A Tenant-based Two-stage Approach to Auditing the Integrity of Virtual Network Function Chains Hosted on Third-Party Clouds

Momen Oqaily
CIISE, Concordia University
Montreal, Canada
m_oqaily@mail.concordia.ca

Suryadipta Majumdar
CIISE, Concordia University
Montreal, Canada
suryadipta.majumdar@concordia.ca

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

Mohammad Ekramul Kabir
CIISE, Concordia University
Montreal, Canada
m_kabi@encs.concordia.ca

Yosr Jarraya
Ericsson Security Research
Montreal, Canada
yosr.jarraya@ericsson.com

A S M Asadujjaman
CIISE, Concordia University
Montreal, Canada
asm.asadujjaman@mail.concordia.ca

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

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

ABSTRACT

There is a growing trend of hosting chains of Virtual Network Functions (VNFs) on third-party clouds for more cost-effective deployment. However, the multi-actor nature of such a deployment may allow a mismatch to silently arise between tenant-level specifications of VNF chains and their cloud provider-level deployment.

Most existing auditing approaches would face difficulties in identifying such an integrity breach. First, relying on the cloud provider may not be sufficient, since modifications made by a stealthy attacker may seem legitimate to the provider. Second, the tenant cannot directly perform the auditing due to limited access to the provider-level data. In addition, shipping such data to the tenant would incur prohibitive overhead and confidentiality concerns.

In this paper, we design a tenant-based, two-stage solution where the first stage leverages tenant-level side-channel information to identify suspected integrity breaches, and then second stage automatically identifies and anonymizes selected provider-level data for the tenant to verify the suspected breaches from the first stage. The key advantages of our solution are: (i) the first stage gives tenants more control and transparency (with the capability of identifying integrity breaches without the provider's assistance), and (ii) the second stage provides tenants higher accuracy (with the capability of rigorous verification based on provider-level data). Our solution is integrated into OpenStack/Tacker (a popular choice for NFV deployment), and its effectiveness is demonstrated via experiments (e.g., up to 90% accuracy with the first stage alone).

1 INTRODUCTION

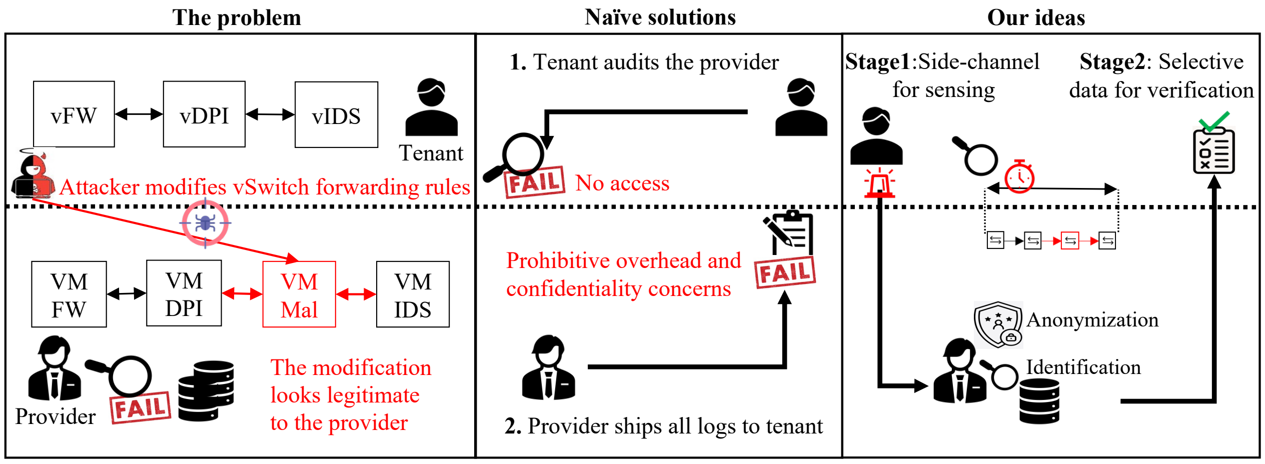
Network Functions Virtualization (NFV) enables on-demand and cost-effective deployment of network services as chains of VNFs on top of an existing cloud infrastructure [11]. A growing trend in NFV is to host the VNFs on third-party clouds for more cost-effective deployment [2, 18, 56], e.g., DISH Network is reportedly deploying its cloud-native 5G network in AWS Cloud [2], and VMware is enabling communications service providers to accelerate the deployment of their VNFs on its VMware Telco Cloud platform [56]. In such scenarios, the provider manages all the virtual and physical resources needed for deploying the specified network services by tenants as a chain of VNFs. Despite its obvious benefits to NFV tenants, the multi-actor nature of such deployment may lead to novel security threats. In particular, potential integrity breaches may silently arise due to unintentional misconfigurations [28] or malicious intents [49] to cause harmful inconsistencies between tenant specifications of VNF chains and their provider deployment.

