ISOTOP: Auditing Virtual Networks Isolation Across Cloud Layers in OpenStack

TAOUS MADI, CIISE, Concordia University, Canada
YOSR JARRAYA, Ericsson Security Research, Canada
AMIR ALIMOHAMMADIFAR, SURYADIPTA MAJUMDAR, and YUSHUN WANG, CIISE, Concordia University, Canada
MAKAN POURZANDI, Ericsson Security Research, Canada
LINGYU WANG and MOURAD DEBBABI, CIISE, Concordia University, Canada

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-tenants data leakage and denial of service. Particularly, virtual networks isolation failures are among the foremost security concerns in the cloud. To remedy these, automated tools are needed to verify security mechanisms compliance with relevant security policies and standards. However, auditing virtual networks isolation is challenging due to the dynamic and layered nature of the cloud. Particularly, inconsistencies in network isolation mechanisms across cloud stack layers, namely the infrastructure management and the implementation layers, may lead to virtual networks isolation breaches that are undetectable at a single layer. In this paper, we propose an off-line automated framework for auditing consistent isolation between virtual networks in OpenStack-managed cloud spanning over overlay and layer 2 by considering both cloud layers' views. To capture the semantics of the audited data and its relation to consistent isolation requirement, we devise a multi-layered model for data related to each cloud-stack layer's view. 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 reasonable time.

CCS Concepts: • **Security and privacy** → *Distributed systems security*;

Additional Key Words and Phrases: Cloud, security, compliance verification, network isolation, consistency, openStack, virtual infrastructure

ACM Reference Format:
Taous Madi, Yosr Jarraya, Amir Alimohammadifar, Suryadipta Majumdar, Yushun Wang, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. 2018. ISOTOP: Auditing Virtual Networks Isolation Across Cloud Layers in OpenStack. *ACM Trans. Web* 21, 4, Article 39 (August 2018), 33 pages. https://doi.org/0000001.0000001

Authors' addresses: Taous Madi, CIISE, Concordia University, Montreal, QC, Canada, t_madi@concordia.encs.ca; Yosr Jarraya, Ericsson Security Research, Montreal, QC, Canada, yosr.jarraya@ericsson.com; Amir Alimohammadifar; Suryadipta Majumdar; Yushun Wang, CIISE, Concordia University, Montreal, QC, Canada, {ami_alim,su_majum,yus_wang}@encs.concordia.ca; Makan Pourzandi, Ericsson Security Research, Montreal, QC, Canada, makan.pourzandi@ericsson.com; Lingyu Wang; Mourad Debbabi, CIISE, Concordia University, Montreal, QC, Canada, {wang,debbabi}@encs.concordia.ca.

1559-1131/2018/8-ART39 $15.00
https://doi.org/0000001.0000001

1 INTRODUCTION

Despite the abundant benefits of the cloud, security and privacy concerns are still holding back its widespread adoption [50]. 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 [26, 27] and CCM 3.0.1 [13]). Particularly, network isolation failures are among the foremost security concerns in the cloud [14, 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 [51]. 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. Note that domain-specific terms used in the paper are summarized in the glossary.

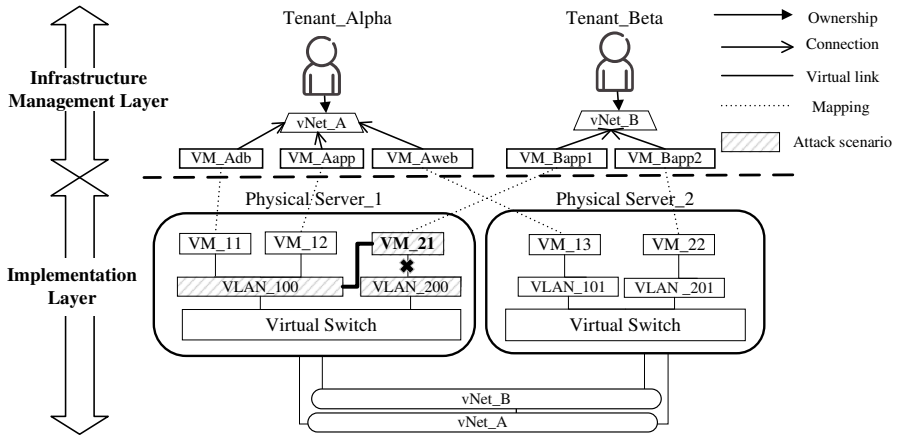

Fig. 1. A Two-Layer View of a Multi-Tenant Virtualized Infrastructure in Cloud: The Infrastructure Management Layer and the Implementation Layer

Motivating Example. Figure 1 illustrates a simplified view of an OpenStack [45] configuration example for virtualized multi-tenant cloud environments. Following a layered architecture [48], 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 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 [27] 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 [46]) 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 [52]. 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 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., [31, 55]) are not designed to handle cross-layer consistent isolation verification. Thus, in this paper, 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[1]. 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:

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

- 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 [53], 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.

The preliminary version of this paper appears in [31]. While the latter focuses on the verification of only the infrastructure management layer, we propose in this paper a different approach that tackles the need of considering complementarity between cloud layers, namely implementation and infrastructure management layers. Thus, we derive a new model capturing network-related entities at each layer, their inter-relationships annotated with cardinality constraints, and cross-layer mapping (Section 2). We propose a new methodology (Section 3) addressing challenges of cross-layer verification and demonstrate how our solution can detect layer 2 virtual network isolation breaches with per-tenant evidences. We also propose a new set of security properties (Section 3.2) related to cross-layer network topology isolation and consistency requirements. Furthermore, we discuss new experimental results.

The remainder of this paper is organized as follows. Section 2 presents a background on network isolation mechanisms, the threat model and our cloud virtualized infrastructure model. Section 3 describes our methodology and the related security properties. Section4 details the integration of our auditing framework into OpenStack. Section 5 experimentally evaluates the performance of our approach. Section 6 discusses the adaptability of our solution to other platforms and possible improvements. Section 7 reviews the related work. Finally, we conclude our paper and provide future directions in Section 8.

2 MODELS

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

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) [2] for providing layer 2 network function to guest VMs at the host level [47].

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,

[2]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.

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 2.1. Figure 2 illustrates a more detailed view of layer 2 virtual networks implementation for the configuration showed in Figure 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 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 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).

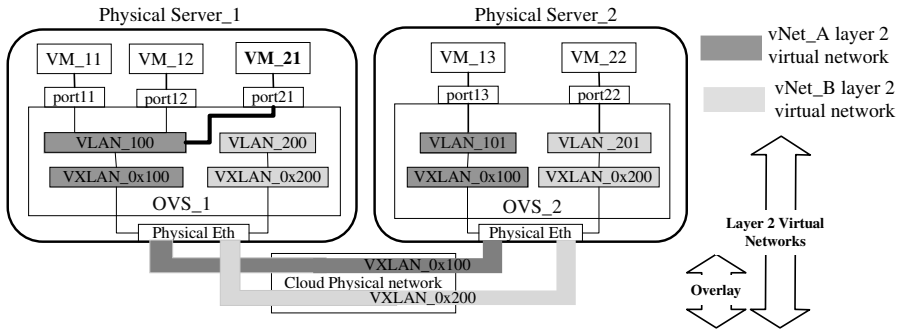

Fig. 2. A Detailed View of the Implementation Layer of Figure 1

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 [42] 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 [51]. 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., [4]).

