Modeling NFV Deployment to Identify the Cross-level Inconsistency Vulnerabilities

L T Sudershan
CIISE
Concordia University
Montreal, QC, Canada
s_akshma@encs.concordia.ca

Mengyuan Zhang
Ericsson Security Research
Ericsson Canada
Montreal, QC, Canada
mengyuan.zhang@ericsson.com

Alaa Oqaily
CIISE
Concordia University
Montreal, QC, Canada
a_oqaily@encs.concordia.ca

Gagandeep Singh Chawla
CIISE
Concordia University
Montreal, QC, Canada
g_chawla@encs.concordia.ca

Lingyu Wang
CIISE
Concordia University
Montreal, QC, Canada
wang@encs.concordia.ca

Makan Pourzandi
Ericsson Security Research
Ericsson Canada
Montreal, QC, Canada
makan.pourzandi@ericsson.com

Mourad Debbabi
CIISE
Concordia University
Montreal, QC, Canada
debbabi@encs.concordia.ca

***Abstract*—By providing network functions through software running on standard hardware, Network Functions Virtualization (NFV) brings many benefits, such as increased agility and flexibility with reduced costs, as well as additional security concerns. Although existing works have examined various security issues of NFV, such as vulnerabilities in VNF software and DoS, there has been little effort on a security issue that is intrinsic to NFV, i.e., as an NFV environment typically involves multiple abstraction levels, the inconsistency that may arise between different levels can potentially be exploited for security attacks. In this paper, we propose the first NFV deployment model to capture the deployment aspects of NFV at different abstraction levels, which is essential for an in-depth study of the inconsistencies between such levels. Based on the model and an implemented NFV testbed, we present concrete attack scenarios in which the inconsistencies are exploited to attack the network functions in a stealthy manner. Finally, we study the feasibility of detecting the inconsistencies through verification.**

***Index Terms*—NFV Security, NFV Deployment, Inconsistency, Verification**

I. Introduction

As one of the main technology pillars of network softwarization and 5G, Network Functions Virtualization (NFV) is seeing rapid adoption especially in the telecommunication industry [16]. By providing network functions through software running on standard hardware, NFV enables network service providers to deploy dynamic, agile and scalable Network Services (NS). Such benefits come from the fact that an NFV deployment stack is usually an integration of various virtualization and SDN technologies together with network orchestration and automation tools.

Despite such advantages, the increased complexity of an NFV stack means the attack surface of NFV environments will be significantly larger than that of traditional networks, leading to novel security vulnerabilities and threats [13]. Existing works [11], [19], [32], [38] have addressed various security threats in NFV (e.g., vulnerabilities in VNFs, vulnerabilities due to orchestration and management complexities, and vulnerabilities resulting from the lack of interoperability) and proposed corresponding solutions (e.g., hypervisor introspection, secure zoning, and image signing).

However, a security issue that is intrinsic to NFV has received little attention, i.e., as NFV environments typically involve several levels of abstraction, the inconsistency between those levels may arise due to the lack of proper synchronization between management and orchestration components, which can be exploited by malicious adversaries for security attacks. Although the inconsistency threats have been investigated in other contexts such as cloud and SDN [21], [37], it has only received limited attention in NFV [9], [31]–[33] and there lacks an in-depth study about *how such inconsistencies may be instantiated and exploited based on concrete deployment of NFV*, and *how such inconsistencies may be modeled and identified based on existing data in NFV*.

In this paper, we first observe a gap between what is needed for understanding the inconsistencies (i.e., detailed information about the NFV deployment) and what is currently available in the ETSI NFV reference architecture [4]. The observation leads us to devise a novel NFV deployment model based on studying existing NFV deployment in open source platforms. Our deployment model complements the ETSI NFV architecture with details about all the critical components of an NFV environment, their relationships, and their levels of abstraction. Our deployment model enables us to present concrete attack scenarios in which the inconsistency vulnerabilities are exploited to attack NFV in a stealthy manner. We validate our model and attacks through implementation based on a real NFV testbed. Finally, we present a feasibility study on the verification solution by gathering information required for identifying the consistency. In summary, the main contributions of this work are threefold:

1) To the best of our knowledge, our NFV deployment model is the first effort to capture how NFV is deployed

2330-2186/19/$31.00 ©2019 IEEE
DOI 10.1109/CloudCom.2019.00034

in the real world based on open source platforms, and we believe such a model may see many other applications.
2) The attack scenarios demonstrate both the feasibility and the severeness of inconsistency-based security threats, which could draw more attention to this issue and provide insights to its mitigation.
3) Our study about the information required for identifying inconsistencies serves as a foundation for developing security verification solutions to detect such threats.

The remainder of the paper is organized as follows. Section II reviews the ETSI NFV architecture and provides a motivating example. Section III introduces our NFV deployment model. Section IV presents the attack scenarios. Section V studies the feasibility of consistency verification. Section VI reviews related work and Section VII concludes the paper.

II. Preliminaries

In this section, we first review the ETSI NFV architecture and then show what additional information is needed to understand the inconsistencies through a motivating example.

A. The ETSI NFV Reference Architecture

Figure 1 shows the NFV reference architecture from ETSI [5] (the callouts are not a part of ETSI NFV architecture and will be explained in Section II-B). The architecture includes three main blocks, namely, VNF, NFVI, and MANO. First, Virtual Network Functions (VNFs) provide a high-level representation of network functions. Second, NFV Infrastructure (NFVI) represents the cloud infrastructure that provides basic compute, network and storage capabilities. Third, NFV Management and Orchestration (MANO) supports dynamically managing and orchestrating the lifecycle of physical and virtual resources, which is further divided into three managerial components, Virtual Infrastructure Manager (VIM), Virtual Network Function Manager (VNFM), and Network Function Virtualization Orchestrator (NFVO), to complete the entire deployment process.

