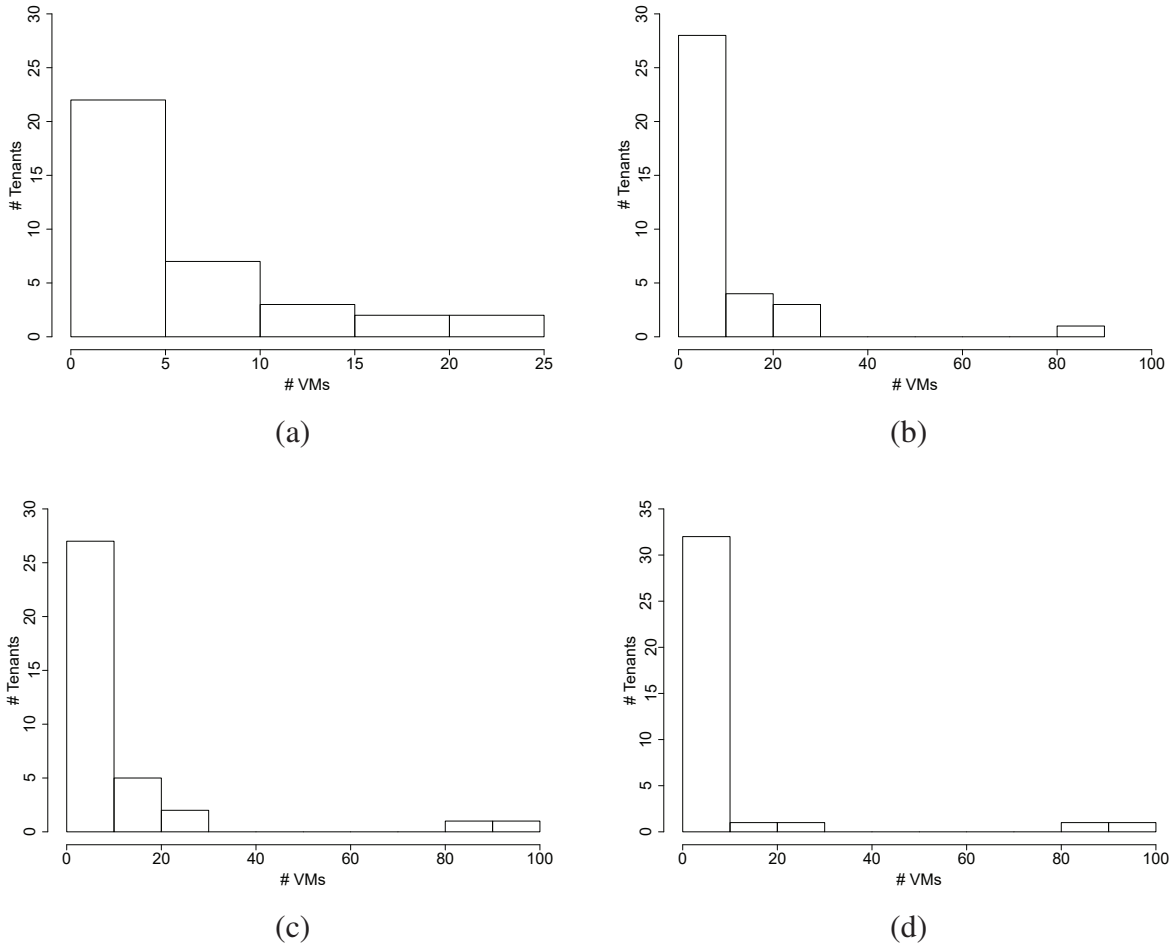


Figure 6.4: Distribution of the number tenants with respect to the number of VMs they own for (a) DS1, (b) DS2, (c) DS3, and (d) DS4. The distribution includes tenants with no running VMs

pattern in the simulation section.

To show the advantage of defining the co-residency extent per tenant as the maximum among the recorded pairwise co-residency extent values, we analyze the distributions of pairwise co-residency extent for all tenants in DS3, the dataset with the highest co-residency extent average (note that similar reasoning can be applied to co-residency intensity). Figure 6.6 characterizes the distributions of the pairwise co-residency extent for different tenants using box-plots[2]. Therein, each box-plot reports five values, namely, the

[2]A box-plot is a rapid visual description of a dataset, which graphically depicts the concentration and spread of numerical data based on quartiles

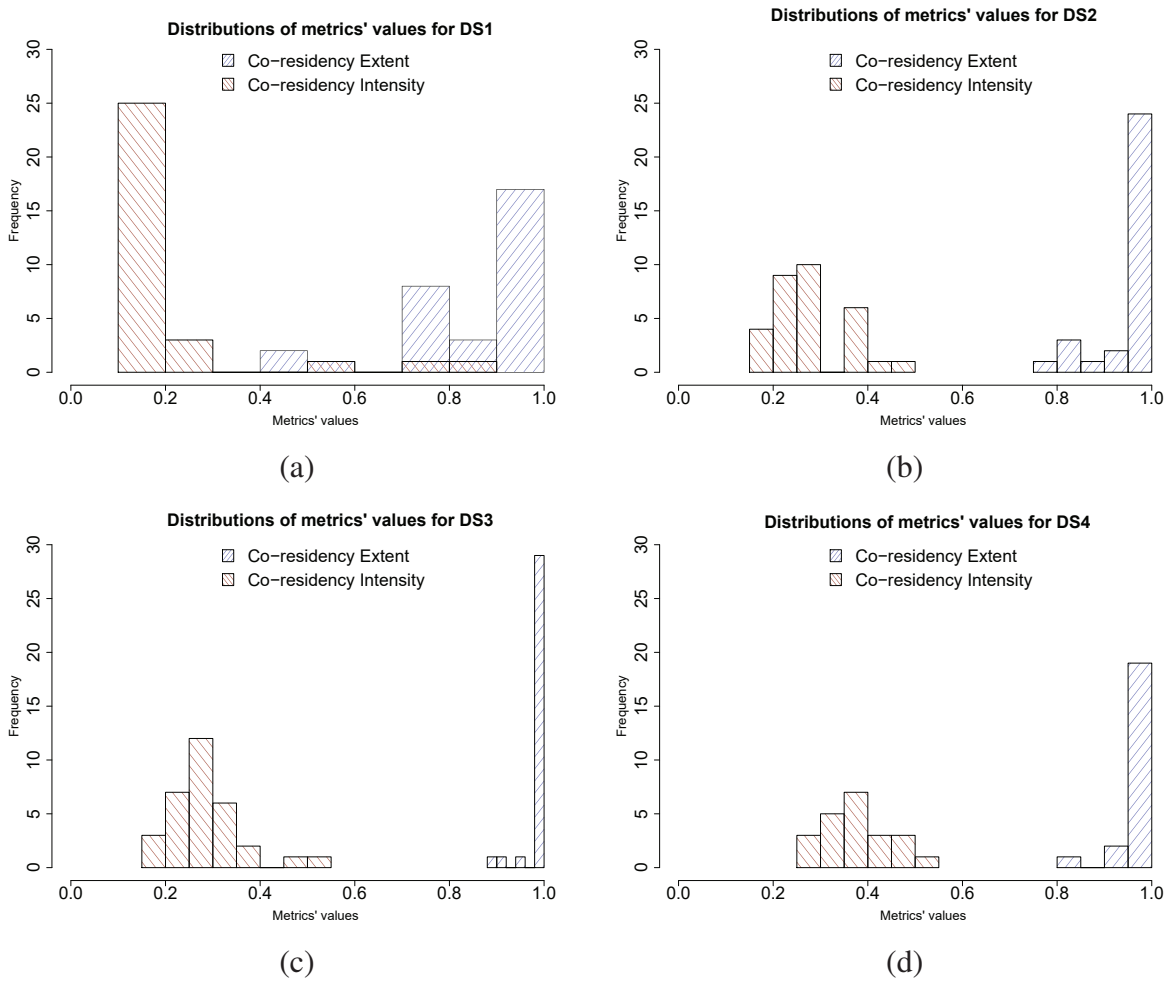


Figure 6.5: Co-residency extent and co-residency intensity in different datasets

minimum value, the first quartile, the median, the third quartiles and the maximum value, for the distribution of pairwise co-residency extent of one tenant in the dataset. We additionally report the average (mean) for each box-plot. For convenience sake, we order the box-plots based on their means and assign a numerical identifier to each one of them. Based on the information provided in Figure 6.6, we can make the following interpretations:

- The five values of box-plots from 24 up to 31 are assimilated to the mean point, which is the maximum value (co-residency extent), and is equal to one. This means that all the pairwise co-residency extent values for the corresponding tenants are equal to one. This high metric value is mainly due to the very small number of VMs owned

Datasets	DS1	D2	DS3	DS4
Average co-residency extent	0.882	0,966	0.992	0.982
Average co-residency intensity	0.217	0.281	0.373	0.285

Table 6.8: Average of co-residency extent and co-residency intensity for different datasets

by the corresponding tenants. Indeed, each one of those tenants has at most one VM, therefore, one co-residency with another tenant will position the metric's value to one.

- Box-plots from 18 up to 23 correspond to tenants owning two VMs, meaning that the pairwise co-residency extent can take two values only, the minimum value is 0.5 for a tenant co-located with one VM only out of the two, and the maximum value is one for a tenant co-residing with the two VMs. We can infer from the average of those box-plots, which varies between 0.65 and 0.75, that the smallest co-residency extent value is more frequent than the largest one. Therefore, choosing the co-residency extent metric to be equal to the largest pairwise co-residency extent, instead of the average, makes our metric more accurate in capturing the upper bound co-residency threat level.