We focus on layer 2 virtual network in this paper, and our work is complementary to existing solutions at other network layers. We assume that not all tenants trust each other. In certain cloud offerings (e.g., private clouds), a tenant can either require not to share any physical resource with all other tenants, or provide a white (or black) list of trusted (or distrusted) tenants that he is (or not) willing to share resources with. Finally, 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 the paper.

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 [16]. 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) [3], Google Cloud Platform (GCP) [23], Microsoft Azure [36], VMware virtual Cloud Director (vCD) [54] and OpenStack [45]). More details will be provided in Table 7 (Section 6). For the implementation model, we relied on performing intensive tests on OpenStack compute and network nodes, then we supported our understanding by exploring the literature [18, 38]. 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. We first present an example that provides intuitions on the proposed model.

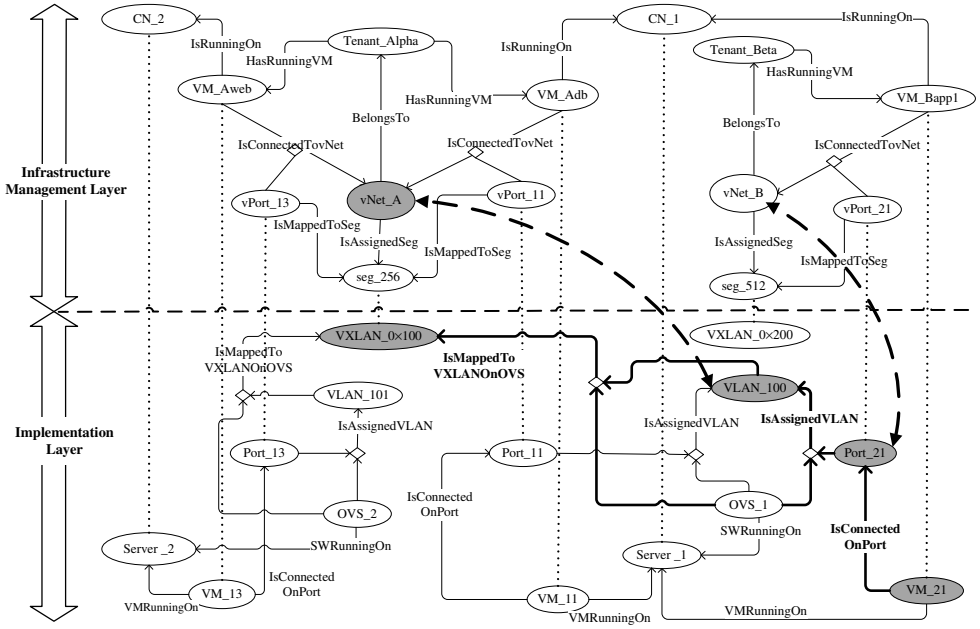

Fig. 3. Subsets of Data and its Relations at the Could Infrastructure Implementation and Management Layers Showing Isolation Violation. At the Implementation Level, VM_21 is Connected on Port_21, that is Assigned VLAN_100 as a Consequence of the Attack. Since VLAN_100 is Mapped to VXLAN 0×100, which 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 2.2. Figure 3 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 1 and Figure 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).

The two-layered model shown in Figure 3 is actually a sub-instance of the model we derived for the cloud infrastructure management and implementation layers that is illustrated in Figure 4. Therein, entities are illustrated using rectangles and arrows represent relationships between those entities. Entities represent data types that are managed by one of the layers. We use cardinality constraints to capture allowed number of data instances for each entity in the context of each relationship.

Infrastructure Management Model. The upper model in Figure 4 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 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 2.3. Ownership is modeled using the BelongsTo relationship in Figure 4 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 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

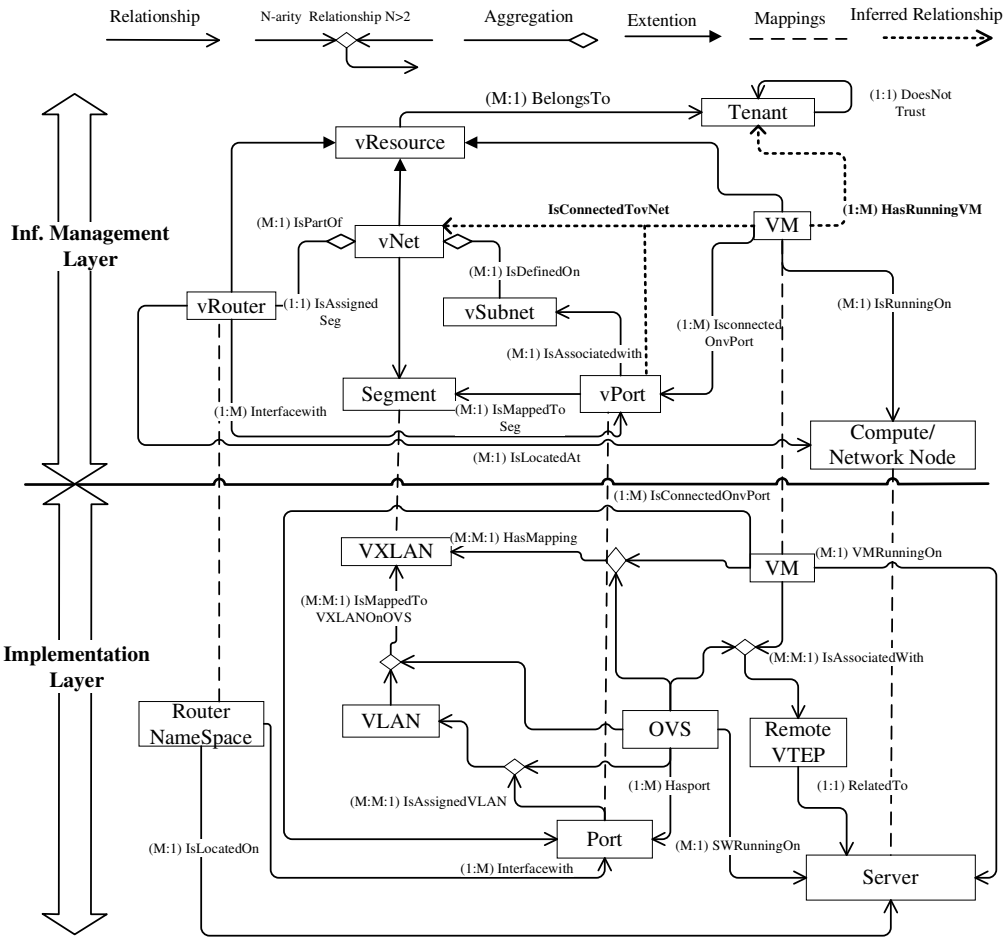

Fig. 4. Two-layered Model for Isolated Multi-Tenant Virtualized Infrastructures in the Cloud: Generic Model for the Infrastructure Management Layer (Upper Model) Mapped into an Implementation-specific Model of the Infrastructure Layer (Lower Model)

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 2.4. At the lower model in Figure 4, 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.

3 METHODOLOGY

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

3.1 Overview

Figure 5 presents an overview of our approach. Our main idea is to use the derived two-layered model (Section 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.