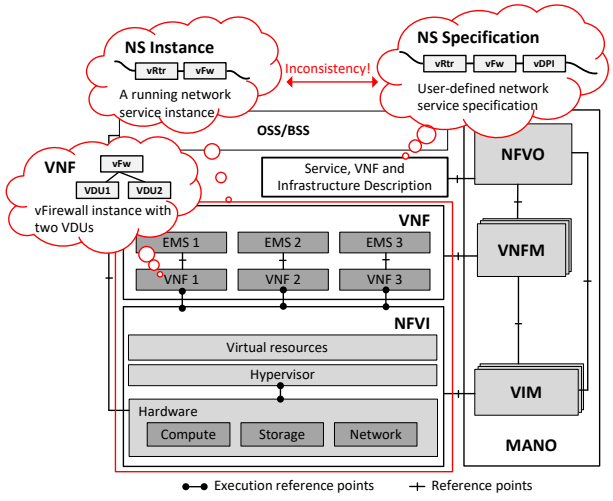

Fig. 1: The ETSI NFV reference architecture [5]

B. Motivating Example

To illustrate what might be missing in the ETSI NFV architecture when it comes to studying the inconsistencies, Figure 2 shows a simple example of inconsistency. First, the NS specification (top of the figure) shows that Bob has specified a virtual firewall (`vFw`) with two Virtual Deployment Units (VDUs), i.e., `VDU2` with `pfSense` for routing and firewalling (the rule shows that any SSH requests should be rejected), and `VDU3` with `Snort` for IDS. Second, the corresponding NS instance depicts the changing state of `VDU2` before and after an attack is launched by another user, Alice. By exploiting a VM hopping vulnerability (e.g., CVE-2015-3456 (Venom), CVE-2015-7835, and CVE-2018-10853), Alice gains control of `VDU2` and modifies its `pfSense` rule to allow SSH requests to the Web server. Importantly, such a change made by Alice on `VDU2` will not be reflected at the higher level (in the VNF Descriptor), which leads to a stealthy attack caused by the inconsistency between the two levels.

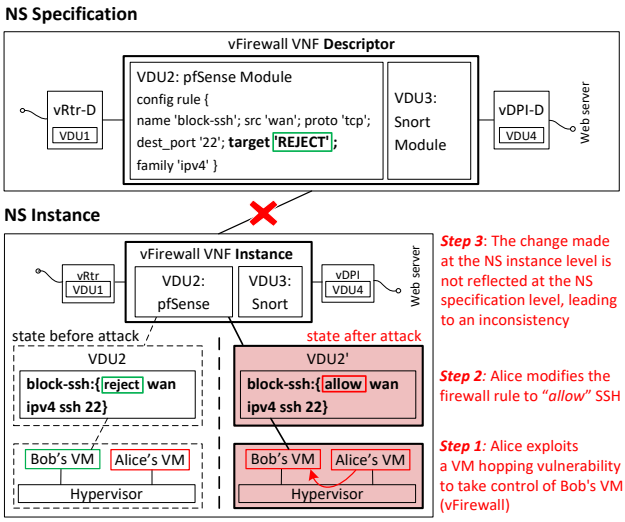

Fig. 2: An inconsistency between the NS specification and instance

To model this attack using the ETSI architecture, we revisit Figure 1. As the callouts show, the attack involves the following NFV deployment details missing in the ETSI architecture.

- The mappings between VDUs and VNFs, NS instance and VNF, NS instance and NFVI are all absent from the ETSI architecture, e.g., the mapping between the virtual firewall descriptor and the two VDUs is essential to understand the inconsistency in our example.
- The mapping between the management and implementation layers of clouds [23] is also missing in the ETSI model, e.g., the mapping between Bob's VM and `VDU2` allows us to link the attack *step 1* and *step 2*.
- Other details like the traffic steering related to the dependencies between the managerial components and the corresponding virtual resources are also missing.

To complement the ETSI architecture with such missing details, the next section devises an NFV deployment model.

III. The NFV Deployment Model

This section proposes a multilevel deployment model for NFV environments. We first provide an overview and then detail each level of the model.

A. Overview

Our NFV deployment model is based on the NFV deployment of several popular open source platforms including Open Networking Automation Platform (ONAP) [25], Tacker [28], OpenStack [28], and OpenDaylight [26]. We extract the operational dependencies between managerial components (i.e., NFVO, VNFM, VIM, and SDN-C) and functional elements (e.g., VNF, VM, SFC, and virtual switches). We separate the NFV stack into four levels as follows. The first level is based on the common deployment aspects from both ONAP and Tacker, i.e., NFVO and VNFM collaborate together to process descriptors and perform high-level management of VNFs and their connectivity. The second and third levels are based on NIST's cloud architecture [23], i.e., the management layer for cloud management operations and the implementation layer for underlying implementations.

Figure 3 presents our multilevel NFV deployment model (middle) with the mapping to the ETSI NFV reference architecture (left) and our motivating example (right). Specifically,

1) The *Service Orchestration* (L1) level is the entry point for NFV users to input their intended design specifications of network services (NS). The managerial components at this level are VNFM and NFVO, which control the onboarding of VNF and NS based on user specifications.
2) The *Resource Management* (L2) level instantiates the specifications. The managerial component at this level is VIM, which receives requests from NFVO and deploys the corresponding NS instance accordingly.
3) The *Virtual Infrastructure* (L3) level incorporates the virtual compute, storage, and network resource pool. The SDN controller (SDN-C) plays a critical role in managing network traffic steering at this level.
4) Finally, the *Physical Infrastructure* (L4) level depicts all physical resources to complete an end-to-end NFV stack.