Most existing security auditing[1] approaches for NFV (a detailed review of related works is in Section 6) would face difficulties against such integrity breaches mainly due to the following challenges. First, relying on the cloud provider (e.g., [26, 62]) may be impractical (as many providers may be reluctant to take the burden of conducting security auditing on behalf of their tenants) or insufficient (as providers typically do not understand tenant-level requirements, e.g., modifying compromised switch forwarding rules may seem wrong to the tenant, but legitimate to the provider). Second, relying on the tenant alone (e.g., [40]) may not be feasible, either, as the tenant typically has limited access to provider-level data, and shipping all such data to the tenant could incur prohibitive overhead and confidentiality concern. Based on comprehensive studies and existing literature [1, 21], it becomes evident that a novel solution is required to tackle those challenges.

High level motivating example, naive solutions, and our proposed ideas.. Figure 1 shows a motivating example to highlight the problem (left), limitations of naïve solutions (middle), and our main ideas (right).

The Problem: The left side of Figure 1 illustrates an example of an integrity breach and the limitation of provider-based auditing. Specifically, the top shows a chain of three VNFs (vFW, vDPI, and vIDS) specified by the tenant, and the bottom depicts that the cloud provider deploys those VNFs on three VMs. Suppose, due to an unintentional misconfiguration [58] or through the exploitation of compromised resources, an attacker (e.g., a co-located tenant) [28]

[1] We focus on *auditing* the integrity of VNF chains (i.e., matching deployment with specification) instead of *detecting* attacks that cause integrity breaches.



The problem
Naïve solutions
Our ideas
vFW
vDPI
vIDS
Tenant
Attacker modifies vSwitch forwarding rules
VM FW
VM DPI
VM Mal
VM IDS
Provider
FAIL
The modification looks legitimate to the provider
1. Tenant audits the provider
FAIL
No access
Prohibitive overhead and confidentiality concerns
FAIL
2. Provider ships all logs to tenant
Stage1:Side-channel for sensing
Stage2: Selective data for verification
Anonymization
Identification

Figure 1: Motivating example

can modify the switch forwarding rules to redirect traffic flowing through the chain to the malicious VM (i.e., VM Mal) into the chain. Since the chain now deviates from its specification, it allows the attacker to steal information, and hence, such a modification represents an integrity breach to the tenant. However, the modification may seem legitimate to the provider, as it is coming from a (seemingly) legitimate tenant, and therefore any provider-based auditing mechanism will likely miss such breaches. Therefore, relying on the provider alone may be insufficient due to its lack of understanding of tenant requirements, which motivates a tenant-based solution.

Tenant-based Naïve Solutions: The middle of Figure 1 illustrates the limitations of two naive solutions. In the first solution, the tenant typically has limited access to provider-level data (e.g., detailed logs or databases), and therefore it cannot directly audit the provider. In the second solution, even if the provider is willing to share, such a data transfer process may lead to prohibitive overhead and confidentiality concerns due to the multi-tenancy nature of NFV.

Our Ideas: The right side of Figure 1 illustrates our main ideas. Intuitively, we keep the auditing workload mostly on the tenant and only involve the provider to share selected anonymized data upon "sensing" (hence *NFVSense*) suspects of integrity breach. First, as illustrated by the clock and packet icons in the figure, the tenant identifies suspected integrity breaches based on tenant-level side-channel information. Second, it performs formal verification on selected provider-level data (which would be automatically identified and anonymized) to confirm (or reject) such suspects.