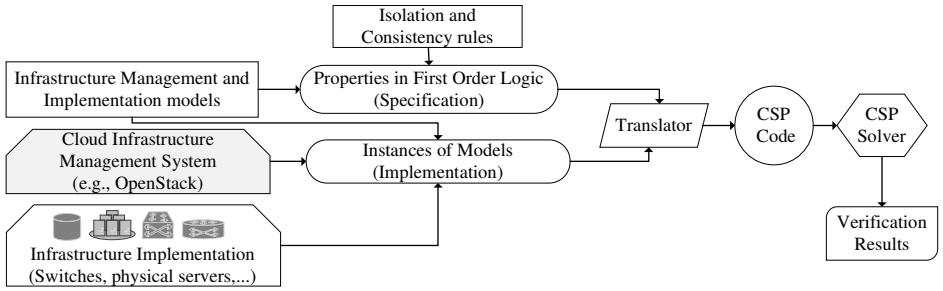

Fig. 5. An Overview of our Verification Approach

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) [5]. 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 [58]. Furthermore, the latter allows to identify the data violating the verified properties as it will be explained in Section 3.3.

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 paper, 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 1 presents an excerpt of the security properties mapped to relevant domains and control classes in security standards, namely CCM [13] (Infrastructure and virtualization security segmentation domain), ISO27017 [27] (Segregation in networks section) and NIST800 [41] (System and communications protection, System and information integrity security controls). These properties are either checked on individual cloud layers (i.e., infrastructure management level or implementation level), or based on information gathered from both layers at the same time. In the following, we provide a brief description for the security properties of interest and discuss examples illustrating how those properties are related to isolation, and how they can be violated.

Physical Isolation (No VM co-residence). Physical isolation [24] aims at preventing side and covert channel attacks, and reducing the risk of attacks staged based on hypervisor and software switches vulnerabilities by hosting VMs in different physical servers. Such attacks might lead to performance degradation, sensitive information leakage, and denial of service.

Example 3.1. (No VM co-residence) Figure 6 consists of two subsets of instances of the infrastructure management model presented in Section 2 focusing on entities Tenant, VM and CN (compute node). 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 its VMs from those of `Tenant_Beta`. However, as illustrated at the right side of Figure 6, `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.

Virtual Resource Isolation (No common ownership). The no common ownership property [31] 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 a vulnerability OSSA-2014-008 [43], reported in the OpenStack Neutron networking service, which allows a tenant to create a virtual port on another tenant's router. An

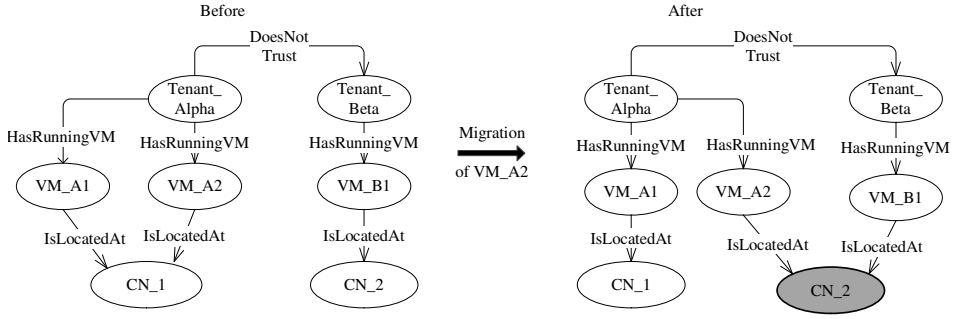

Fig. 6. Subsets of the Infrastructure Management Model Instances Before and After Violation of *No VM Co-residence* Property Illustrating an Example of Data on VM Locations. After Migration, `VM_A2` Becomes Co-resident with `VM_B1` at Compute Node `CN_2`.

instance of our infrastructure management model can capture this violation as illustrated in Figure 7. 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 `Router_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 it is the initiator of the port creation. But since the port is connected to `Router_A1`, the created port would be considered as a common resource for both tenants.

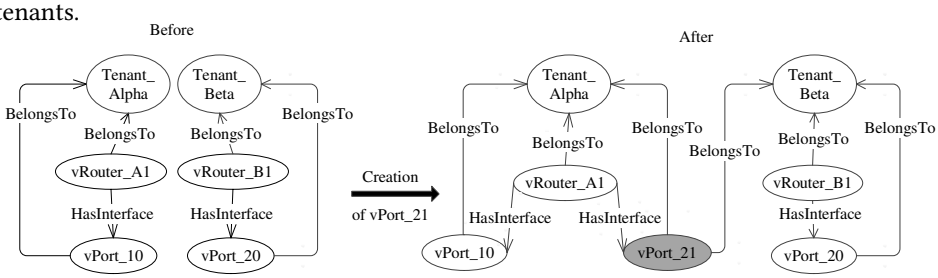

Fig. 7. Subsets of the Infrastructure Management Model Instances Before and After Violation of *No Common Ownership* Property Illustrating an Example of Data on Ports and Routers Ownership. After Creating Port `vPort_21`, the Latter Becomes Owned by Two Tenants.

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. Example 2.2 provided in Section 2 illustrates a topology isolation violation using an instance of our model.

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 3.3. (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.
Physical isolation	•	•	•	No VM co-residence (P1)	VMs of a tenant should not be placed on the same compute node as VMs of a non trusted tenant	×	
Virtual resources isolation	•	•	•	No common ownership (P2)	All tenant-specific resources should belong to a unique tenant	×	
Topology isolation	•	•	•	Mappings unicity Virtual Networks-Segments (P3)	Virtual networks and segments should be mapped one-to-one	×	
				Mappings unicity Ports-Segments (P4)	vPorts should be mapped to unique segments	×	
				Correct association Ports-Virtual Networks (P5)	VMs should be attached tot he virtual networks they are connected to through the right vPorts	×	
				Mapping unicity Ports-VLANs (P6)	Ports should be mapped tounique VLANs		×
				Mapping unicity VLANs-VXLANs (P7)	VLANs and VXLANs should be mapped one-to-one on a given server		×
				Overlay tunnels isolation (P8)	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 (P9)	Consistency between VMs locations at the implementation level and at the management level	×	×
				Ports consistency (P10)	Consistency between vPorts in the implementation level and their counterparts in the management level	×	×
				Virtual links consistency (P11)	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 1. Excerpt of Security Properties

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.

3.3.1 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 3, 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 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.