Table I lists the main abbreviations we use in this paper.

Acronym	Full Name	Acronym	Full Name
CP	Connection Point	SFC	Service Function Chain
EMS	Element Management System	SDN-C	SDN Controller
FC	Flow Classifier	VDU	Virtual Deployment Unit
MANO	Management and Orchestration	VIM	Virtual Infrastructure Manager
NFP	Network Function Path	VM	Virtual Machine
NFVI	NFV Infrastructure	VNF	Virtual Network Functions
NFVO	NFV Orchestrator	VNFD	VNF Descriptor
NS	Network Service	VNFFG	VNF Forwarding Graph
NSD	Network Service Descriptor	VNFFGD	VNFFG Descriptor
PPG	Port Pair Group	VNFM	VNF Manager

TABLE I: Main acronyms used in this paper

B. L1: Service Orchestration Level

L1 includes the deployment components such as NFVO, VNFM, NSD, VNF, and EMS. Two key NFV managerial components at L1 are NFVO and VNFM, which manage NSs in the form of catalogs such as network service descriptor (NSD). NSD can include other descriptors, such as VNF descriptors (VNFD) and VNF forwarding graph descriptors (VNFFGD). Descriptors provide all necessary network service specifications and implementation information in structured templates for orchestration. For example, the traffic steering is defined in VNFFGD; the incoming network traffic would first pass flow classifiers (FC) before being forwarded to a specific network function path (NFP). An NFP connects VNFs with connection points (CPs) using virtual links.

To deploy an NS instance, an NFV user specifies the descriptors as inputs for NFVO. After NFVO validates the technical accuracy of such inputs, NFVO and VNFM inform VIM to allocate the underlying resources to implement the NS. NFVO manages the VNFFG (network topology), whereas VNFM with Element Management System (EMS) performs high-level management of the individual VNFs based on users' specifications, e.g., the logical mapping between VDUs and VNFs. However, the implementation details at lower levels are not reflected in VNFM. We place VNFs at the same level of VNFM because of this operational dependency.

Example 1: Tenant Bob wants to deploy an NS with three VNFs to steer traffic to two destinations. The right side of L1 in Figure 3 shows this NS is deployed using four descriptors. The NSD for this instance is the composition of `VNFFG1D`, `vRtrD`, `vFwD`, and `vDPID`. The lower part of this subfigure bridges the descriptors with the implementation. For example, `vFw` at this level includes the interpretation of its corresponding descriptor `vFwD` and the instance identifiers of `VDU2` and `VDU3` from the lower level. This logical mapping helps to understand the deviation between `vFwD` and `VDU2` should any inconsistencies occur. Logically, `NFP1` contains the sequential order of connection points (e.g., `CP:vRtr`) to chain all three VNFs. `FC1` classifies all HTTP traffic to `NFP1`, while `FC2` sends the management traffic through `NFP2`.

C. L2: Resource Management Level

L2 contains virtual resources such as VDUs, subnets, SFC, network ports, etc., depicting how the NFV virtual resources are created and managed inside the cloud environment. VIM is directly responsible for provisioning, interconnecting and decommissioning these virtual resources contained in an NFVI-PoP domain (an NFVI instance). VDUs at L2 follow a many-to-one relationship with the VNFs from L1 and a one-to-one logical mapping with the VMs from the lower level. VNFFGs from L1 are instantiated as service function chains (SFCs; also referred to as port chains). SFC is a sequence of port pair groups, which consist of one or more port pairs.

Once the resource allocation request is received from NFVO, VIM allocates the compute, storage and network resources corresponding to the given descriptors. Then, the virtual resources, such as VDUs, subnets, network ports, routers, service chains, are instantiated to build an NS. Although VIM can be considered as a part of both L2 and L3, the management operations are executed by cloud tenants from L2 through VIM to directly manage the virtual resources. Hence, VIM is considered as a part of L2 in our model.