- Box-plots from one up to 17 correspond to tenants whose average number of VMs is 20.4, which is more reasonable compared to the two previous cases. For those tenants, the average of pairwise co-residency extent varies between 0.285 and 0.666. However, we can see also that in all box-plots, most of the observed pairwise co-residency extent values are concentrated on the lower whisker and on the interquartile range (between the first and the third percentiles), whereas large metric values are less observed, which is reflected by large upper whiskers. For instance, in box-plot three, 75% of the observed pairwise co-residency extent values are below 0.481, whereas large metric values are rare. This again shows the benefit of considering the maximum pairwise co-residency extent as the metric value for evaluating the worst

case scenarios of co-residency.

- A malicious tenant a, trying to achieve type I attacks (e.g., side channel attacks) against his victim v, will obviously launch an abnormally large number of VMs or tune a VM launching strategy (as will be detailed in Section 6.5) in order to maximize his coverage of the victim's VMs. This will considerably increase the pairwise co-residency extent with respect to the victim such that any other normal tenant t sharing the cloud will have a lower pairwise co-residency extent with respect to the victim. This can be expressed as $\forall t \in T \setminus \{a\} : \textit{pairwise-extent}(v,t) < \textit{pairwise-extent}(v,a)$. Therefore, choosing the co-residency extent metric to be equal to the highest among the pairwise co-residency extent values constitutes the best option to capture this abnormal increase, since any other aggregated values (e.g., mean, median, percentiles) will loose the accuracy because of the large number of tenants sharing the cloud as can be seen in Figure 6.6.

Clearly, the co-residency extent metric reveals critical information according to the upper bound co-residency threats related to type I attacks. The relationship between our metrics and the attack types will be further elaborated in Section 6.5. Although the above discussion emphasizes the applicability and the effectiveness of our metrics, it is worth noting that as we test our metrics on the data collected only from a small portion of the cloud data center, the metric values we report do not reflect any fact about the co-residency threat levels of the whole data center.

6.4.2 Mitigation through Migration

As discussed in Section 6.3, large values of co-residency extent and/or intensity reveal an increased threat of co-residency attacks. To reduce this threat, the CSP can monitor the metrics' values and take some mitigation actions whenever the values exceed a specific threshold.

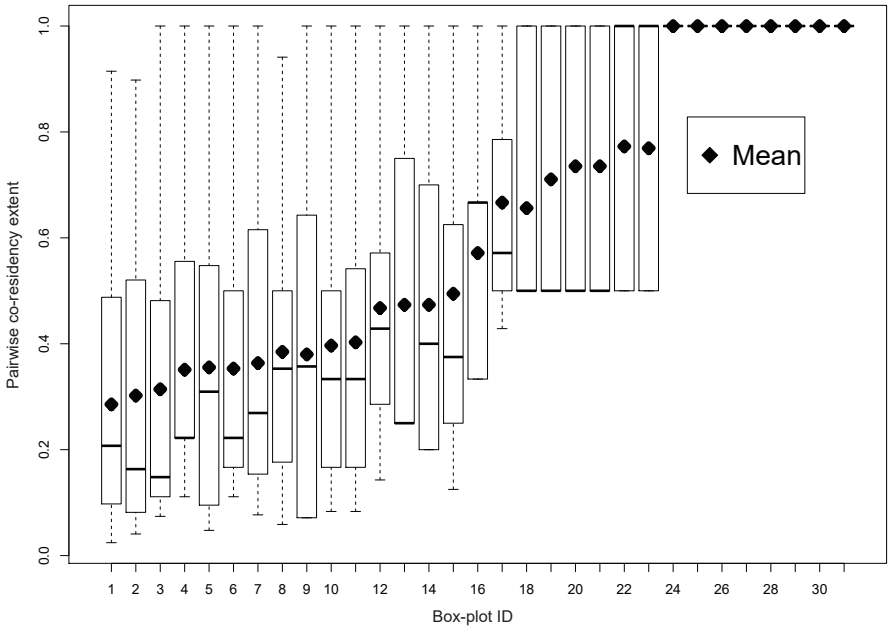


Figure 6.6: Characterizing the distributions of pairwise co-residency extent values for different tenants in DS3

Assume the CSP defines the co-residency extent threshold for his tenants as the average of their pairwise co-residency extent plus a value $\varepsilon = 0.1$. For instance, as depicted in Figure 6.7, the average pairwise co-residency extent of tenant *T1* in dataset *DS*3 is equal to 0.35 (before migration), therefore, the co-residency extent threshold for this tenant should be equal to $TSH(T1) = 0.45$. We recall that the co-residency between $T1$ and $T34$ is illustrated in the left side of Figure 6.3. Therein, one can see that $T34$ has the maximum value of the pairwise co-residency extent with respect to $T1$ (*pairwise-extent*$(T1, T34) =$ *Co-residency-extent(T1)* $= 1$).

Obviously, this configuration does not comply with the threshold set by the CSP. Additionally, if tenant *T34* is an attacker targeting the full coverage of tenant *T1*'s VMs, then the deployment illustrated in the figure will cause the largest scale damage to tenant *T1*. To bring the cloud deployment to a compliant state, and hence reduce the co-residency

threat, the CSP needs to perform a set of migration events. To do so, the VMs to be migrated and the number of source hosts need to be carefully selected in order to minimize the cost related to migration. At the same time, these migrations need to quickly decrease the co-residency extent to prevent large scale damage to the victim and minimize the non-compliance time period.

Algorithm 2 shows an example of heuristics that can be used. First, for each tenant t' having pairwise co-residency extent with respect to t exceeding the threshold TSH, we identify the set of shared hosts, $SourceHosts$, and classify them into an increasing order based on the number of VMs of tenant t' in each host. Then, we choose the first host in the list as the source host (src) of the migration event. Finally, we migrate the VMs of tenant t' inside src to a set of candidate hosts that do not accommodate any of tenant t VMs. The choice of the exact destination host will be depending on the adopted VM placement policy.

Algorithm 2 Threshold-based Migration

procedure MIGRATION(t,ε)
  TSH=*Average-pairwise-extent*(t)+ε
  ExtentArray=Pairwise_Extent(t)
  ClassifyInDecreasingOrder(ExtentArray)
  Max_Extent=GetFirst(ExtentArray)
  while Max_Extent> TSH **do**
    Select-tenant t' such that Pairwise-extent(t,t')=Max_Extent
    SourceHosts=$z_t \cap z_{t'}$
    ClassifyInIncreasingOrder(SourceHosts,$|R_{t',z}|$) // Classify SourceHosts based on $|R_{t',z}|$
    src=GetFirstZone(SourceHosts)
    candidate-destinations=$Z \setminus z_t$
    MigrateVMs(R_t', src, candidate-destinations)
    Evaluate-Extent(t,t')
    ClassifyInDecreasingOrder(ExtentArray)
    Max_Extent=GetFirst(ExtentArray)

Figure 6.7 reports the pairwise co-residency extent of tenant *T1* with respect to other tenants before and after the migration operations. We can see from this figure that the co-residency extent for *T1* decreased from 1 down to 0.444, which is slightly larger than the

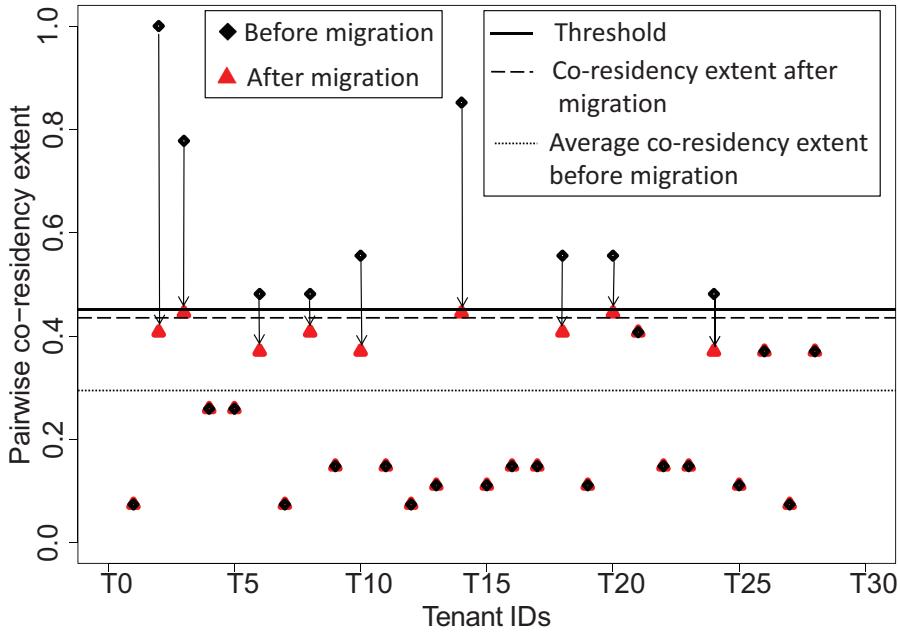


Figure 6.7: Changes in pairwise co-residency extent values for tenant *T1* in dataset *DS*3 before and after migration events

average pairwise co-residency extent before migration (0.35) but lower than the threshold (0.45). This scenario shows the usefulness of our metrics for mitigating the risk related co-residency threats in cloud deployments. It is worth noting that the thresholds for metrics' values can be specified by the CSP based on several factors (e.g., the average pairwise co-residency extent per tenant, the size of the data center, the overall number of VMs). We leave the study of systematic approaches for defining appropriate thresholds, and the design of heuristics providing better optimization for different objectives as part of future work.

6.5 Simulation

In this section, we evaluate our proximity metrics by comparing their results with the percentage of successful simulated attacks under two well-known VM-placement algorithms implemented in CloudSim [127], a widely used cloud environment simulator.