Relations	Def. at	Evaluate to $True$ if
$BelongsTo(r, t)$	Mgmt.	The resource r is owned by tenant t
$HasRunningVM(t, vm)$	Mgmt.	The tenant t has a running virtual machine vm
$DoesNotTrust(t1, t2)$	Mgmt.	Tenant $t2$ is not trusted by tenant $t1$ which means that $t1$'resources should not share the same hardware with $t2$' instances
$IsRunningOn(vm, cn)$	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 portvp
$HasPort(sw, p)$	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 2. Relations in the Implementation and Infrastructure Management Models Encoded in FOL

Properties	FOL Expressions
No VM co-residence (P1)	$\forall t1, t2 \in TENANT, \forall vm1, vm2 \in VM, \forall cn1, cn2 \in COMPUTEN : HasRunningVM(t1, vm1) \wedge HasRunningVM(t2, vm2) \wedge DoesNotTrust(t1, t2) \wedge IsRunningOn(vm1, cn1) \wedge IsRunningOn(vm2, cn2) \rightarrow \neg(cn1 = cn2)$
No common ownership (P2)	$\forall r \in vResource, \forall t1, t2 \in TENANT: BelongsTo(r, t1) \wedge BelongsTo(r, t2) \rightarrow (t1 = t2)$
Mappings unicity Virtual Networks and Segments (P3)	$\forall vNet1, vNet2 \in vNET, \forall seg1, seg2 \in Segment : [IsAssignedSeg(vNet1, seg1) \wedge IsAssignedSeg(vNet2, seg2) \wedge \neg(vNet1 = vNet2) \rightarrow \neg(seg1 = seg2)] \wedge [IsAssignedSeg(vNet1, seg1) \wedge IsAssignedSeg(vNet2, seg2) \wedge \neg(seg1 = seg2) \rightarrow \neg(vNet1 = vNet2)]$
Mappings unicity Ports-Segments (P4)	$\forall seg1, seg2 \in Segment, \forall vp \in vPORT : IsMappedToSeg(vp, seg1) \wedge IsMappedToSeg(vp, seg2) \rightarrow (seg1 = seg2)$
Correct association Ports-Virtual Networks (P5)	$\forall vm \in VM, \forall vNet \in vNET, \forall seg1, seg2 \in Segment, \forall vp \in vPort : IsConnectedTovNet(vm, vNet, vp) \wedge IsAssignedSeg(vNet, seg1) \wedge IsMappedToSeg(vp, seg2) \rightarrow (seg1 = seg2)$

Table 3. Isolation Properties at the Infrastructure Management Level in FOL

Properties	FOL Expressions
Mapping unicity Ports-VLANs (P6)	$\forall sw \in OVS, \forall p \in Port, \forall vlan1, vlan2 \in VLAN : HasPort(sw, p) \wedge IsAssignedVLAN(sw, p, vlan1) \wedge IsAssignedVLAN(sw, p, vlan2) \rightarrow (vlan1 = vlan2)$
Mapping unicity VLANs-VXLANs (P7)	$\forall vxlan1, vxlan2 \in VXLAN, \forall vlan \in vlan, \forall sw \in OVS, \forall p \in PORT : (IsAssignedVLAN(sw, p, vlan) \wedge IsMappedToVXLANOnOVS(sw, vlan, vxlan1) \wedge IsMappedToVXLANOnOVS(sw, vlan, vxlan2) \rightarrow (vxlan1 = vxlan2)$
Overlay tunnels isolation (P8)	$\forall vm \in VM, \forall sw1, sw2 \in OVS, \forall p \in PORT, \forall vxlan1, vxlan2 \in VXLAN, \forall s1, s2 \in Server, \forall vtep \in RemoteVTEP, \forall vlan \in VLAN : HasPort(sw1, p) \wedge SWRunningOn(sw1, s1) \wedge IsConnectedOnPort(VM, sw1, p) \wedge IsAssignedVLAN(sw1, p, vlan) \wedge IsMappedToVXLANOnOVS(sw1, vlan, vxlan1) \wedge IsAssociatedWith(sw2, vm, vtep) \wedge HasMapping(sw2, vm, vxlan2) \wedge IsRelatedTo(vtep, s2) \rightarrow (s1 = s2) \wedge (vxlan1 = vxlan2)$

Table 4. Isolation Properties at the Implementation Level in FOL

3.3.2 Properties Expressions. Security properties presented in Table 1 can be expressed as FOL predicates over FOL relations defined in Table 2.

Table 3 shows FOL predicates for the isolation properties at the infrastructure management model. Table 4 presents FOL predicates for the isolation properties at the implementation model. Table 5 summarizes the expressions of consistency-related properties.

Properties	FOL Expressions
VM location consistency (P9)	$\forall vm1 \in VM, \forall cn \in COMPUTEN : IsRunningOn((vm1, cn) \rightarrow$ $\exists vm2 \in iVM, \exists s \in SERVER : VMRunningOn(vm2, s) \wedge (vm1 = vm2) \wedge (cn = s)$
Ports consistency (P10)	$\forall vNet \in vNET, \forall seg \in Segment, \forall vp \in vPORT : IsAssignedSeg(vNet, seg)$ $\wedge IsMappedToSeg(vp, seg) \rightarrow$ $[\exists sw \in OVS, \exists vxlan \in VXLAN, \exists vlan \in VLAN, \exists p \in PORT : IsAssignedVLAN(sw, p, vlan)$ $\wedge IsMappedToVXLANOnOVS(sw, vlan, vxlan) \wedge (seg = vxlan)(vp = p)]$
Virtual links consistency (P11)	$\forall vm1 \in iVM, \forall vxlan \in VXLAN, \forall sw \in OVS, \forall vlan \in VLAN, \forall p \in PORT :$ $IsConnectedOnPort(vm1, sw, p) \wedge$ $IsAssignedVLAN(sw, p, vlan) \wedge IsMappedToVXLANOnOVS(sw, vlan, vxlan) \rightarrow$ $[\exists vm2 \in vVM, \exists vNet \in vNET, \exists seg \in Segment, \exists vp \in vPORT :$ $IsConnectedTovNet(vm2, vNet, vp) \wedge (vm1 = vm2) \wedge$ $IsAssignedSeg(vNet, seg) \wedge (seg = vxlan)]$

Table 5. Topology Consistency Properties in FOL

3.3.3 Isolation Verification. As discussed before (Section 3.3.1), 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 3, 4 and 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 2.2.

Example 3.4. 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 ($P11$). To find whether there exist relations' tuples that falsify this property ($\neg P11$), we first formulate the CSP instance using the negative form of the corresponding predicate, which corresponds to the following predicate:

$$\begin{aligned}\neg P11 = \exists vm1 \in iVM, \exists vxlan \in VXLAN, \exists sw \in OVS, \exists vlan \in VLAN, \exists p \in PORT, \\ \forall vm2 \in vVM, \forall vNet \in vNET, \forall seg \in Segment, \forall vp \in vPORT : \\ IsConnectedOnPort(vm1, sw, p) \wedge IsAssignedVLAN(sw, p, vlan) \wedge \\ IsMappedToVXLANOnOVS(sw, vlan, vxlan) \wedge \\ \neg IsConnectedTovNet(vm2, vNet, vp) \vee \neg(vm1 = vm2) \vee \\ \neg IsAssignedSeg(vNet, seg) \vee \neg(seg = vxlan)\end{aligned} \tag{1}$$

By verifying predicate 1 over all the aforementioned relations' instances, the solver finds an assignment such that the above predicate becomes true, which means that the property $P11$ is violated. The predicate instance that caused the violation can be written as follows:

$$\begin{aligned}IsConnectedOnPort(VM\_21, OVS\_1, Port\_21) \wedge \\ IsAssignedVLAN(OVS\_1, Port\_21, vlan\_100) \wedge \\ IsMappedToVXLANOnOVS(OVS\_1, vlan\_100, vxlan\_0 \times 100) \wedge \\ \neg IsConnectedTovNet(VM\_Bapp1, vNet\_B, vPort\_21) \vee \neg(VM\_21 = VM\_Bapp1) \vee \\ \neg IsAssignedSeg(vNet\_B, seg\_512) \vee \neg(seg\_512 = vxlan\_0 \times 100)\end{aligned} \tag{2}$$

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 2 satisfying the constraint. This set of tuples provides the evidence about what values breached the security property $P11$. 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 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 [44], an open-source framework implementing policy as a service for OpenStack.