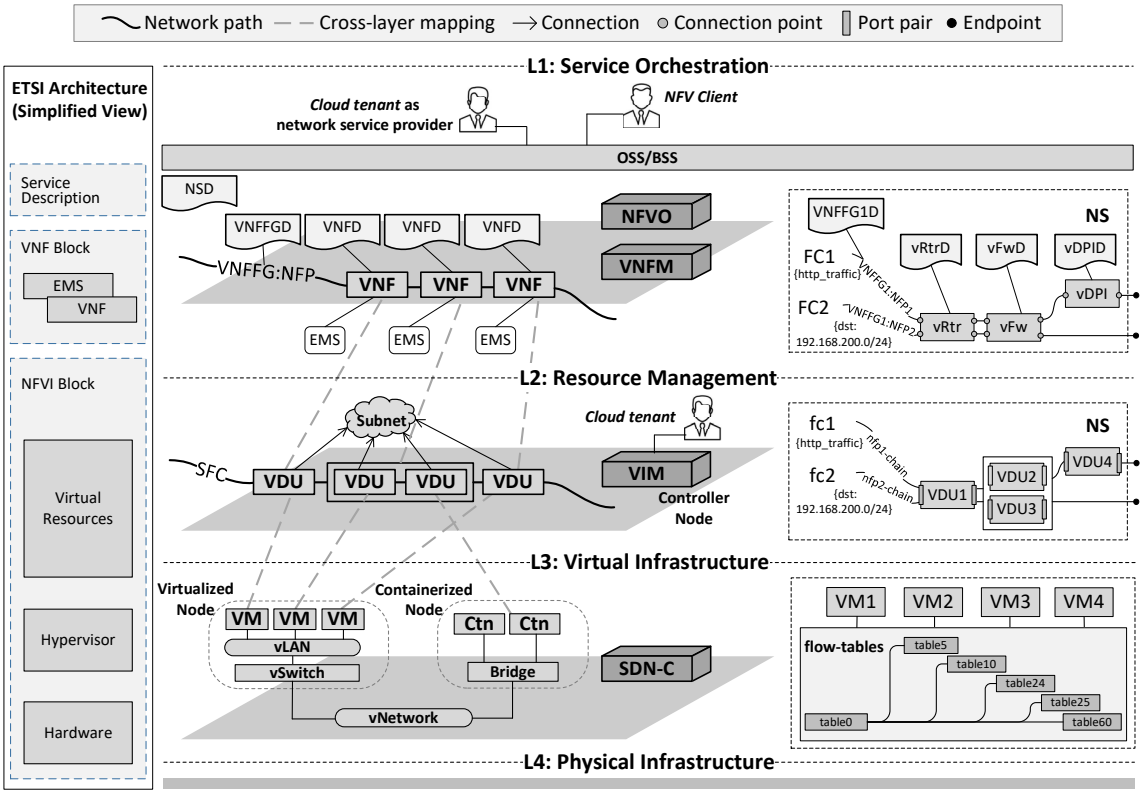

Fig. 3: The multilevel NFV deployment model

Example 2: Once VIM receives the resource creation request from NFVO, it creates the deployment units (e.g., `VDU1`) as shown on L2. These VDUs are then configured based on NSD. For example, the mapping of `vFw` to `VDU2` and `VDU3` is configured to allow HTTP traffic and block SSH traffic. After the creation and configuration of VNFs, the network paths are created as defined in VNFFGD. These function paths are implemented as port-chains `nfp1-chain` and `nfp2-chain` with the respective flow-classifier instances `fc1` and `fc2`. By definition, `nfp1-chain` is an ordered list of port-pair-groups (CPs at L1) `PPG1`, `PPG2` and `PPG3` corresponding to the virtual instances `VDU1`, `VDU2` and `VDU3`, respectively.

D. L3: Virtual Infrastructure Level

L3 includes SDN-C and the virtual networking elements, such as virtual switches, VLANs, VxLANs, virtual routers, flow tables, virtual bridges, and the corresponding interfaces. SDN-C is the management element inside this level (we follow the most widely adopted use case in the SDN usage specification [6] to place SDN controller at this level for providing network connectivity). Flow-tables form the fundamental elements for network traffic steering. These flow-tables are populated with flow-rules, which depict the forwarding behavior of the SFC. Once the service chains are created, the corresponding traffic flows for each chain are deployed in the virtual switches for traffic steering through VMs. The deployment of traffic flow rules is carried out by the SDN-C; this is another reason for us to place SDN-C at this level.

Listing 1: Flow-rule that forwards HTTP traffic to vRtr VNF

```
cookie=0x794188fe368fe901,
    duration=1108281.174s,table=0,
    n_packets=0, n_bytes=0, priority=30,tcp,
    in_port="qvo6094f47a-3f",
    nw_src=192.168.100.11,
    nw_dst=192.168.100.100/32, actions=group:1
```

Example 3: Listing 1 shows the flow-rule that classifies and forwards the HTTP traffic through `vRtr` to the web server. `table0` acts as an FC for the SFC. Each flow-rule has a *match* criteria followed by the *action* that is to be performed for the matched traffic. In case of `nfp1-chain`, simple *match* criteria is to select all HTTP packets originating from the *in_port:* `qvo6094f47a-3f` with the *nw_src* as `192.168.100.11` and the destination as `192.168.100.100/32`. Then the *action* is to forward the matched packets to `vRtr` VNF, which is denoted as `group:1` referencing the VNF's port-pair-group. The traffic is then forwarded to `group:2` (`vFw`) and `group:3` (`vDPI`) before it reaches the actual destination.

E. L4: Physical Infrastructure Level

This level includes all the physical entities (e.g., COTS servers as controller and compute nodes, and physical network functions (PNFs)) that are involved in the NFV stack. The actors involved at this level are typically the physical infrastructure operators or facility managers who also represent the managerial components. The model does not include many details at this level for the sake of clarity.

IV. Exploiting Inconsistencies in NFV Stack

In this section, we first discuss inconsistencies in NFV and explore potential attacks for exploiting the inconsistencies at different levels. We then implement a testbed and two concrete attack scenarios to validate our model.

A. Inconsistencies and Attacks

The inconsistency between the NS specification and instance as discussed in our motivating example (Section II-B) is only a special case. Despite the fact that NFVO is considered as the "brain" of an NFV environment, the other managerial components can operate at each level autonomously, which is referred to as the "split-brain" issue in the literature [7]. For example, VIM and SDN-C can manipulate virtual resources and virtual network freely without going through NFVO. Such autonomous management is intentional in order to effectively manage multiple domains in a single NFV environment (e.g., there can be multiple VIMs managing many NFVI Points of Presence (NFVI-PoP)). However, the lack of synchronization is not intended [31], and it can lead to inconsistencies whenever the states of functional elements managed by two different managerial components differ from each other.

Therefore, the inconsistency may potentially arise between any managerial components and their functional elements inside an NFV environment. To that end, our multilevel NFV deployment model provides a foundation for analyzing potential inconsistencies, as it sufficiently captures the relationships between different levels of components in an NFV environment. For example, the inconsistencies between the management level and the implementation level of cloud [21] and SDN [37] could be mapped to L2 and L3, while the inconsistencies between user specification and the actual deployment could be captured by comparing L1 to the lower levels.