6.5.1 Simulation Environment

First, we evaluate our metrics in a small data center (300 hosts) with two sets of configurations for the hosts. For the first configuration with low frequency usage hosts, we specify the capacity of hosts as 4GB RAM, 1,000 GB storage space and 10,000 MB/s bandwidth. For the second configuration with high frequency usage hosts, we set the capacity of hosts to 40 GM RAM, 10,000 GB storage space and 10,000 MB/s bandwidth. In both configurations, we consider VMs' resource requirements as 512 MB RAM, 10 GB storage space, and 1,000 MB/s bandwidth, which is also the default configuration for VMs in CloudSim. The data center can accommodate 500 VMs in the first configuration, whereas it can host up to 5,000 VMs in the second configuration. The purpose of designing two different capacity configurations is to study the variations of our metrics in data centers with different characteristics. We also increase the number of hosts up to 3,000 to study the impact of larger clouds on the metrics' values.

Background workload. Recent studies demonstrated that VMs' requests for arrival and departure follow the power law distribution [128]. As the latter is not provided by default in CloudSim, we have implemented it to generate realistic workload requests based on statistics in [128].

Placement Algorithms. In the latest version of CloudSim (version 4.4), the libraries for placement algorithms of containers have been added, however, those for VMs are still missing. Thus, we implemented the two placement algorithms, namely, most-VM and least-VM policies, that are widely used in the literature [62] and in open source cloud platforms, such as OpenStack [111]. The most-VM placement policy is based on workload stacking to reduce resource consumption, while the least-VM policy is based on workload balancing.

Attacker Launch Strategies. Similarly to other works (e.g., [62, 29]), we assume that

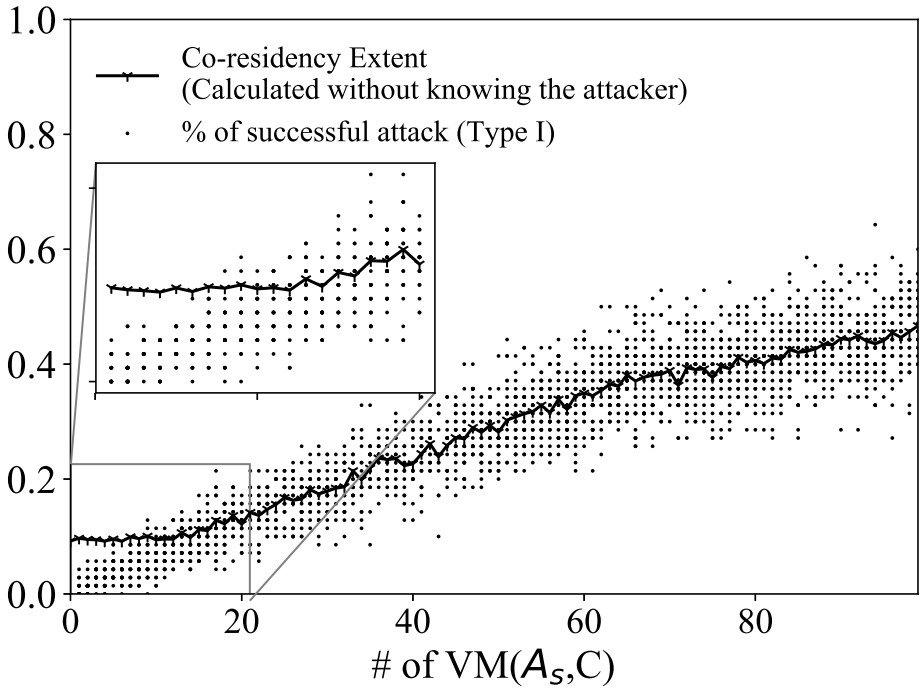
attackers could gather the knowledge about placement algorithms and adjust their VM-launch strategies accordingly to increase their chance of co-residency. For this purpose, we have implemented the launch strategies described in [62]. Note that our metrics are evaluated based on the deployments only (without having any knowledge about attackers) and are independent of attackers' launch strategies.

Parameter settings. In each simulation, we generate 5 hours of workload, which corresponds to around 12,000 VMs' requests. An attacker with a launch strategy, A_s, starts *VM(A_s,C)* VMs within one log configuration, *C*. The metrics' values in all figures are the average for at least 500 iterations.

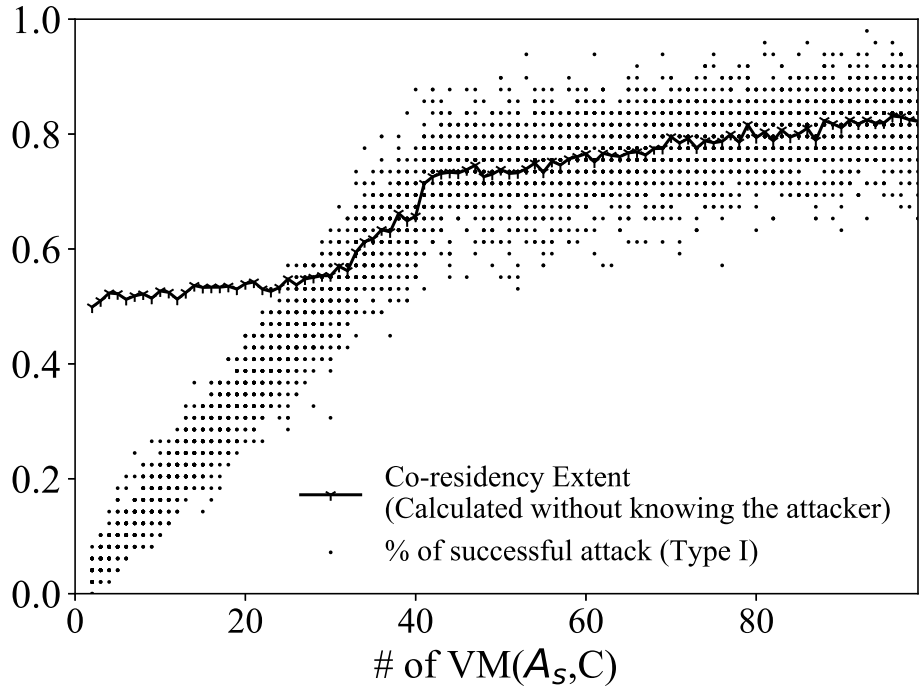
6.5.2 Effectiveness of Proximity Metrics

In the first set of simulations, we compare co-residency extent with the percentage of successful type I attacks (see Table 6.2). In this type of attacks (e.g., Hammer attack, CPU consumption attack), the attacker's objective is to co-reside with the maximum number of victim's VMs. In our simulation, we have chosen one malicious user, with a specific VM launch strategy tuned based on the placement policy, as attacker, and we defined the percentage of successful attacks as the fraction of victim's VMs covered by the attacker. Note that our metrics are calculated only based on the deployment status of the cloud environment, which means that no prior knowledge of the attacker is required in our calculations. Figure 6.8 and Figure 6.9 show the evolution of our metrics and the percentage of successful type I attacks under our two sets of host configurations.

Results and Implications for Co-residency Extent in Most-VM Placement Policy. Figure 6.8 reports the co-residency extent metric compared to the percentage of successful attacks under the most VM placement policy, both for a data center with low frequency usage hosts (in Figure 6.8a) and for a data center with high frequency usage hosts (in Figure 6.8b). Based on those figures, we can make the following observations. First, the

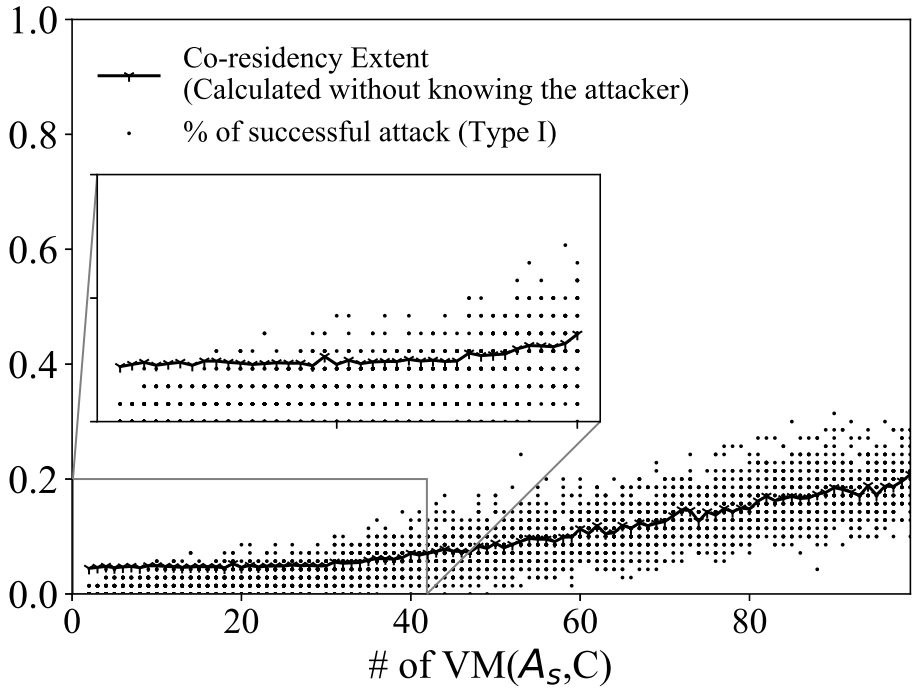


(a)