Specifically, we propose *NFVSense*, a tenant-based, two-stage approach to audit the integrity of VNF chains hosted on third-party clouds. In the first stage, *NFVSense* combines tenant-level side-channel information with machine learning (ML) techniques to identify potential suspects of integrity breaches. As such an approach will likely introduce false positives, the second stage automatically crafts selective data requests to the provider for each suspect from the first stage, anonymizes the data before returning it to the tenant, and finally performs rigorous verification based on such data to confirm (or reject) the suspects. We implement and integrate *NFVSense* into OpenStack/Tacker [33], a popular choice for NFV deployment. We evaluate the accuracy and efficiency of *NFVSense* through extensive experiments. In summary, our main contributions are as follows.

- To the best of our knowledge, *NFVSense* is the first tenant-based approach to auditing the integrity of VNF chains. The two-stage design of *NFVSense* leads to two key advantages: i) the first stage gives the tenant more control and transparency to audit the underlying deployment; ii) the second stage provides higher accuracy to the tenant who can perform rigorous verification based on selected provider-level data (which has been automatically identified and anonymized).
- To realize this two-stage design, *NFVSense* i) utilizes the tenant-level side channel information using Network Performance Tomography (NPT) and active probing techniques; ii) conducts verification on selected and anonymized provider-level data to audit the integrity of VNF chains hosted on third-party clouds.
- The applicability of *NFVSense* is demonstrated through its integration into OpenStack/Tacker, a popular cloud/NFV platform. Our experiments using several NFV datasets demonstrate the effectiveness of *NFVSense* (e.g., up to 90% of accuracy with the first stage alone).

The rest of the paper is organized as follows. Section 2 provides the preliminaries and our threat model. Section 3 details our methodology. Section 4 presents implementation details and experimental results. Section 5 discusses several aspects of *NFVSense*. Section 6 reviews the related works. Finally, Section 7 concludes the paper.

2 PRELIMINARIES

This section provides the essential preliminaries and defines our threat model.

Background on NFV. NFV enables the virtualization of network services and consists of two major abstraction levels as follows (according to the ETSI NFV reference architecture [11]):

(1) Tenant Level: This level is specified and managed by an NFV tenant who specifies its network services as a chain of several VNFs, e.g., virtual firewalls and IDS.
(2) Provider Level: This level is managed by a cloud infrastructure provider who instantiates the tenant's specifications of VNFs using both virtual resources, e.g., VMs, and physical resources, e.g., CPU and memory.

Rationale and Challenges in Using Performance-related Side-channels.

Our Rationale. We choose NFV performance-related side-channel information since, as Table 1 shows, there exist performance bottlenecks at different NFV abstraction levels, which can provide useful input for identifying any integrity breaches [3, 5, 7, 15], e.g., the impact from the hardware level can be more severe than the virtualization level [5]. The last column of Table 1 shows the parameters we choose to extract side-channel information at the corresponding level (the parameters selection are discussed in Section 3.2 and evaluated in the experiments Section 4).

Challenges. Using NFV performance side channels for identifying integrity breaches faces the following challenges, which will be addressed in Section 3:

- Limited Data Access: Since tenants cannot access the provider-level configuration, system-wide profiling tools to monitor hardware-based performance counters (e.g., cache-references or cross-system events, such as LinuxPerf [24] and OProfile [35]) cannot be utilized for our purpose.
- Performance Sensitivity and Probing Overhead: Network performance measurements in NFV can be highly sensitive to different factors (e.g., VNF's functionalities [55] and network workload [7]). Therefore, performance measurements must be repeated under different combinations of workloads and other parameters to ensure sufficient coverage.
- Fallacy of Profiling Standalone VNFs: The existing studies (e.g., [40, 54]) show that performance measurements of VNFs can change depending on their relative order in a chain. Also, adding or removing VNFs from a chain may affect the overall performance [54]. Consequently, establishing the performance profile of a chain can only be done by continuously measuring and analysing the performance of the chain as a whole after deployment [55].
- Other Performance Factors: There exist other factors that impact the network performance, such as the status of the VNF (e.g., active/passive) which may affect I/O waiting and processing time. The network I/O overheads in the virtual switches (delivering traffic either between VNFs or between the VNF and external network) can affect the performance [55].