Next, we investigate potential attacks exploiting the cross-level inconsistencies based on our deployment model shown in Figure 3 and a concrete implementation based on OpenStack Tacker [28] and OpenDaylight (ODL) [26] (which will be detailed in Section IV-B). We will focus on possible attacks originated at L2 and L3, respectively, which could cause inconsistencies with the user specifications given at L1.

Threat Model. We assume that the adversary can be a malicious cloud tenant, an admin operator, or an external attacker who controls a virtual machine (e.g., via malware infection) with system privilege. The adversary is assumed to share part of the infrastructure (e.g., compute hosts and physical network) with the victim. We also assume that the cloud infrastructure can have vulnerabilities that the adversaries may identify and exploit. We do *not* assume the adversaries can compromise the managerial elements, and we do *not* assume the adversaries can compromise SDN controllers or switches.

Attacks at L2. The potential attacks originated at this level could lead to the modification of VDUs, port-chain, security groups, etc., which could all lead to an inconsistent state of the NFV system. We discuss several possibilities to achieve such attacks in the following.

- Through VIM, a malicious cloud admin, a cloud operator colluding with an external attacker, or a malicious cloud user exploiting a privilege escalation vulnerability will be able to modify the functional elements of another user's NS. Over time, OpenStack has seen several privilege escalation and sensitive data exposure vulnerabilities, such as OSSA-2016-005 and OSSA-2017-004 [29]. By exploiting such vulnerabilities, an attacker can perform many unauthorized operations, such as updating a service chain by including a malicious VNF, for the system to reach an inconsistent state.
- As illustrated in our motivating example, an attacker can also take over the control of a VM through exploiting a hypervisor vulnerability and then modifying either its configurations or the traffic flow to lead to an inconsistent state in which the attacker's actions at L2 are not reflected at L1 causing a stealthy attack.

Attacks at L3. Existing security threats in SDN, such as malware infection, topology poisoning [15], control plane saturation [34], and state manipulation attacks [36], can be employed to manipulate the traffic flow in L3. We discuss some possible attacks at this level as follows.

- An attacker can send crafted packets to the SDN controller to externally trigger undesirable events that lead to inconsistencies. For example, by enabling/disabling the network interfaces on a host, the attacker can trigger host-related events such as *HOST_JOIN*, *HOST_LEAVE*, etc.
- An attacker can compromise a virtual switch and program it to modify traffic flows to cause inconsistencies. For example, a critical vulnerability (*CVE-2018-1078* [27]) in ODL can be exploited to cause uncontrolled communication between VNFs by programming the switch to reconnect to the network upon new flow update events.

B. Implementation

To validate our deployment model and demonstrate concrete attack scenarios, we have implemented a real NFV testbed with a telemetry NFV network service. We use OpenStack [28] as the VIM, which is considered as an essential cloud management solution by 96% of the CSPs, while more than 60% of the telecom operators are already using OpenStack for their NFV deployments [30]. OpenStack Tacker [28], an official OpenStack project for building a generic NFVM and NFVO based on ETSI MANO Architectural framework, is integrated to deploy and operate virtual network services on the VIM. We adopt the most widely used TOSCA [24] definition standards for defining network service descriptors. An ODL SDN controller is implemented to build an OpenFlow-enabled NFV system. In our implementation, Tacker uses OpenStack Heat [28] for VNF lifecycle management and user-defined VNF descriptors are uploaded to the VNFM module of Tacker through Horizon/CLI. We build our testbed on a SuperServer 6029P-WTR equipped with Intel(R) Xeon(R) Bronze 3104 CPU @ 1.70GHz and 128GB of RAM. Figure 4 illustrates the detailed implementation and depicts different deployment stages as described in Section III.

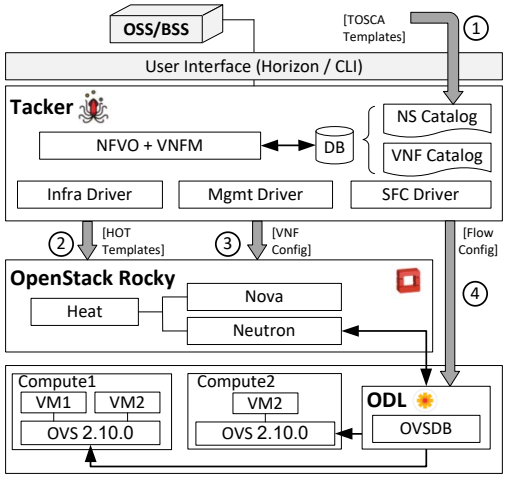

Fig. 4: A real-world NFV deployment on our testbed implemented using OpenStack Tacker and ODL. The circled numbers indicate deployment stages: 1) Onboarding the NS Descriptors, 2) Deploying the VNFs, 3) Configuring the VNFs and 4) Instantiating the NS

Attack Scenario 1. Using our testbed, we have implemented a concrete attack that targets the integrity of a service function chain (SFC). In Figure 5, Bob is a network service provider serving enterprise NFV clients who happens to share the physical infrastructure with a malicious tenant Alice. The red dashed line shows a compromised service chain instance, `nfp1-chain`, which is modified by Alice to include a malicious VNF. However, as our test has shown, such a modification at L2 will not be reflected at L1, leading to an inconsistent state of the NFV stack and a stealthy attack allowing Alice to inspect or modify traffic passing the chain.