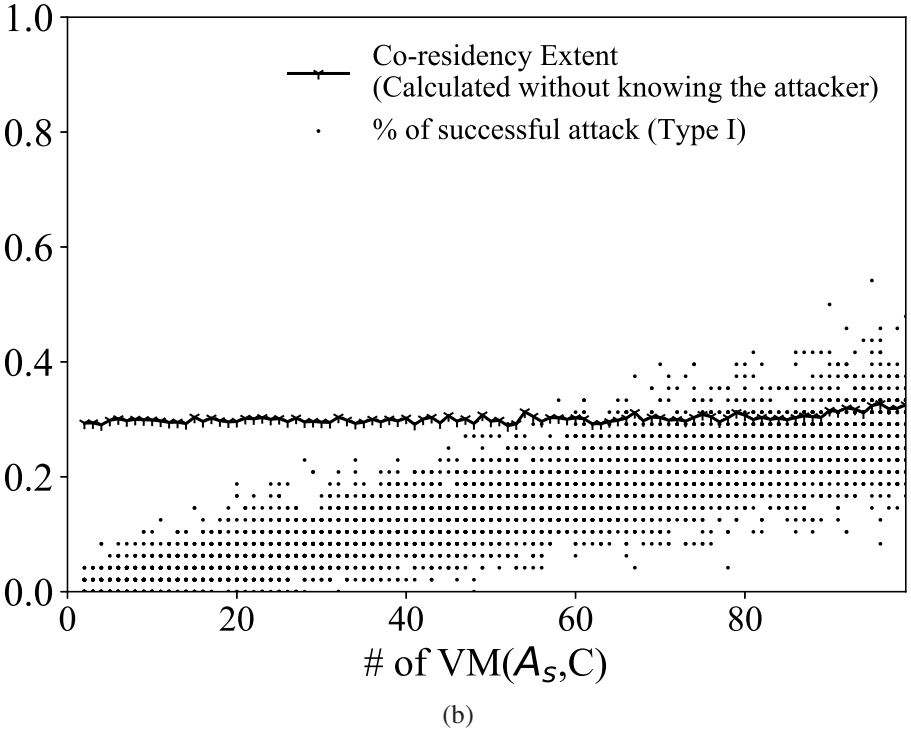


(b)

Figure 6.8: Comparing co-residency extent with the percentage of successful type I attacks under most-VM placement policy, in the two sets of configurations (a) low frequency usage hosts, and (b) high frequency usage hosts



(a)



(b)

Figure 6.9: Comparing co-residency extent with the percentage of successful type I attacks under least-VM placement policy, in the two sets of configurations (a) low frequency usage hosts, and (b) high frequency usage hosts

metric reaches higher values in the data center with the second configuration (as shown in Figure 6.8b). This is mainly due to the nature of the placement policy which tends to stack VMs together, and to the large number of VMs each host is able to accommodate. Second, under both host configurations, attacker could achieve higher co-residency with targeted victims with a tuned launch strategy. Third, co-residency extent provides the security evaluation base-line for a data center. When the number of attacker's launched VMs is low ($VM(A_s,C) < 5$ in Figure 6.8a, and $VM(A_s,C) < 30$ in Figure 6.8b), the metric's values exhibit a plateau trend, which represents the normal co-residency threat level in the data center, as the attacker could not achieve enough co-residency with the victim. Considering such configuration, our metric can be used by the CSP to profile his data center under normal tenants' behavior, and to provide end users with more transparency according to the standard co-residency threat level. Fourth, when the number of attacker's launched VMs goes beyond 5 in the first configuration (shown in Figure 6.8a), and beyond 30 in the second configuration (shown in Figure 6.8b), the co-residency extent follows the percentage of successful attacks as the attacker start achieving a higher degree of co-residency with respect to the victim compared to other tenants in the data center. Note that the average number of VMs per tenant for the first configuration is 10, while it is equal to 49 in the second configuration.

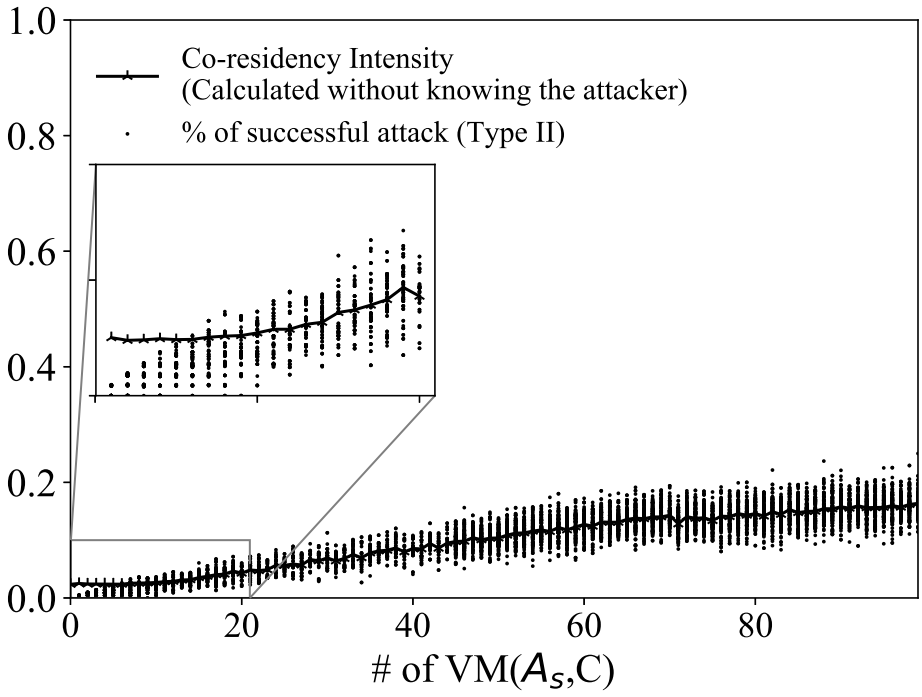
Based on this, we can conclude that our metric can be used both to profile the normal co-residency threat level for a given data center, and to capture the abnormal increase in the degree of co-residencies, which might be due to attackers' malicious behavior in launching their VMs.

Results and Implications for Co-residency Extent in Least-VM Placement Policy. Figure 6.9 reports the co-residency extent metric compared to the percentage of successful attacks under the least VM placement policy, both for a data center with low frequency usage hosts (Figure 6.9a) and for a data center with high frequency usage hosts (Figure 6.9b).

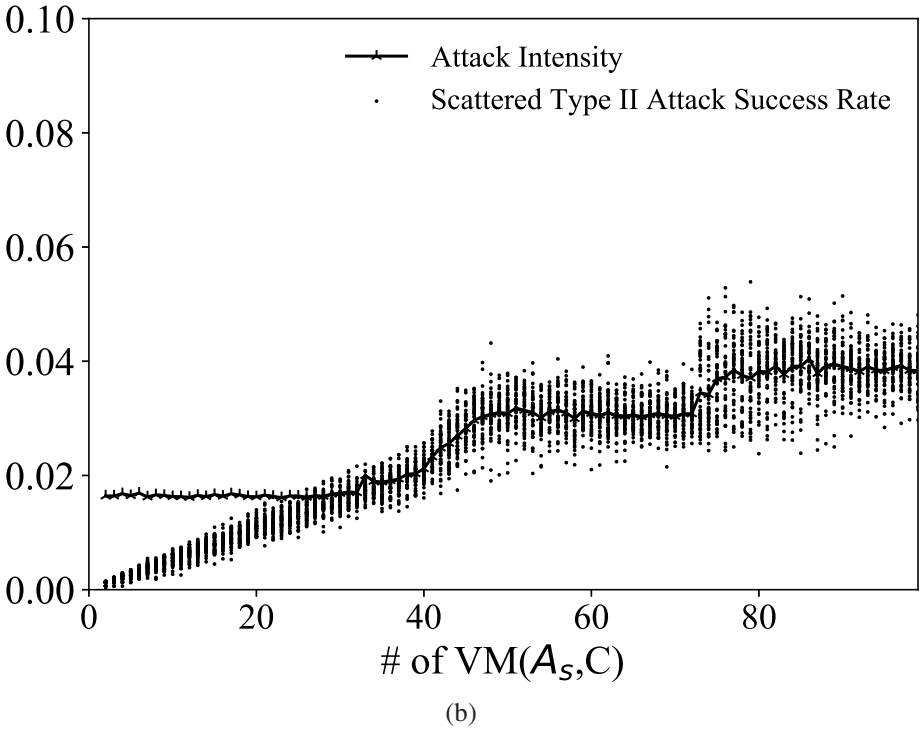
From this figure, we observe that the attacker has a much lower chance to manipulate the least VM placement policy to achieve higher co-residency with the victim, compared to the most VM placement policy in Figure 6.8. Furthermore, in the configuration with low frequent used hosts (Figure 6.9a), attacker could achieve higher co-residency by increasing the number of VMs. However, under the second configuration in Figure 6.9b, achieving enough co-residency becomes a more challenging task for the attacker. In both cases, our metric captures the worst-case scenario inside the data center, which indicates the highest co-residency threat level. Additionally, by observing the simulation results from Figure 6.8 (most VM placement policy) and Figure 6.9 (least VM placement policy), we can conclude that our metric could serve as a reasonable means to evaluate the co-residency threat level related to different VM placement policies.

The objective of the second set of simulations is to study the correlation between the co-residency intensity and the percentage of success for type II attacks under the most and least VM placement policies. Since type II attacks (e.g., power attack and bandwidth saturation attack) mostly rely on placing a large number of attacker's VMs inside the same hosts with the victim, we defined the percentage of success as the percentage of attacker's resources co-residing with victim's VMs.