Table 1: Excerpt of existing NFV performance bottlenecks.

Level	Bottleneck	Impact	Parameter
Physical	Memory size, number of CPUs, disk operation and hypervisor type	Variation in packet processing time [5]	Unique gap in RTT
Virtual	I/O interrupts, number of ports, virtual switching	VNF performance degrades [15]	Probing window/ workload
Virtual resources	Stateful and stateless connections, VNF functionality, VNF image	Longer packets delivery time [7]	Connection type and VNF type

Threat Model. The basis of our work is the "trust but verify" principle behind most security auditing techniques. More specifically, we assume the cloud provider and its infrastructure are both trusted by the tenant, but the tenant may still be concerned about unintentional user mistakes (made by cloud operators working for the provider) or stealthy attacks (which evade detection by the provider). Therefore, our solution is not meant to replace existing security mechanisms (e.g., security auditing and attack detection) of the provider, but rather to give the tenant additional control through performing independent tenant-based auditing.

Under such assumptions, we focus on a specific class of *in-scope threats*, i.e., integrity breaches in the form of invisible (to the tenant) modifications to the provider-level deployment of VNF chains, e.g., injection of malicious resources [6, 60], traffic redirection to bypass VNFs such as firewalls [48], reduction in virtual and physical resources [61], etc. We assume such threats are not thwarted by the provider because they come from external attackers who exploit zero-day attacks, or from malicious insiders, or simply unintentional user mistakes or misconfigurations caused by cloud operators themselves.

The *out-of-scope threats* include any attacks that can breach the integrity of network services without affecting network performance at any NFV layer. Also, we do not consider attacks that are directly visible to a tenant (e.g., removing a VM and its corresponding VNF) and thus do not require our solution. Moreover, similar to most side channel-based solutions (e.g., [58, 59]), we do not consider truly malicious providers or powerful attackers who have full control over the VNFs and thus can tamper with the captured performance results or the VNFs running *NFVSense*, or who can launch adversarial attacks against our tool (e.g., compromise the VNF chain at deployment time, or tamper with the training process using malicious samples). Finally, our work focuses on identifying integrity breaches (the consequences) rather than detecting or preventing attacks (the causes).

3 NFVSENSE

This section presents the methodology of *NFVSense.*

3.1 Overview

As shown in Figure 2, *NFVSense* is comprised of the following two stages: Stage 1: *Identifying Suspect Breaches*, and Stage 2: *Selected Data Verification.*

Stage 1: Identifying Suspected Breaches. This stage is to identify suspected breaches by only using tenant-level side-channels information as follows.

1. *Information Gathering and Processing:* When the chain is up and running, *NFVSense* gathers tenant-level performance measurements of the deployed topology as side-channel information, such as Round Trip Time (RTT). Using network performance tomography and active probing, it first characterizes the RTT between all pairs of VNFs in a tenant chain. Then it processes the collected data (i.e., outlier detection and filtering) to prepare for profiling in the next step.
2. *Performance Profiling:* Based on the impacts on the performance measures of VNFs, *NFVSense* identifies breaches by profiling the processed performance information as follows. It first learns an *Identification Model* (i.e., a binary classification ML model) to identify normal behavior and abnormal behavior (resulted from integrity breaches) and then learns a *Classification Model* (i.e., a signature-based multi-class classification ML model) to classify different types of breaches based on their impact on performance.