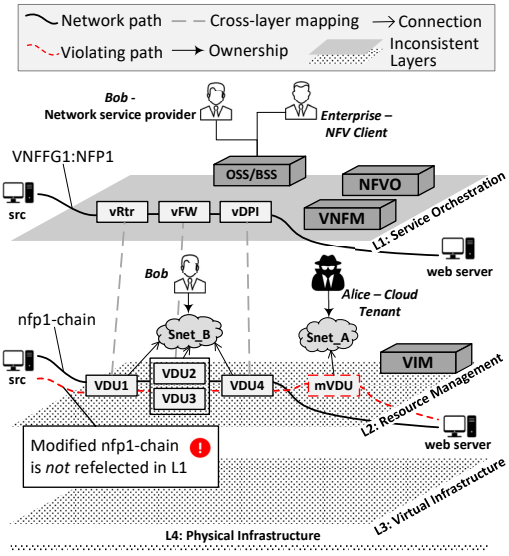

Fig. 5: An implemented attack at L2 causing inconsistency between the path specification `VNFFG1:NFP1` and its instance `nfp1-chain`

More specifically, Figure 6 shows the attack timeline. The port-chain instance (`nfp1-chain`) of `NFP1` consists of three port-pair-groups corresponding to the three VNFs at t_{-i} time. Alice at t_k could perform the aforementioned attack to execute the *neutron port-chain-update* command which would update `nfp1-chain` by adding the port-pair-group of a malicious VNF `mVDU`. Upon the execution of the *neutron port-chain-update* command, the flow-rules will be updated in the virtual switches for `vDPI` to forward the traffic further to `mVDU` as opposed to the path definition in the VNFFGD element of L1. To this end, the `mVDU` VNF is added as a hop in Bob's port-chain allowing Alice to have unauthorized access to any traffic flowing through `NFP1` without being noticed.

Attack Scenario 2. The second concrete attack targets the flow-tables at L3 for causing inconsistencies between L3 and the upper levels. When the network topology gets updated, the SDN controller will install new flow-rules in the virtual switches to reflect the changes. To manipulate the flow-tables at L3, an attacker can trigger a virtual switch reconciliation (a functionality to ensure that switches properly reflect intended controller configurations after restarts) by sending crafted network packets during a network topology update. This would cause the old flow-rules to be installed instead of the new flows. Therefore, the traffic will be steered as specified by the old flow definition contravening the topology update.

Specifically, assuming the end user (NFV Client) updates `NS`'s topology by changing its `VNFFG1` definition at L1 to add `vDPI` to `NFP2`, which will redirect the management server's traffic through `vDPI` for further analysis. Upon the update, the corresponding port-chain (`nfp2-chain`) and flow-rules

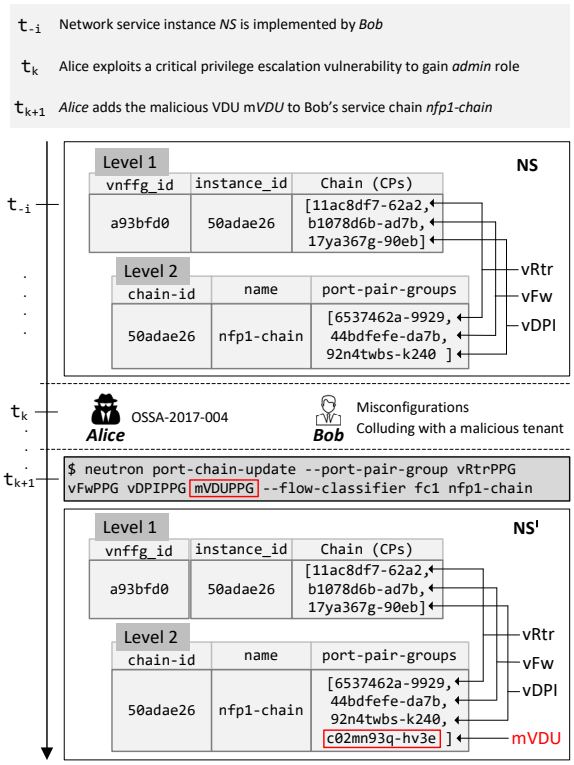

Fig. 6: An illustration of the attack timeline when Alice modifies Bob's port-chain `nfp1-chain` by adding a malicious `mVDU` at L2

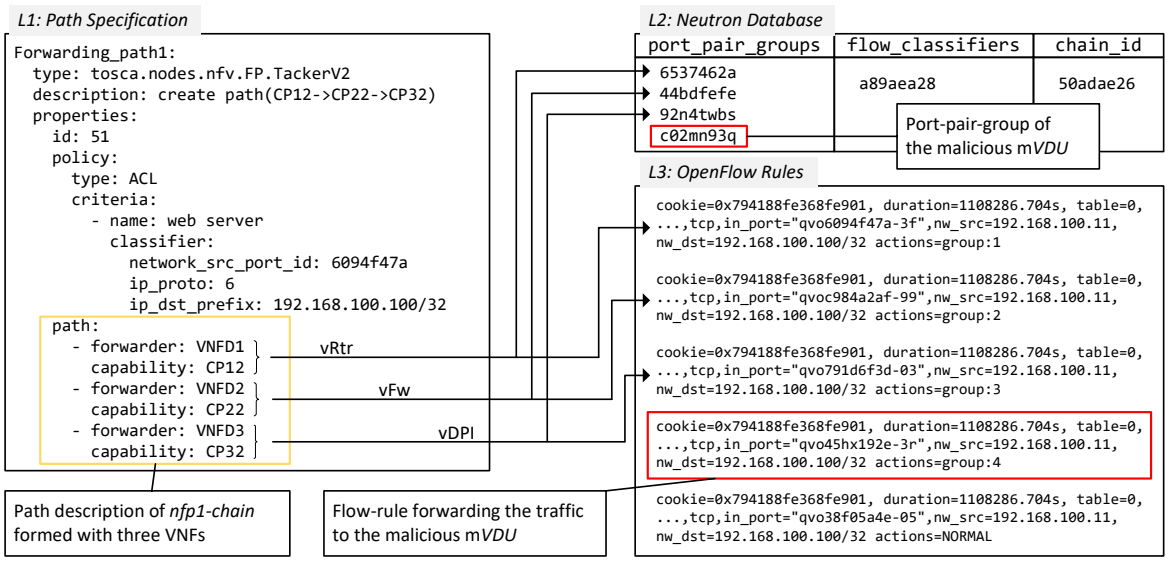

Fig. 7: An example showing the feasiblity of verifying inconsistencies based on data extracted from different levels