Results and Implications for Co-residency Intensity. Figure 6.10 (resp. Figure 6.11) reports the co-residency intensity values and the percentage of successful attacks under the most (resp. least) VM placement policy, both for a data center with low frequency usage hosts, and for a data center with high frequency usage hosts. We can observer that, overall, the general trend of co-residency intensity increases at a slower pace than the co-residency extent under the same configurations. This is because intensity relies on the total number of attacker's VMs that achieved co-residency with the victim within the same hosts, which makes the attacks more difficult to achieve. Similarly to the co-residency extent, the metric reaches higher values under the most VM placement policy and for the data center

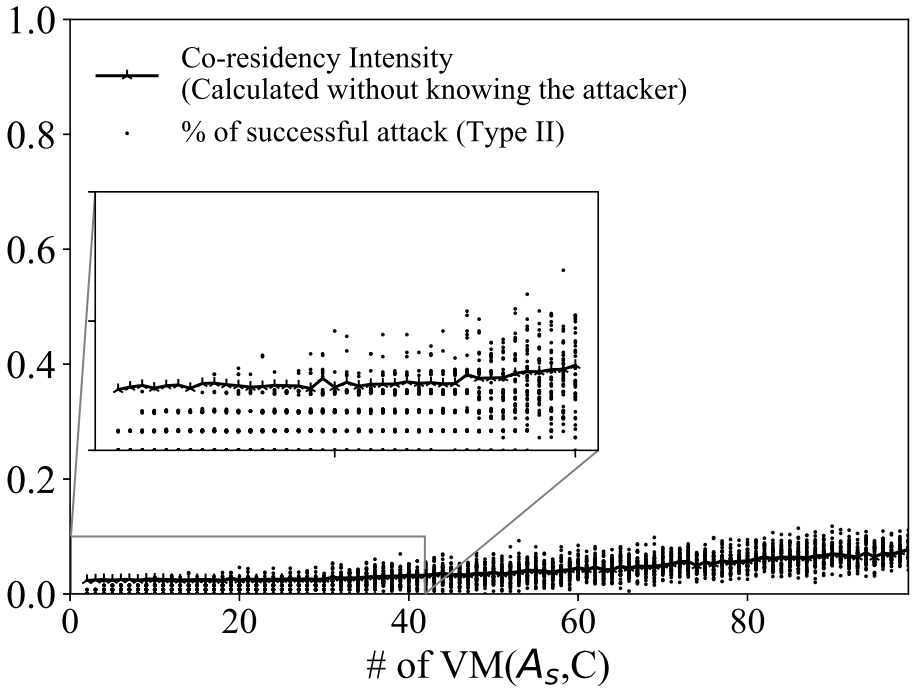


(a)

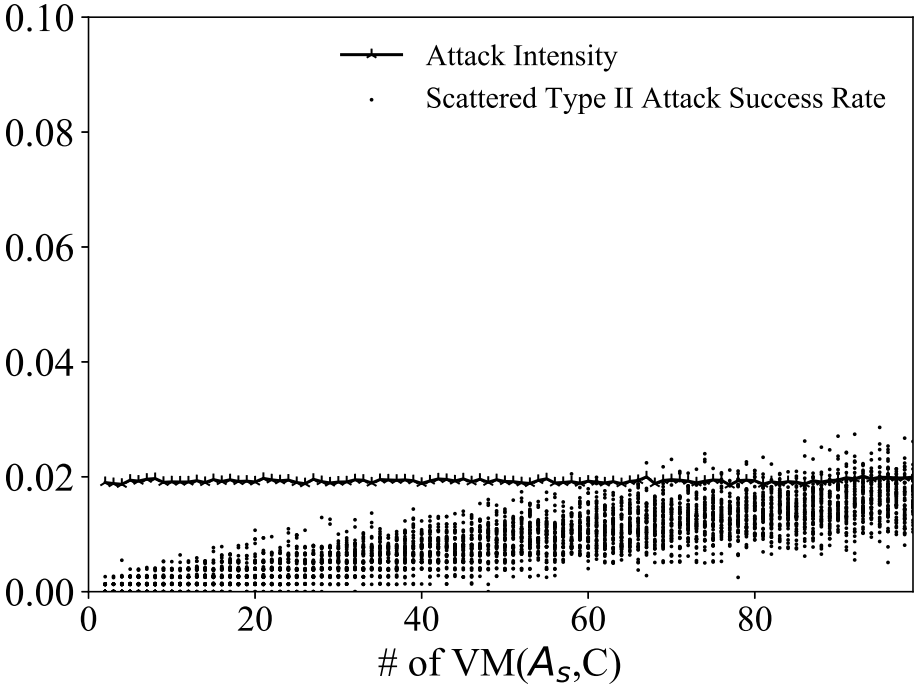


(b)

Figure 6.10: Comparing co-residency intensity with the percentage of successful type II attacks under most-VM placement policy, in the two sets of configurations (a) low frequency usage hosts (b) high frequency usage hosts



(a)



(b)

Figure 6.11: Comparing co-residency intensity with the percentage of successful type II attacks under least-VM placement policy, in the two sets of configurations(a) low frequency usage hosts (b) High frequency usage hosts

configuration with high frequency usage hosts, as illustrated in Figure 6.10b.

The third set of simulation compares the multi-tenancy attack surface, which indicates the overall co-residency threat level and does not correlate with any type of attacks, with the two metrics evaluated in previous simulation-sets, under the two VM-placement policies, while varying the number of victim's VMs. In Figure 6.12a, we omit the result of co-residency intensity metric since it stays plateau with the increase of the victim's VMs under the most-VM placement policy. In Figure 6.12b, the co-residency extent and the co-residency intensity correspond to the left y-axis , while the multi-tenancy attack surface (MAS) corresponds to the right y-axis. In this set of simulations, the number of victim's VMs increases up to 100, while the number of attacker's VMs is fixed to 50.

Results and Implications. In Figure 6.12, we can observe that the trend for the multi-tenancy attack surface is similar to other metrics. A mixed trend for both metrics could be observed in both Figure 6.12a and Figure 6.12b. Since the co-residency intensity stays plateau in Figure 6.12a, the multi-tenancy attack surface follows the co-residency extent at the beginning then stays plateau after a certain number of victim's VMs. In Figure 6.12b, the multi-tenancy attack surface is closer to the co-residency extent when the number of victim's VMs remains small, while the trend gets closer to the maximum attacker's intensity with the increased number of victim's VMs. This is mainly because when the number of victim's VMs increases, the increase of co-residency extent is faster than co-residency intensity. Later on, when the co-residency extent changes at a slower pace, the multi-tenancy attack surface captures better the change in co-residency intensity.

Figure 6.13a shows the change of metrics with the increased number of attacker's VMs under different log configurations, $VM(A_s,C)$. Here again, we can see that the multi-tenancy attack surface metric maintains strong ability to capture the change of both metrics.

Figure 6.13b reports the variation of our metrics' values for data centers with different sizes. Therein, we increase the number of hosts from 300 up to 3,000. The number of