4.1 Architecture

Figure 8 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 paper, we assume that they are parsable XML files.

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

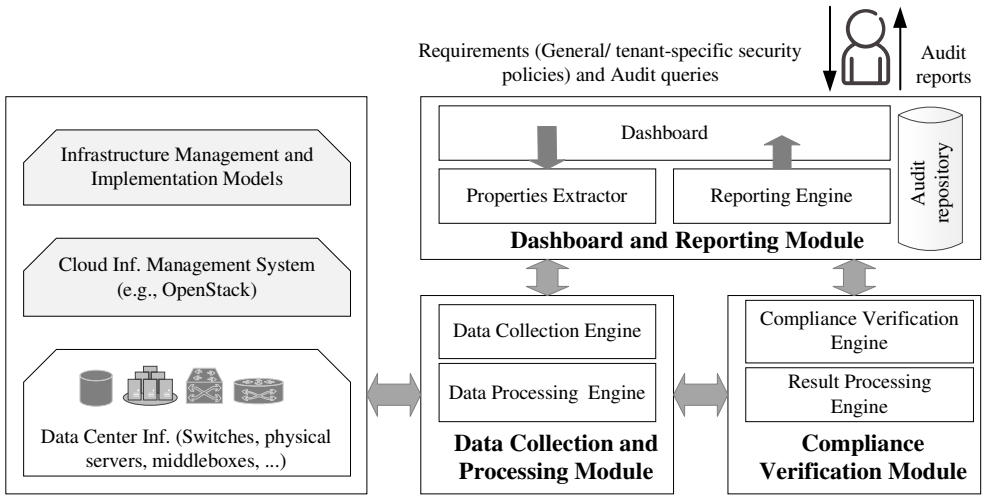

Fig. 8. A High-Level Architecture of our Cloud Auditing Solution

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.

4.2 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 in managing virtual infrastructure and 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 implements the virtual infrastructure. Following is a brief description of Nova, Neutron and OVS:

Nova [45] 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 (IaaS) model.

Neutron [45] This OpenStack project provides tenants with capabilities to build networking topologies through the exposed API, relying on three object abstractions, namely, networks, subnets and

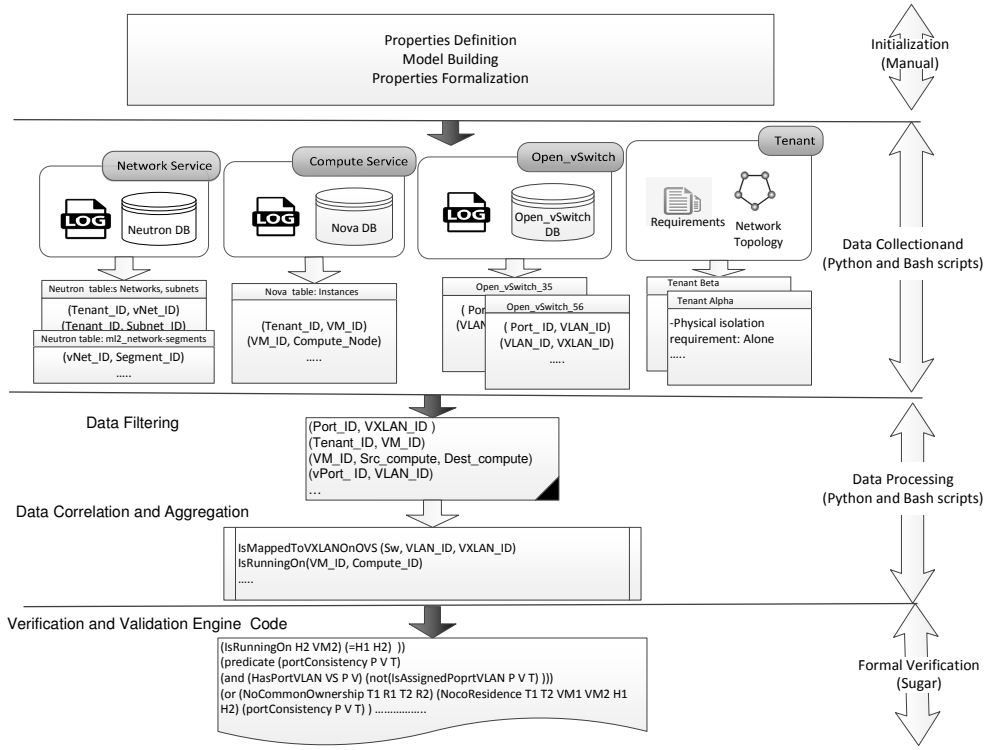

Fig. 9. Our OpenStack-based auditing solution with the example of data collection, formatting, correlation building and Sugar source generation

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)[3] 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 [22] 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.

4.3 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 engine. Figure 9 illustrates the steps of our auditing process. In the following, we describe our implementation details.

[3]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`.

Relations	Sources of Data
BelongsTo	Table *Instances* in Nova database and *Routers, Subnets, Ports* and *networks* in Neutron database
HasRunningVM	Table *Instances* in Nova database
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
DoesnotTrust	The tenant physical isolation requirement input

Table 6. Sample Data Sources in OpenStack, Open vSwitch and Tenants' Requirements

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 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 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 10 relates relations of property $P11$ 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.

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

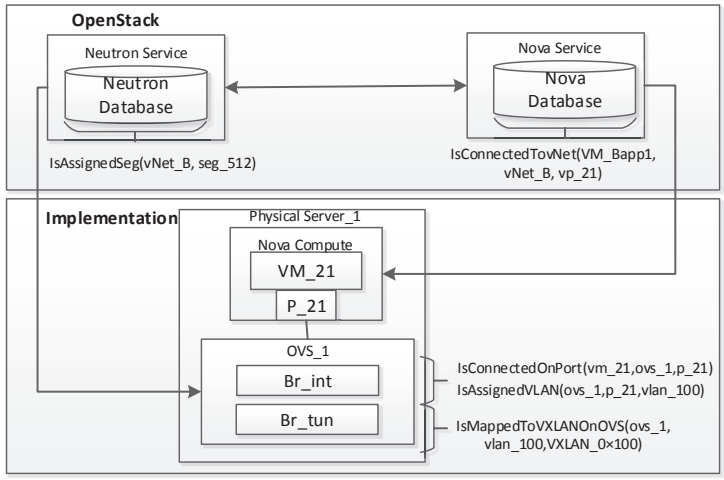

Fig. 10. Mapping of Relations Involved in Property P11 to their Data Sources

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 4.1. 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 2.2.

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 1 for an excerpt of the code) using processed data and translated properties. Finally, Sugar is used to verify the security properties.

The predicate *P11* 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 3.4), meaning that `VM_Bapp1` is connected on the wrong layer 2 virtual network.