in the virtual switches must be updated at L2 and L3, respectively. Meanwhile, an attacker triggers a *SWITCH_LEAVE* event by continuously resetting TCP sessions [36] between `compute1_vswitch` and the controller. When the switch reconnects, the old flow-rules are re-installed as a consequence of the node-reconciliation vulnerability, leading the traffic of `NFP2` to be forwarded to the management server directly without passing `vDPI`. However, L1 and L2 are not aware of this change in the actual traffic flow, leading the NFV system into an inconsistent state.

In addition to the attacks presented in this section, there might be many other ways for exploiting the cross-level inconsistencies in an NFV stack (e.g., attacking the flow-classifier component). To address this issue, we provide a feasibility study on identifying inconsistencies through verification.

V. Feasibility of Consistency Verification

Security verification using formal methods [20], [33], [37] or graph-based approaches [17] has seen applications in clouds and SDN, and it can also provide a viable solution for detecting the aforementioned inconsistencies in NFV. A key challenge is to understand what data needs to be collected from which component of an NFV stack in order to identify the inconsistencies. Based on our multilevel deployment model, we provide some preliminary results on the data collection to show the feasibility of verifying the consistency of NFV.

Specifically, Figure 7 shows an excerpt of the data sources within each level of our deployment model.

- *L1*: the descriptors at this level represent users' requirements, which provide the baseline for verifying consistency. For example, NSD defines the path specification `nfp1-chain` with three VNFs (`vRtr`, `vFw` and `vDPI`) and the corresponding network path.
- *L2*: the information related to VNFs and SFCs can be extracted from VIM through several sources, such as Heat, Nova, Neutron databases, and the VDUs (e.g., VNF configurations and logs). The order of VNFs in a chain is preserved as the order of records in Neutron database, while the number of VNFs in a chain is corresponding to the number of records under the same chain_id. In our example, chain_id *50adae26* contains four VNFs instead of three as defined in NSD. The inconsistency created by `mVDU` could be detected by comparing those data.
- *L3*: flow-tables from the virtual switches contain the information related to the forwarding behavior of NS. The highlighted flow-rule is the maliciously added flow, which forwards the traffic to `mVDU` (*group 4* in the rule), and also shows the inconsistency w.r.t. NSD.

More generally, the user-defined NS specification at L1 and the NS deployment related-data at L2 and L3 represent the current system state and will comprise the main inputs to verification mechanisms for detecting inconsistencies.

VI. Related Work

To the best of our knowledge, this is the first work proposing a concrete model for the deployment aspects of NFV. A few other models of NFV architecture are proposed for different use cases, e.g., mobile edge computing in 5G [8] and efficient VNF placement [22]. Pattaranantakul et al. [31] propose a framework to dynamically manage security functions in NFV. Hoang et al. [14] propose an extended NFV architecture that uses Tacker to support containers.

Unlike our work, which focuses on the inconsistencies, most existing studies on NFV security [11], [19], [32], [38] focus on issues related to virtualization. Lal et al. [19] propose to adapt several well-known best practices like VM separation, hypervisor introspection, and remote attestation. Pattaranantakul et al. [32] adopt best practices like access control to address virtualization-related threats in NFV.

Most of the existing verification solutions in NFV focus on service function chaining (SFC) [10], [12], [35], [39]. Fayazbakhsh et al. [10] discuss the need for verifying the NS

properties related to functionality, performance and accounting. Zhang et al. [39] propose a scheme called vSFC to verify a range of SFC violations. Unlike our work, these solutions do not consider the consistency w.r.t. given specifications. The verification approach proposed by Shin et al. [33] can verify inconsistency as one of the security properties, although the model of dependencies between service components is rather simplistic and lacks most of the essential details in our model.

Other works (e.g., [1]–[3], [18]) focus on detecting and mitigating security threats in NFV. Basile et al. [1] propose to add a new policy manager component to enforce security policies during deployment and configuration of security functions. Blaise et. al [2] propose an anomaly detection solution based on Markov chain property to ensure the correctness of VNF placement in a chain. Coughlin et al. [3] integrate trusted computing based solution Intel SGX to enforce privacy with secure packet processing. Our deployment model could potentially be used to study the integration of such solutions.

VII. Conclusion

In this work, we presented a multilevel NFV deployment model, which complements the ETSI architecture with essential details for exploring potential inconsistency vulnerabilities in NFV. Our model showed that the autonomous management components at different levels render cross-level inconsistencies an intrinsic threat to NFV. We validated our model by implementing an NFV testbed and concrete attack scenarios. Finally, our study on the data collection paved the way for developing verification-based detection solution. As future work, in addition to formalizing the proposed NFV deployment model, we will extend our network-centric model to include other computing or storage managerial components. Furthermore, we intend to develop security verification tools based on open source NFV environments.

Acknowledgment

The authors thank the anonymous reviewers for their valuable comments. This work is partially supported by the Natural Sciences and Engineering Research Council of Canada and Ericsson Canada under the NSERC/Ericsson Industrial Research Chair (IRC) in SDN/NFV Security.