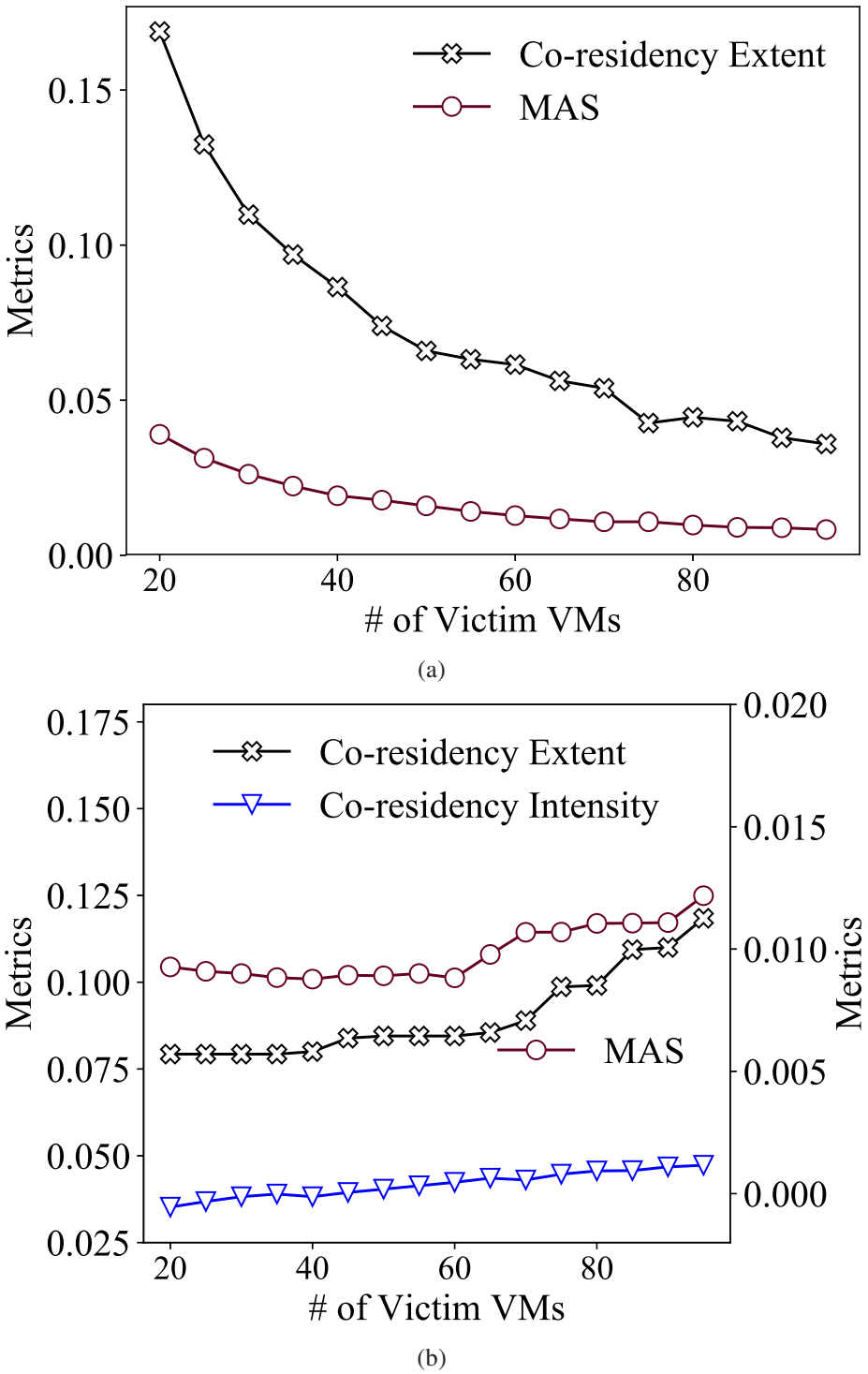
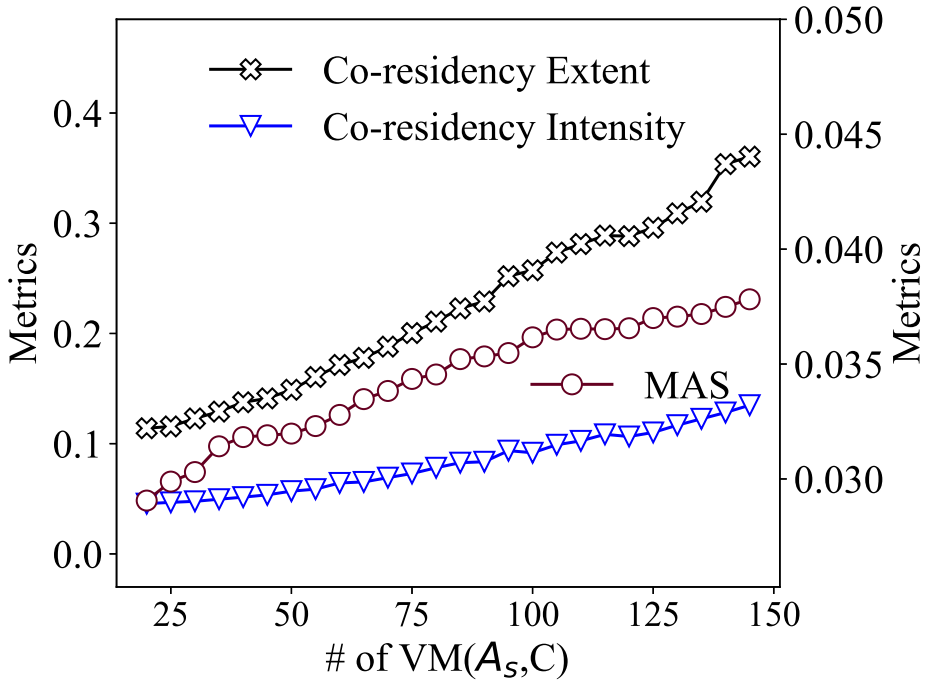
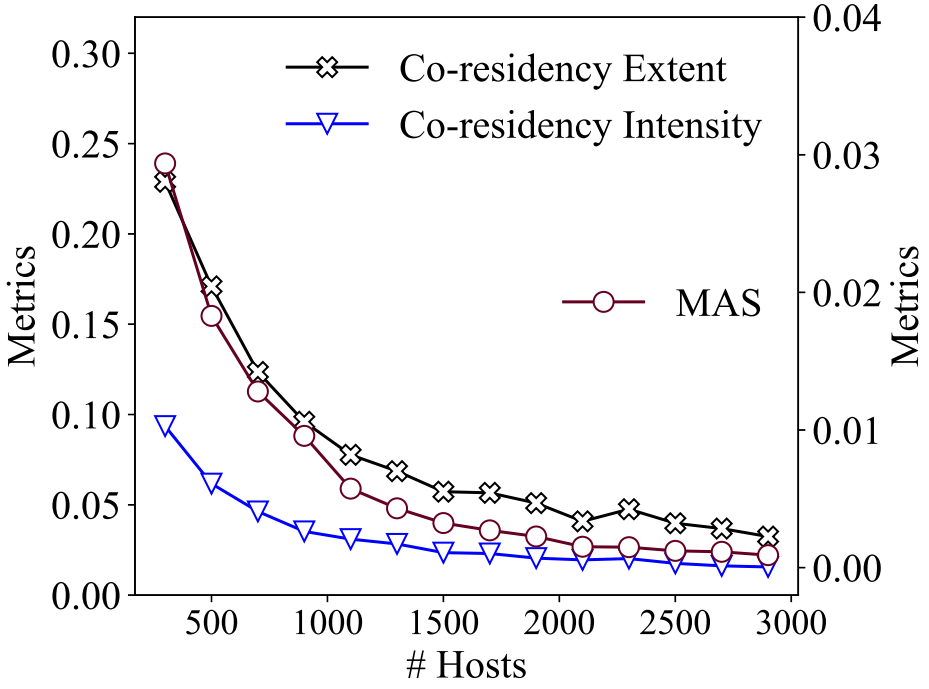


Figure 6.12: Comparing the multi-tenancy attack surface metric with co-residency extent and co-residency intensity while increasing the number of victim's VMs for (a) most-VM, and (b) least-VM placement policy



(a)



(b)

Figure 6.13: Comparing the multi-tenancy attack surface with co-residency extent and co-residency intensity while (a) increasing the number of attacker's VMs, and (b) varying the number of hosts in the data center

victim's and attacker's VMs are set to 100, and the number of legitimate tenants and VMs for the data center changes proportionally to the size of the data center.

Summary of Results. All the metrics demonstrate that the co-residency threat level of a particular tenant decreases with the increasing size of the data center, as our metrics all decrease. We conclude that larger data centers minimize the likelihood of co-residency attacks since attackers have lower chance to co-reside with the victim's resources. The multi-tenancy attack surface metric captures the change of both metrics in this simulation. However, it correlates better with the co-residency extent for small data centers.

6.6 Summary

In this work, we proposed *ProxiMet*, a suite of security metrics to evaluate the proximity between tenants' resources. The main benefit of our approach was to evaluate and mitigate the cloud co-residency threats. To this end, we first extracted a set of common co-residency aspects from the most known co-residency attacks in the cloud along two dimensions, i.e., the extent and the intensity. Then, we designed a set of metrics to evaluate the proximity between tenants' resources along the extracted aspects, as a means to evaluate and mitigate the co-residency threats. To show the effectiveness and usefulness of our metrics, we conducted a case study based on a real cloud, and performed extensive simulations using CloudSim based on well-known cloud placement policies.

Chapter 7

Conclusion

The security implications emanating from virtualization technologies on one side, and the multi-tenancy model on the other side, constitute the main factors that are still holding back prospective cloud customers. Bringing more visibility and transparency to the cloud infrastructure deployments is an important step to overcome this setback. Security compliance auditing and threat evaluation constitute valuable solutions in this respect. However, existing solutions for auditing virtual infrastructures isolation do not consider the complex interdependencies between the cloud stack layers (e.g., the infrastructure management and the implementation layers), which may result in subtle isolation breaches going unnoticed. Furthermore, systematic ways and effective metrics for evaluating cloud threats from tenants' perspective are largely missing in the literature.

This thesis tackled the aforementioned limitations by proposing solutions for cloud security compliance auditing and threat evaluation. First, we proposed an automated audit framework based on formal methods for verifying the cloud infrastructure configuration correctness from the structural point of view. Then, we applied the proposed framework for verifying cross-layer virtual network isolation, one of the most important security properties to cloud customers. Furthermore, we integrated our auditing system into OpenStack, and presented our experimental results on assessing several properties related to virtual

network isolation.

In the context of threat evaluation, based on our analysis for multiple cross-tenant attacks, we proposed suites of security metrics to quantify the proximity between tenants' virtual resources inside the cloud. The cloud provider may apply those metrics to evaluate and mitigate multi-tenancy threats in cloud deployments. To demonstrate the effectiveness of our metrics and show their usefulness, we conducted case studies based on both real and synthetic cloud data. We further performed extensive simulations on CloudSim. Our results show that our metrics effectively capture the threat-level related to the multi-tenancy situation in the cloud, which paves the way for the design of effective mitigation solutions to reduce the side-effect of cloud resource sharing.

As a future direction, we intend to leverage our auditing framework for continuous compliance checking. This will be achieved by monitoring various events, and triggering the verification process whenever a security property is affected by the changes. We also intend to extend our solution to Network Function Virtualization (NFV) environments, where physical security appliances are replaced by their virtual counterparts, which provides the cloud with even more flexibility, allowing the dynamic definition and implementation of complex policies. This makes security breaches easier to happen and emphasizes the need for security compliance verification.

As for threat-level evaluation, we intend to study multi-tenancy attacks taking advantage from shared storage and propose a storage distance accordingly. We also plan to propose cloud management strategies to enforce distances as a means to control the multi-tenancy risk. Another future direction consists of investigating the usability of our metrics for the runtime-detection of different types of co-residency attacks through monitoring. This would enable to capture attacks in their early stages to avoid large scale damage. Another interesting direction consists of devising new mechanisms to empower tenants with the capabilities to verify by themselves the risk related to the actual cloud deployment of

their virtual infrastructures without breaching other tenants' privacy. Finally, the approach we propose for deriving security metrics from potential attacks could be extended for the design of a universal framework that can be used not only for evaluating the security posture of tenants' virtual infrastructures inside cloud/NFV environments, but also to predict attacks when combined with learning mechanisms and monitoring techniques.

Bibliography

[1] datacenterknowledge. Survey: One-third of cloud users' clouds are private, heavily OpenStack, 2015. Available at: `http://www.datacenterknowledge.com`.

[2] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862 – 876, 2010.

[3] ISO. org. Iso/iec 11889-1:2009, 2013.

[4] Institute of Electrical and Electronics Engineers. Ieee 802.1q- 2005. 802.1q - virtual bridged local area networks, 2005.

[5] Karen, Scarfone and Murugiah, Souppaya and Paul, Hoffman. Guide to Security for Full Virtualization Technologies, 2011.

[6] Kui Ren, Cong Wang, and Qian Wang. Security challenges for the public cloud. *IEEE Internet Computing*, 16(1):69–73, January 2012.

[7] Cloud Security Alliance. Cloud control matrix CCM v3.0.1, 2014. Available at:`https://cloudsecurityalliance.org/research/ccm/`.

[8] ISO Std IEC. ISO 27017. *Information technology- Security techniques (DRAFT)*, 2012.

[9] Hongyi Zeng, Shidong Zhang, Fei Ye, Vimalkumar Jeyakumar, Mickey Ju, Junda Liu, Nick McKeown, and Amin Vahdat. Libra: Divide and conquer to verify forwarding tables in huge networks. In *(NSDI 14). Seattle, WA: USENIX Association*, pages 87–99, Seattle, WA, 2014. USENIX Association.