Listing 1. Sugar Source Code

```
// Declaration
(domain iVM 0 100000)(domain OVS 0 400)(domain PORT 0 100000)
(domain VLAN 0 10000) (domain VXLAN 0 10000)(doamin vVM 0 100000)
(domain VNET 0 10000)  (domain SEGMENT 0 10000)(domain VPORT 0 100000)
( int  vm1 iVM) (int  vm2 vVM)(int sw OVS) (int  p  PORT)(int vlan  VLAN)
( int  vxlan  VXLAN)(int vnet VNET) (int  seg  SEGMENT) (int vp VPORT)
// Relations  Declarations  and Audit data  as  their  support from the  infrastructure  manangement level
( relation  IsConnectedTovNet 3  (supports  (18045  6100  21)  (18037  6150  7895)(18038  6120  2566)  (  18039 6230  554)(18040  6230
      4771) (966) ) )
( 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 3.3.3, 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 4.2. From Example 4.1, 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 Integration Into OpenStack Congress

To demonstrate the service agnostic nature of our framework, we further integrate our system with the OpenStack Congress service [44]. 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 [44]. 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.

5 EXPERIMENTS

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

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 [53]. All the verification experiments are run on an Amazon EC2 C4.Large Ubuntu 16.04 machine (2 vCPU and 3.75GB of memory).

5.2 Results

Experimental results for *physical isolation*, *virtual resources isolation* and *port consistency* properties are reported in the preliminary version of this paper [31]. In addition, we consider three additional properties from table 1, where each is selected from one of the three categories defined therein. Thus, we consider for the experiments the following three properties, one from each category:

- *Mapping unicity virtual networks-segments* (P3), which is a topology isolation property checked at the infrastructure management level.
- *Mapping unicity VLANs-VXLANs* (P7), which is a topology isolation property checked at the implementation level.
- *Virtual links consistency* (P11), 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 2.2. 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

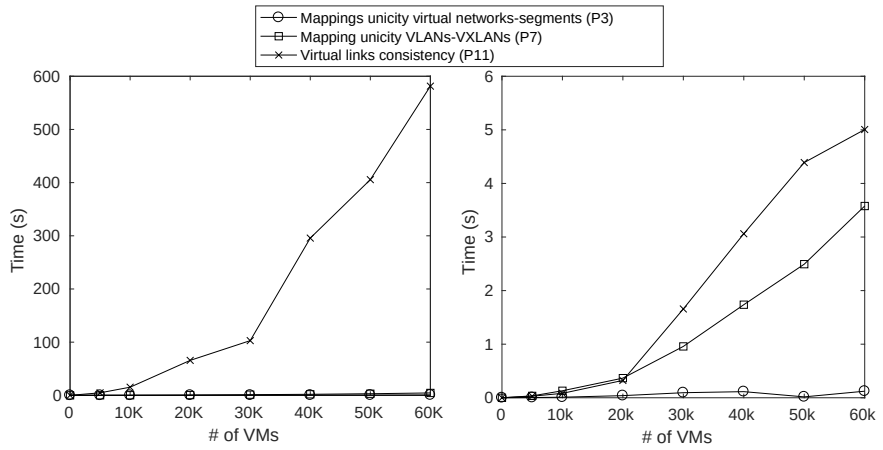

Fig. 11. Verification time as a function of number of VMs for properties P3, P7, and P11: (left side) time to report no breach of compliance, and (right side) time to find the first breach and build evidence of non-compliance

cases. Therefore, we present in Figure 11, 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 2.2 by randomly modifying some VLAN ports and VLAN to VXLAN mappings.

As indicated in the left chart of Figure 11, the time required for verifying P3 and P7, 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 P11 is 102s for 30k VMs and 581s for 60k VMs. The difference in response time for P11 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 P11 to get more acceptable boundaries.

According to Figure 11 (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 Figure12 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 P11 and does not exceed 25% for P3. Also, the highest memory usage observed does not exceed 8% for P11 verification (see the right chart of Figure 12), and 3.3% for the largest dataset for P7. 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 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 P11. According to Figure 13, Treengling solver provides the longest response time

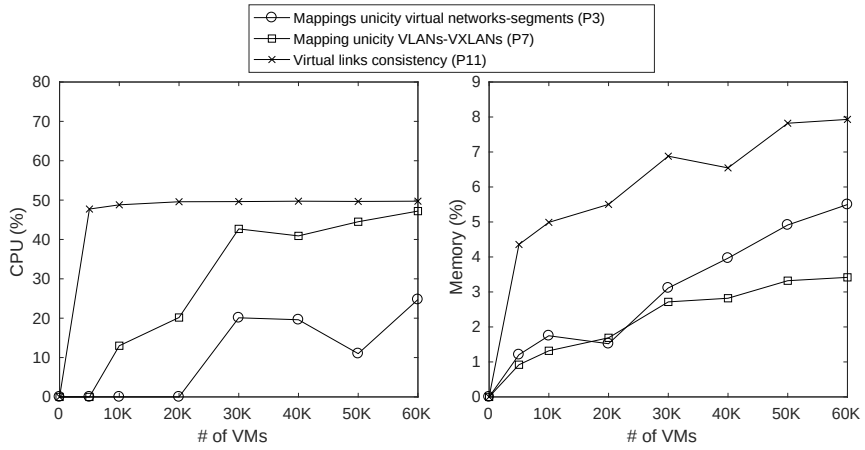

Fig. 12. CPU (left side) and memory (right side) usage to verify no-compliance breach for properties P3, P7 and P11

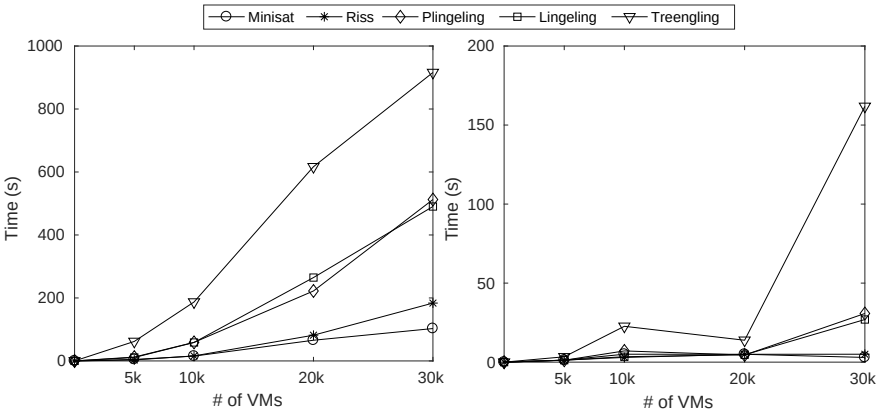

Fig. 13. Verification time using different SAT solvers for P11 as a function of the number of VMs: (left side) time to report no breach of compliance, and (right side) time to find the first breach and build evidence of non-compliance

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 P11. To this end, we consistently split the data supports for the relations `IsConnectedToVnet` and `IsAssignedSeg` of P11 over multiple CSP files (up to 16 files), and repeated the supports for the relations `IsConnectedOnPort`, `IsAssignedVLAN` and `IsMappedToVXLANOnOVS` to maintain data interdependency. Figure 14 reports the response times for the parallel verification of different CSP sub-instances of P11 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 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.

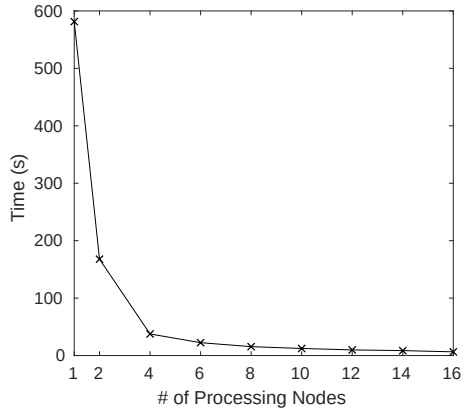

Fig. 14. Verification time as function of the number of processing nodes for P11 for a dataset of 60k VMs, where each processing node verifies a separate CSP sub-instance of P11

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 be found in [40]). Expressing new security properties with other kinds of constraints may require performance to be reassessed through new experiments.

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 [35], which is an interesting future direction.

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) [3], Google Cloud Platform (GCP) [23], Microsoft Azure [36] and VMware virtual Cloud Director (vCD) [54] 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 7 maps the entities of our infrastructure management view model to their counterparts in the cloud platforms cited above.

Eucalyptus [21] is an open source IaaS management system. The Eucalyptus virtual private cloud (VPC) is implemented with MidoNet [37], 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.

7 RELATED WORK

Table 8 summarizes the qualitative comparison between existing works on compliance verification in the cloud and this work. We compare the proposals based on the types of verified properties, structural or operational, 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) [33].

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 7. Mapping virtual infrastructure model entities into different cloud platforms

Proposal	Properties		Coverage		Approach	
	Struct-ural	Operat-ional	One/Multiple layers	Cross-layer	Retro-active	Intercept-and-check
Anteater [32]		•	One		•	
Hassel [29]		•	One		•	
VeriFlow [30]		•	One			•
NetPlumber [28]		•	One			•
Save [7]	•		One		•	
CloudRadar [10]	•		One			•
Xu et al. [55]	•			•	•	
Congress [44]	•		One		•	•
Majumdar et al. [34]	•		One		•	
Madi et al. [31]	•		One	•	•	
This work	•		Multiple	•	•	

Table 8. Comparing Features of Existing Solutions With our Work. The Symbol (•) Indicates that the Proposal Offers the Corresponding Feature