References

[1] C. Basile, A. Lioy, C. Pitscheider, F. Valenza, and M. Vallini. A novel approach for integrating security policy enforcement with dynamic network virtualization. In *NetSoft'15*, pages 1–5, 2015.

[2] A. Blaise, S. Wong, and A. H. Aghvami. Virtual network function service chaining anomaly detection. In *ICT'18*, pages 411–415, 2018.

[3] M. Coughlin, E. Keller, and E. Wustrow. Trusted click: Overcoming security issues of NFV in the cloud. In *SDN-NFV@CODASPY'17*, pages 31–36, 2017.

[4] ETSI. ETSI. Available at: https://www.etsi.org/.

[5] ETSI. Network functions virtualisation architectural framework, 2013.

[6] ETSI. Network functions virtualisation - Report on SDN usage in NFV architectural framework, 2015.

[7] ETSI. Network function virtualisation (NFV); Reliability; Report on the resilience of NFV-MANO critical capabilities, 2017.

[8] ETSI. MEC in 5G networks, 2018.

[9] ETSI. Network functions virtualisation (NFV) release 3; Management and orchestration; Architecture enhancement for security management specification, 2018.

[10] S. K. Fayazbakhsh, M. K. Reiter, and V. Sekar. Verifiable network function outsourcing: Requirements, challenges, and roadmap. In *Workshop on Hot topics in middleboxes and network function virtualization (HotMiddlebox'13)*, pages 25–30, 2013.

[11] M. D. Firoozjaei, J. P. Jeong, H. Ko, and H. Kim. Security challenges with network functions virtualization. *Future Generation Computer Systems*, 67:315–324, 2017.

[12] M. Flittner, J. M. Scheuermann, and R. Bauer. Chainguard: Controller-independent verification of service function chaining in cloud computing. In *NFV-SDN'17*, pages 1–7, 2017.

[13] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[14] C.-P. Hoang, N.-T. Dinh, and Y. Kim. An extended virtual network functions manager architecture to support container. In *ICISS'18*, pages 173–176, 2018.

[15] S. Hong, L. Xu, H. Wang, and G. Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *NDSS'15*, pages 8–11, 2015.

[16] Intel. Realising the benefits of network functions virtualisation in telecoms networks, 2014.

[17] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *NSDI'13*, pages 15–27, 2013.

[18] S. Lal, A. Kalliola, I. Oliver, K. Ahola, and T. Taleb. Securing VNF communication in NFVI. In *CSCN'17*, pages 187–192, 2017.

[19] S. Lal, T. Taleb, and A. Dutta. NFV: Security threats and best practices. *IEEE Communications Magazine*, 55(8):211–217, 2017.

[20] N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, and G. Varghese. Checking beliefs in dynamic networks. In *NSDI'15*, pages 499–512, 2015.

[21] T. Madi, Y. Jarraya, A. Alimohammadifar, S. Majumdar, Y. Wang, M. Pourzandi, L. Wang, and M. Debbabi. ISOTOP: Auditing virtual networks isolation across cloud layers in openstack. *ACM TOPS*, 22(1):1:1–1:35, 2018.

[22] H. Moens and F. De Turck. VNF-P: A model for efficient placement of virtualized network functions. In *CNSM'14*, pages 418–423, 2014.

[23] NIST. The NIST definition of cloud computing, 2011.

[24] Oasis. Topology and Orchestration Specification for Cloud Applications (TOSCA), 2013.

[25] ONAP. Open Network Automation Platform. Available at: https://www.onap.org/.

[26] OpenDaylight. OpenDaylight Project. Available at: https://www.opendaylight.org/.

[27] OpenDaylight. CVE-2018-1078: OpenDaylight - Insecure behavior in node reconciliation process, 2018.

[28] OpenStack. OpenStack. Available at: https://www.openstack.org/.

[29] OpenStack. Openstack security advisories. Available at: https://security.openstack.org/ossalist.html.

[30] OpenStack. Heavy reading study on CSPs and OpenStack, 2016.

[31] M. Pattaranantakul, R. He, A. Meddahi, and Z. Zhang. SecMANO: Towards network functions virtualization NFV based security management and orchestration. In *IEEE Trustcom/BigDataSE/ISPA*, pages 598–605, 2016.

[32] M. Pattaranantakul, R. He, Q. Song, Z. Zhang, and A. Meddahi. NFV security survey: From use case driven threat analysis to State-of-the-art countermeasures. *IEEE Communications Surveys & Tutorials*, 20(4):3330–3368, 2018.

[33] M.-K. Shin, Y. Choi, H. H. Kwak, S. Pack, M. Kang, and J.-Y. Choi. Verification for NFV-enabled network services. In *ICTC'15*, pages 810–815, 2015.

[34] S. W. Shin and G. Gu. Attacking software-defined networks: A first feasibility study. In *HotSDN'13*, pages 165–166, 2013.

[35] B. Tschaen, Y. Zhang, T. Benson, S. Banerjee, J. Lee, and J.-M. Kang. SFC-Checker: Checking the correct forwarding behavior of service function chaining. In *NFV-SDN'16*, pages 134–140, 2016.

[36] L. Xu, J. Huang, S. Hong, J. Zhang, and G. Gu. Attacking the brain: Races in the SDN control plane. In *USENIX Security'17*.

[37] Y. Xu, Y. Liu, R. Singh, and S. Tao. Identifying SDN state inconsistency in OpenStack. In *SOSR'15*, page 11, 2015.

[38] W. Yang and C. Fung. A survey on security in network functions virtualization. In *NetSoft'16*, pages 15–19, 2016.

[39] X. Zhang, Q. Li, J. Wu, and J. Yang. Generic and agile service function chain verification on cloud. In *IWQoS'17*, pages 1–10, 2017.