[10] Sören Bleikertz. Automated security analysis of infrastructure clouds. Master's thesis, Technical University of Denmark and Norwegian University of Science and Technology, 2010.

[11] Amazon. Amazon virtual private cloud, 2017. Available at: `https://aws.amazon.com/vpc`.

[12] Google. Google compute engine subnetworks beta, 2017. Available at: `https://cloud.google.com`.

[13] Microsoft. Microsoft Azure virtual network, 2016. Available at: `https://azure.microsoft.com`.

[14] Alan Shieh, Srikanth Kandula, Albert Greenberg, and Changhoon Kim. Seawall: Performance isolation for cloud datacenter networks. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 1–1, Berkeley, CA, USA, 2010. USENIX Association.

[15] Liu Fangfei, Yarom Yuval, Ge Qian, Heiser Gernot, and Lee Ruby B. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*, pages 605–622, May 2015.

[16] Data Center knowledge. Survey: One-third of cloud users' clouds are private, heavily open- stack. Available at: `http://www.datacenterknowledge.com/archives/2015/01/30/survey-half-of-private-clouds-are-openstack-clouds`.

[17] Cloud Security Alliance. Cloud computing top threats in 2016, Feb 2016.

[18] Valentin Del Piccolo, Ahmed Amamou, Kamel Haddadou, and Guy Pujolle. A survey of network isolation solutions for multi-tenant data centers. *IEEE Communications Surveys Tutorials*, PP(99):1–1, 2016.

[19] Chiang Ron C., Rajasekaran Sundaresan, Zhang Nan, and Huang H. Howie. Swiper: Exploiting virtual machine vulnerability in third-party clouds with competition for i/o resources. *IEEE Transactions on Parallel and Distributed Systems*, 26(6):1732–1742, June 2015.

[20] Peter, Mell and Timothy, Grance. The NIST definition of Cloud Computing, 2011.

[21] Open vSwitch. Open vswitch, 2016. Available at: `http://openvswitch.org/`.

[22] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA, May 2015. USENIX Association.

[23] Alan T. Litchfield and Abid Shahzad. Virtualization technology: Cross-vm cache side channel attacks make it vulnerable. *CoRR*, abs/1606.01356, 2016.

[24] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 19–35, Austin, TX, 2016. USENIX Association.

[25] Zhang Yinqian, Juels Ari, Oprea Alina, and Reiter Michael K. Homealone: Co-residency detection in the cloud via side-channel analysis. In *2011 IEEE Symposium on Security and Privacy*, pages 313–328, May 2011.

[26] CSA. Best practices for mitigating risks in virtualized environments, April 2015.

[27] Paul G Dorey. Multi-tenancy security risks-customer expectations for leading cloud service providers - an architectural approach. Technical report, CSO Confidential, 2014.

[28] Zhang Xu, Haining Wang, and Zhenyu Wu. A measurement study on co-residence threat inside the cloud. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 929–944, Washington, D.C., 2015. USENIX Association.

[29] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. A placement vulnerability study in multi-tenant public clouds. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 913–928, Washington, D.C., 2015. USENIX Association.

[30] Suryadipta Majumdar, Yosr Jarraya, Taous Madi, Amir Alimohammadifar, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. *Proactive Verification of Security Compliance for Clouds Through Pre-computation: Application to OpenStack*, pages 47–66. Springer International Publishing, Cham, 2016.

[31] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. *ACM SIGCOMM Computer Communication Review*, 41(4):290–301, 2011.

[32] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *Presented as part of the 9th USENIX Symposium*

on Networked Systems Design and Implementation (NSDI 12), pages 113–126, San Jose, CA, 2012. USENIX.

[33] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 15–27, Lombard, IL, 2013. USENIX.

[34] Peyman Kazemian, Michael Chan, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In *NSDI*, pages 99–111, Lombard, IL, 2013. USENIX.

[35] Sören Bleikertz, Thomas Gro, Matthias Schunter, and Konrad Eriksson. Automated information flow analysis of virtualized infrastructures. In Vijay Atluri and Claudia Daz, editors, *ESORICS*, volume 6879 of *Lecture Notes in Computer Science*, pages 392–415, Berlin, Heidelberg, 2011. Springer.

[36] Sören Bleikertz, Carsten Vogel, and Thomas Groß. Cloud radar: Near real-time detection of security failures in dynamic virtualized infrastructures. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 26–35, New York, NY, USA, 2014. ACM.

[37] Yang Xu, Yong Liu, Rahul Singh, and Shu Tao. Identifying SDN State Inconsistency in OpenStack. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 11:1–11:7, New York, NY, USA, 2015. ACM.

[38] OpenStack. Policy as a service ("congress"), 2014. Available at:`http://wiki.openstack.org/wiki/Congress`.

[39] Suryadipta Majumdar, Taous Madi, Yushun Wang, Yosr Jarraya, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. Security compliance auditing of identity and access management in the cloud: Application to openstack. In *IEEE CloudCom*, pages 58–65, Vancouver, Canada, 2015. IEEE.

[40] S. Majumdar, T. Madi, Y. Wang, Y. Jarraya, M. Pourzandi, L. Wang, and M. Debbabi. User-level runtime security auditing for the cloud. *IEEE Transactions on Information Forensics and Security*, 13(5):1185–1199, May 2018.

[41] Taous Madi, Suryadipta Majumdar, Yushun Wang, Yosr Jarraya, Makan Pourzandi, and Lingyu Wang. Auditing security compliance of the virtualized infrastructure in the cloud: Application to openstack. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY '16, pages 195–206, New York, NY, USA, 2016. ACM.

[42] Taous Madi, Yosr Jarraya, Amir Alimohammadifar, Suryadipta Majumdar, Yushun Wang, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. Isotop: Auditing virtual networks isolation across cloud layers in openstack. *ACM Trans. Priv. Secur.*, 22(1):1:1–1:35, October 2018.

[43] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: Detecting security attacks in software-defined networks. In *NDSS Symposium*, San Diego, California, 2015. Internet Society.

[44] Hongkun Yang and Simon S Lam. Real-time verification of network properties using atomic predicates. In *ICNP*, pages 1–11, Goettingen, Germany, Oct 2013. IEEE.

[45] Yogesh Mundada, Anirudh Ramachandran, and Nick Feamster. Silverline: Data and network isolation for cloud services. In *Proceedings of the 3rd USENIX Conference*

on Hot Topics in Cloud Computing, HotCloud'11, pages 13–13, Berkeley, CA, USA, 2011. USENIX Association.

[46] Thibaut Probst, Eric Alata, Mohamed Kaâniche, and Vincent Nicomette. An approach for the automated analysis of network access controls in cloud computing infrastructures. In *Network and System Security*, pages 1–14. Springer, Xi?an, China, 2014.

[47] Sören Bleikertz, Thomas Gross, M. Schunter, and K. Eriksson. Automating security audits of heterogeneous virtual infrastructures. Technical Report RZ3786, IBM, 2010.

[48] Sören Bleikertz, Thomas Groß, and Sebastian Mödersheim. Automated verification of virtualized infrastructures. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 47–58, New York, NY, USA, 2011. ACM.

[49] Frank Doelitzscher, Christoph Reich, Martin Knahl, Alexander Passfall, and Nathan Clarke. An agent based business aware incident detection system for cloud environments. *Journal of Cloud Computing*, 1(1):9, 2012.

[50] Yong Xiang, Hu Li, Sen Wang, Charley Peter Chen, and Wei Xu. Debugging openstack problems using a state graph approach. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '16, pages 13:1–13:8, New York, NY, USA, 2016. ACM.

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

[52] Sören Bleikertz, Carsten Vogel, Thomas Groß, and Sebastian Mödersheim. Proactive security analysis of changes in virtualized infrastructures. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, pages 51–60, New York, NY, USA, 2015. ACM.

[53] H. Baek, A. Srivastava, and J. V. d. Merwe. Cloudsight: A tenant-oriented transparency framework for cross-layer cloud troubleshooting. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 268–273, May 2017.

[54] U. M. Ismail, S. Islam, and H. Mouratidis. Cloud security audit for migration and continuous monitoring. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 1081–1087, Aug 2015.

[55] Carlos Cotrini, Thilo Weghorn, David Basin, and Manuel Clavel. Analyzing first-order role based access control. In *2015 IEEE 28th Computer Security Foundations Symposium*, pages 3–17, Verona, Italy, July 2015. IEEE.

[56] Perry Alexander, Lee Pike, Peter Loscocco, and George Coker. Model checking distributed mandatory access control policies. *ACM Trans. Inf. Syst. Secur.*, 18(2):6:1–6:25, July 2015.

[57] Jesus Luna, Hamza Ghani, Tsvetoslava Vateva, and Neeraj Suri. Quantitative assessment of cloud security level agreements: A case study. *Proc. of Security and Cryptography*, pages 64–73, 2012.

[58] Jesus Luna Garcia, Robert Langenberg, and Neeraj Suri. Benchmarking cloud security level agreements using quantitative policy trees. In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, pages 103–112. ACM, 2012.

[59] Ahmed Taha, Ruben Trapero, Jesus Luna, and Neeraj Suri. Ahp-based quantitative approach for assessing and comparing cloud security. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, pages 284–291. IEEE, 2014.

[60] Jesus Luna, Ahmed Taha, Ruben Trapero, and Neeraj Suri. Quantitative reasoning about cloud security using service level agreements. *IEEE Transactions on Cloud Computing*, PP(99), 2015.

[61] Yi Han, Jeffrey Chan, Tansu Alpcan, and Christopher Leckie. Virtual machine allocation policies against co-resident attacks in cloud computing. In *IEEE International Conference on Communications (ICC'14)*, pages 786–792, 2014.

[62] Yi Han, Jeffrey Chan, Tansu Alpcan, and Christopher Leckie. Using virtual machine allocation policies to defend against co-resident attacks in cloud computing. *IEEE Trans. Dependable Sec. Comput.*, 14(1):95–108, 2017.