To the best of our knowledge, no work has been tackling auditing 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., [19, 57]). For instance, Anteater [32] 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 [29] 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 [30], NetPlumber [28] (extension of [29]), and AP verifier [56] 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 [57] 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 [19] 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. [39] 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., [6, 49]) focus on firewalls and security groups. Probst et al. [49] 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 [6] 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.

Other works focus on virtualization aspects (e.g., [7–9]). Bleikertz et al. [7, 9] 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. [10] 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 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 [20], 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. [55] 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. However, they 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.

There exist other works (e.g., [44], [11], [33]) offering runtime security policy checking and enforcement in the cloud. Our previous work in [33] proactively verifies security compliance efficiently through pre-computation by utilizing dependency models. Weatherman [11] aims at mitigating misconfigurations and enforcing security policies in a virtualized infrastructure.

Congress [44] 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 the contributions (See Section 4 for details).

In the same fashion as the current work, formal verification approaches in [2, 15, 34] are proposed for checking security compliance in other security domains, mainly Identity and Access Control. Majumdar et al. [34] 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. [15] use FOL to express Role-based Access Control (RBAC) policies and rely on an off-the-shelf SMT solver to analyze them. In [2], 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.

8 CONCLUSION

Auditing compliance of the cloud with respect to security standards faces several challenges. In this paper, 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 it was shown in this paper, 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. As future directions, 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. A second area of investigation is to extend the list of security properties with the operational properties. This would allow to check the compliance of the forwarding network functionality.

Acknowledgment. We thank the anonymous reviewers for their insightful comments and suggestions. This work is partially supported by the Natural Sciences and Engineering Research Council of Canada and Ericsson Canada under CRD Grant N01823 and by PROMPT Quebec.

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. 9

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. 3

Network Segments Isolated broadcast domains within a network. 2

Open vSwitch Open-source software switch implementation designed to be used in hypervisors to provide connectivity to guest VMs. 4

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 [17] for detailed statistics). 2, 4

Overlay Networks Virtual networks that create a virtual topology on top of the physical network [12]. 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. 3

TPM Stands for Tursted Platform Module. It is a standard [1] for dedicated microcontrollers endowed with cryptographic keys to secure the hardware. 6

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. 2, 3

Virtual Switches Software-based switches running at the hypervisor-level and provide connectivity to virtual machines (VMs). 4

VLAN A standardized [25] 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. 3, 4

VTEP Virtual bridges responsible for encapsulating and de-encapsulating packets in overlay networks. 9

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. 4

REFERENCES

[1] ISO. org. 2013. ISO/IEC 11889-1:2009.
[2] Perry Alexander, Lee Pike, Peter Loscocco, and George Coker. 2015. Model Checking Distributed Mandatory Access Control Policies. *ACM Trans. Inf. Syst. Secur.* 18, 2, Article 6 (July 2015), 25 pages. https://doi.org/10.1145/2785966
[3] Amazon. 2017. Amazon Virtual Private Cloud. Available at: https://aws.amazon.com/vpc.
[4] Mihir Bellare and Bennet Yee. 1997. *Forward integrity for secure audit logs.* Technical Report. Citeseer.
[5] Mordechai Ben-Ari. 2012. *Mathematical logic for computer science.* Springer Science & Business Media, London.
[6] Sören Bleikertz. 2010. *Automated Security Analysis of Infrastructure Clouds.* Master's thesis. Technical University of Denmark and Norwegian University of Science and Technology.
[7] Sören Bleikertz, Thomas GroSS, Matthias Schunter, and Konrad Eriksson. 2011. Automated Information Flow Analysis of Virtualized Infrastructures. In *ESORICS (Lecture Notes in Computer Science)*, Vijay Atluri and Claudia Díaz (Eds.), Vol. 6879. Springer, Berlin, Heidelberg, 392–415.

[8] Sören Bleikertz, Thomas Groß, and Sebastian Mödersheim. 2011. Automated Verification of Virtualized Infrastructures. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop (CCSW '11)*. ACM, New York, NY, USA, 47–58. https://doi.org/10.1145/2046660.2046672

[9] Sören Bleikertz, Thomas Gross, M. Schunter, and K. Eriksson. 2010. *Automating Security Audits of Heterogeneous Virtual Infrastructures*. Technical Report RZ3786. IBM.

[10] Sören Bleikertz, Carsten Vogel, and Thomas Groß. 2014. 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)*. ACM, New York, NY, USA, 26–35. https://doi.org/10.1145/2664243.2664274

[11] Sören Bleikertz, Carsten Vogel, Thomas Groß, and Sebastian Mödersheim. 2015. Proactive Security Analysis of Changes in Virtualized Infrastructures. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC 2015)*. ACM, New York, NY, USA, 51–60. http://doi.acm.org/10.1145/2818000.2818034

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

[13] Cloud Security Alliance. 2014. Cloud control matrix CCM v3.0.1. https://cloudsecurityalliance.org/research/ccm/

[14] Cloud Security Alliance. 2016. Cloud Computing Top Threats in 2016.

[15] Carlos Cotrini, Thilo Weghorn, David Basin, and Manuel Clavel. 2015. Analyzing First-Order Role Based Access Control. In *2015 IEEE 28th Computer Security Foundations Symposium*. IEEE, Verona, Italy, 3–17.

[16] Crandall et al. 2012. *Virtual Networking Management White Paper*. Technical Report. DMTF. DMTF Draft White Paper.

[17] datacenterknowledge. 2015. Survey: One-Third of Cloud Users' Clouds are Private, Heavily OpenStack. Available at: http://www.datacenterknowledge.com.

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

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

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

[21] Hewlett Packard Enterprise. 2017. HPE Helion Eucalyptus. Available at: http://www8.hp.com/us/en/cloud/helion-eucalyptus.html.

[22] Open Networking Foundation. 2013. OpenFlow Switch Specification. Available at:http://www.gesetze-im-internet.de/englisch_bdsg.

[23] Google. 2017. Google Compute Engine subnetworks beta. Available at: https://cloud.google.com.

[24] Stephen Gutz, Alec Story, Cole Schlesinger, and Nate Foster. 2012. Splendid Isolation: A Slice Abstraction for Software-defined Networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*. ACM, New York, NY, USA, 79–84. http://doi.acm.org/10.1145/2342441.2342458

[25] Institute of Electrical and Electronics Engineers. 2005. Ieee 802.1q- 2005. 802.1q - Virtual Bridged Local Area Networks.

[26] ISO Std IEC. 2005. ISO 27002:2005.

[27] ISO Std IEC. 2012. ISO 27017.

[28] Peyman Kazemian, Michael Chan, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. 2013. Real Time Network Policy Checking Using Header Space Analysis. In *NSDI*. USENIX, Lombard, IL, 99–111.

[29] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX, San Jose, CA, 113–126. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/kazemian

[30] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. 2013. 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)*. USENIX, Lombard, IL, 15–27. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/khurshid

[31] Taous Madi, Suryadipta Majumdar, Yushun Wang, Yosr Jarraya, Makan Pourzandi, and Lingyu Wang. 2016. 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)*. ACM, New York, NY, USA, 195–206. http://doi.acm.org/10.1145/2857705.2857721

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

[33] Suryadipta Majumdar, Yosr Jarraya, Taous Madi, Amir Alimohammadifar, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. 2016. *Proactive Verification of Security Compliance for Clouds Through Pre-computation: Application to OpenStack.* Springer International Publishing, Cham, 47–66.
[34] Suryadipta Majumdar, Taous Madi, Yushun Wang, Yosr Jarraya, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. 2015. Security Compliance Auditing of Identity and Access Management in the Cloud: Application to OpenStack. In *IEEE CloudCom*. IEEE, Vancouver, Canada, 58–65.
[35] Ruben Martins, Vasco Manquinho, and Inês Lynce. 2012. An overview of parallel SAT solving. *Constraints* 17, 3 (01 Jul 2012), 304–347.
[36] Microsoft. 2016. Microsoft Azure Virtual Network. Available at: https://azure.microsoft.com.
[37] Midokura. 2017. Run MidoNet at Scale. Available at: http://www.midokura.com/midonet/.
[38] H. Moraes, M. A. M. Vieira, Í. Cunha, and D. Guedes. 2016. Efficient virtual network isolation in multi-tenant data centers on commodity ethernet switches. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, Vienna, Austria, 100–108.
[39] Yogesh Mundada, Anirudh Ramachandran, and Nick Feamster. 2011. Silverline: Data and Network Isolation for Cloud Services. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'11)*. USENIX Association, Berkeley, CA, USA, 13–13. http://dl.acm.org/citation.cfm?id=2170444.2170457
[40] Naoyuki Tamura. 2010. Syntax of Sugar CSP description. Available at: http://bach.istc.kobe-u.ac.jp/sugar/current/docs/syntax.html.
[41] NIST, SP. 2003. NIST SP 800-53. , 800–53 pages.
[42] OpenStack. 2014. Ossa-2014-008: Routers can be cross plugged by other tenants. Available at: https://security.openstack.org/ossa/OSSA-2014-008.html.
[43] OpenStack. 2014. OSSA-2014-008: Routers can be cross plugged by other tenants. Available at: https://security.openstack.org/ossa/OSSA-2014-008.html.
[44] OpenStack. 2014. Policy as a Service ("Congress"). Available at:http://wiki.openstack.org/wiki/Congress.
[45] OpenStack. 2015. OpenStack open source cloud computing software. Available at: http://www.openstack.org.
[46] Diego Perez-Botero, Jakub Szefer, and Ruby B. Lee. 2013. Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers. In *Proceedings of the 2013 International Workshop on Security in Cloud Computing (Cloud Computing '13)*. ACM, New York, NY, USA, 3–10.
[47] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon, Martin Casado, and Scott Shenker. 2009. Extending Networking into the Virtualization Layer. In *HotNets*. ACM, YorkCity,NY.
[48] Penny Pritzker and Patrick D. Gallagher. 2013. *NIST Cloud Computing Standards Roadmap*. Technical Report. NIST, Gaithersburg, MD, United States. 108 pages. NIST Special Publication 500-291.
[49] Thibaut Probst, Eric Alata, Mohamed Kaâniche, and Vincent Nicomette. 2014. An Approach for the Automated Analysis of Network Access Controls in Cloud Computing Infrastructures. In *Network and System Security*. Springer, Xi?an, China, 1–14.
[50] Kui Ren, Cong Wang, and Qian Wang. 2012. Security Challenges for the Public Cloud. *IEEE Internet Computing* 16, 1 (Jan 2012), 69–73.
[51] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. 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)*. ACM, New York, NY, USA, 199–212.
[52] Cisco Systems Sean Convery. 2002. Hacking Layer 2: Fun with Ethernet switches. BlackHat Briefings.
[53] Naoyuki Tamura and Mutsunori Banbara. 2008. Sugar: A CSP to SAT translator based on order encoding. *Proceedings of the Second International CSP Solver Competition* (2008), 65–69.
[54] VMware. 2017. vCloud Director. Available at: https://www.vmware.com/fr/products/vcloud-director.html.
[55] Yang Xu, Yong Liu, Rahul Singh, and Shu Tao. 2015. Identifying SDN State Inconsistency in OpenStack. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15)*. ACM, New York, NY, USA, 11:1–11:7.
[56] Hongkun Yang and Simon S Lam. 2013. Real-time verification of network properties using Atomic Predicates. In *ICNP*. IEEE, Goettingen, Germany, 1–11.
[57] Hongyi Zeng, Shidong Zhang, Fei Ye, Vimalkumar Jeyakumar, Mickey Ju, Junda Liu, Nick McKeown, and Amin Vahdat. 2014. Libra: Divide and conquer to verify forwarding tables in huge networks. In *(NSDI 14). Seattle, WA: USENIX Association*. USENIX Association, Seattle, WA, 87–99.
[58] Shuyuan Zhang and Sharad Malik. 2013. SAT Based Verification of Network Data Planes. In *Automated Technology for Verification and Analysis*, Dang Van Hung and Mizuhito Ogawa (Eds.). Lecture Notes in Computer Science, Vol. 8172. Springer International Publishing, Cham, 496–505.

Received March 2017; revised November 2017; accepted August 2018