[63] Jin Han, Wanyu Zang, Songqing Chen, and Meng Yu. Reducing security risks of clouds through virtual machine placement. In Giovanni Livraga and Sencun Zhu, editors, *Data and Applications Security and Privacy XXXI*, pages 275–292, Cham, 2017. Springer International Publishing.

[64] Soo-Jin Moon, Vyas Sekar, and Michael K. Reiter. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *CCS '15*, pages 1595–1606, New York, NY, USA, 2015. ACM.

[65] M. Azab and M. Eltoweissy. Migrate: Towards a lightweight moving-target defense against cloud side-channels. In *SPW'16*, pages 96–103, May 2016.

[66] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds.

In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[67] Cloud Security Alliance. Top ten big data security and privacy challenges, 2012.

[68] Opendaylight. The OpenDaylight platform, 2015. Available at: `https://www.opendaylight.org/`.

[69] OpenStack. Nova network configuration allows guest vms to connect to host services, 2015. Available at: `https://wiki.openstack.org/wiki/OSSN/OSSN-0018`.

[70] Shakeel Butt, H. Andrés Lagar-Cavilla, Abhinav Srivastava, and Vinod Ganapathy. Self-service cloud computing. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 253–264, New York, NY, USA, 2012. ACM.

[71] Mihir Bellare and Bennet Yee. Forward integrity for secure audit logs. Technical report, Citeseer, 1997.

[72] TechNet. Cloud services foundation reference architecture- reference model, 2013.

[73] ISO Std IEC. ISO 27002:2005, 2005.

[74] Cloud Security Alliance. The notorious nine cloud computing top threats in 2013, February 2013.

[75] OpenStack. Ossa-2014-008: Routers can be cross plugged by other tenants, 2014. Available at: https://security.openstack.org/ossa/OSSA-2014-008.html.

[76] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. SP '11, pages 313–328, Washington, DC, USA, 2011. IEEE Computer Society.

[77] Stephen Gutz, Alec Story, Cole Schlesinger, and Nate Foster. Splendid isolation: A slice abstraction for software-defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 79–84, New York, NY, USA, 2012. ACM.

[78] Diego Perez-Botero, Jakub Szefer, and Ruby B. Lee. Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the 2013 International Workshop on Security in Cloud Computing*, Cloud Computing '13, pages 3–10, New York, NY, USA, 2013. ACM.

[79] IBM Corporation. Ibm point of view: Security and cloud computing, 2009.

[80] Frank Hans-Ulrich Doelitzscher. *Security Audit Compliance For Cloud Computing*. PhD thesis, Plymouth University, February 2014.

[81] NIST, SP. NIST SP 800-53, 2003.

[82] The European Network, , and Information Security Agency. Cloud computing benefits, risks and recommendations for information security, December 2012.

[83] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing v 3.0, 2011.

[84] Naoyuki Tamura and Mutsunori Banbara. Sugar: A CSP to SAT translator based on order encoding. *Proceedings of the Second International CSP Solver Competition*, pages 65–69, 2008.

[85] Mordechai Ben-Ari. *Mathematical logic for computer science*. Springer Science & Business Media, London, 2012.

[86] Shuyuan Zhang and Sharad Malik. Sat based verification of network data planes. In Dang Van Hung and Mizuhito Ogawa, editors, *Automated Technology for Verification and Analysis*, volume 8172 of *Lecture Notes in Computer Science*, pages 496–505. Springer International Publishing, Cham, 2013.

[87] Kui Ren, Cong Wang, and Qian Wang. Security challenges for the public cloud. *IEEE Internet Computing*, 16(1):69–73, Jan 2012.

[88] Penny Pritzker and Patrick D. Gallagher. Nist cloud computing standards roadmap. Technical report, NIST, Gaithersburg, MD, United States, 2013. NIST Special Publication 500-291.

[89] Cisco Systems Sean Convery. Hacking layer 2: Fun with ethernet switches, 2002. BlackHat Briefings.

[90] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon, Martin Casado, and Scott Shenker. Extending networking into the virtualization layer. In *HotNets*, YorkCity,NY, October 2009. ACM.

[91] OpenStack. Ossa-2014-008: Routers can be cross plugged by other tenants, 2014. Available at: https://security.openstack.org/ossa/OSSA-2014-008.html.

[92] Yushun Wang, Taous Madi, Suryadipta Majumdar, Yosr Jarraya, Amir Alimohammadifar, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. Tenantguard: Scalable runtime verification of cloud-wide vm-level network isolation. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[93] Crandall et al. Virtual networking management white paper. Technical report, DMTF, 02 2012. DMTF Draft White Paper.

[94] VMware. vcloud director, 2017. Available at: https://www.vmware.com/fr/products/vcloud-director.html.

[95] H. Moraes, M. A. M. Vieira, . Cunha, and D. Guedes. Efficient virtual network isolation in multi-tenant data centers on commodity ethernet switches. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 100–108, Vienna, Austria, May 2016. IEEE.

[96] ONF. Openflow switch specification, April 2013. Available at:http://www.gesetze-im-internet.de/englisch_bdsg.

[97] Naoyuki Tamura. Syntax of sugar csp description, 2010. Available at: http://bach.istc.kobe-u.ac.jp/sugar/current/docs/syntax.html.

[98] Ruben Martins, Vasco Manquinho, and Inês Lynce. An overview of parallel sat solving. *Constraints*, 17(3):304–347, Jul 2012.

[99] Hewlett Packard Enterprise. Hpe helion eucalyptus, 2017. Available at: http://www8.hp.com/us/en/cloud/helion-eucalyptus.html.

[100] Midokura. Run midonet at scale, 2017. Available at: http://www.midokura.com/midonet/.

[101] CSA. The notorious nine cloud computing top threats in 2013, 2013.

[102] Zhang Xu, Haining Wang, Zichen Xu, and Xiaorui Wang. Power attack: An increasing threat to data centers. In *Proceedings of 2014 Annual Network and Distributed System Security Symposium (NDSS'14)*, 2014.

[103] S. Al-Haj, E. Al-Shaer, and H. V. Ramasamy. Security-aware resource allocation in clouds. In *2013 IEEE International Conference on Services Computing*, pages 400–407, June 2013.

[104] Tianwei Zhang and Ruby B. Lee. Host-based dos attacks and defense in the cloud. In *Proceedings of the Hardware and Architectural Support for Security and Privacy*, HASP '17, pages 3:1–3:8, New York, NY, USA, 2017. ACM.

[105] Y. Park, S. Y. Chang, and L. M. Krishnamurthy. Watermarking for detecting freeloader misbehavior in software-defined networks. In *2016 International Conference on Computing, Networking and Communications (ICNC)*, pages 1–6, Feb 2016.

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

[107] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 309–322, Berkeley, CA, USA, 2011. USENIX Association.

[108] Md Hasanul Ferdaus, Manzur Murshed, Rodrigo N. Calheiros, and Rajkumar Buyya. *Network-Aware Virtual Machine Placement and Migration in Cloud Data Centers*, pages 31–42. IGI Global, Hershey, USA.

[109] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07*, pages 13–23, New York, NY, USA, 2007. ACM.

[110] D. S. Marcon, R. R. Oliveira, M. C. Neves, L. S. Buriol, L. P. Gaspary, and M. P. Barcellos. Trust-based grouping for cloud datacenters: Improving security in shared infrastructures. In *2013 IFIP Networking Conference*, pages 1–9, May 2013.

[111] Openstack. Openstack. https://www.openstack.org/.

[112] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, August 2008.

[113] Mehmet Sinan Inci, Berk Gulmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Seriously, get off my cloud! cross-vm rsa key recovery in a public cloud, 2015.

[114] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M. Swift. Resource-freeing attacks: Improve your cloud performance (at your neighbor's expense). In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 281–292, New York, NY, USA, 2012. ACM.

[115] SANS Institute, InfoSec Reading Room. An introduction to securing a cloud environment, 2012.

[116] Amazon Web Services. Overview of security processes, June 2016.

[117] Y. Han, T. Alpcan, J. Chan, C. Leckie, and B. I. P. Rubinstein. A game theoretical approach to defend against co-resident attacks in cloud computing: Preventing co-residence using semi-supervised learning. *IEEE Transactions on Information Forensics and Security*, 11(3):556–570, March 2016.

[118] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 29–40, New York, NY, USA, 2011. ACM.

[119] Fangfei Zhou, Manish Goel, Peter Desnoyers, and Ravi Sundaram. Scheduler vulnerabilities and coordinated attacks in cloud computing. *CoRR*, abs/1103.0759, 2011.

[120] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 159–173, Bellevue, WA, 2012. USENIX.

[121] Suaad S. Alarifi and Stephen D. Wolthusen. Robust coordination of cloud-internal denial of service attacks. *2013 International Conference on Cloud and Green Computing*, pages 135–142, 2013.

[122] Yang Liu and Jogesh K. Muppala. A survey of data center network architectures. 2013.

[123] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50, August 2009.

[124] CVE Numbering Authorities. Cve-2015-3456, 2015.

[125] Yossi Azar, Seny Kamara, Ishai Menache, Mariana Raykova, and Bruce Shepard. Co-location-resistant clouds. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, CCSW '14, pages 9–20, New York, NY, USA, 2014. ACM.

[126] Salman A. Baset, Long Wang, and Chunqiang Tang. Towards an understanding of oversubscription in cloud. In *Presented as part of the 2nd USENIX Workshop on Hot*

Topics in Management of Internet, Cloud, and Enterprise Networks and Services, San Jose, CA, 2012. USENIX.

[127] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[128] Yi Han, Jeffrey Chan, and Christopher Leckie. Analysing virtual machine usage in cloud computing. In *IEEE Ninth World Congress on Services, SERVICES'13*, pages 370–377, 2013.