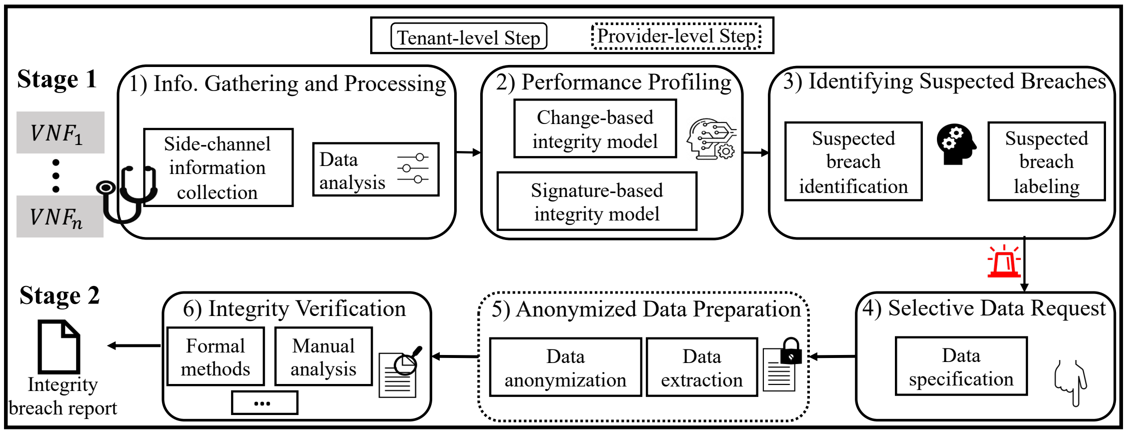

Tenant-level Step
Provider-level Step
Stage 1
VNF_1
VNF_n
1) Info. Gathering and Processing
Side-channel information collection
Data analysis
2) Performance Profiling
Change-based integrity model
Signature-based integrity model
3) Identifying Suspected Breaches
Suspected breach identification
Suspected breach labeling
Stage 2
4) Selective Data Request
Data specification
5) Anonymized Data Preparation
Data anonymization
Data extraction
6) Integrity Verification
Formal methods
Manual analysis
...
Integrity breach report

Figure 2: An overview of *NFVSense*.

3. *Identifying Suspected Breaches:* By implementing two ML models (from Step 2), *NFVSense* identifies suspected breaches in VNF chains and classifies them. Specifically, it first checks the current network performance of all VNF pairs in the chain with the *identification model* to identify any suspected breach, then determines its type by using the *classification model*. Finally, based on the type of the suspected breach, a tenant queries the provider for selected data (e.g., port forwarding rules for a set of VMs) for further verification in *Stage 2*. We will further elaborate *Stage 1* in Section 3.2.

Stage 2: Selected Data Verification. This stage is to conduct the verification on anonymized provider-level data that are specific to a suspected breach from Stage 1 and to confirm (or reject) it as follows.

1. *Selected Data Request:* To minimize the overhead for the provider, *NFVSense* only asks for selected provider-level data relevant to a suspected breach based on its source and type. For instance, if *NFVSense* suspects an integrity breach from a malicious VM injection (as in Figure 1), it only requests for the path forwarding logs of the VMs in the VNF chain corresponding to the suspected breach.
2. *Anonymized Data Preparation:* The main goal of this step is to provide property-preserved output that is suitable for auditing tasks. *NFVSense* leverages iCAT [37] to anonymize the selected provider-level data to ensure both the privacy requirements of the provider and the utility requirements of the tenant. Finally, the anonymized data is forwarded to the following steps.
3. *Data Verification:* To confirm an actual integrity breach from the suspects, *NFVSense* leverages existing verification solutions (using formal methods [36]) or performs a manual inspection on the anonymized data sent from the previous step. If the suspected breach is confirmed (e.g., VM port forwarding rules are modified), the tenant asks the provider for further action (e.g., mitigate the breaches). We will elaborate on *Stage 2* in Section 3.3.

3.2 Stage 1: Identifying Suspected Breaches

In the following, we elaborate on the steps of Stage 1.

Information Gathering and Processing. To train the *identification* and *classification* models (described in the next step), *NFVSense* has to assess both the normal (i.e., breach-free condition) and the abnormal behavior (i.e., breached conditions). Hence, in this step, from the tenant level, *NFVSense* gathers and processes round trip time (RTT) as side-channel information for a VNF chain deployed at the provider level. More specifically, we adopt the following two steps: i) to assess the normal behavior, *NFVSense* collects RTT values from $time_0$ when the service is created and hence assumed to be free of integrity breach; ii) to understand the abnormal behaviors, *NFVSense* simulates different attack scenarios to mimic different integrity breaches and collects respective RTT values.

To that end, *NFVSense* collects data from all VNF pairs after establishing an active probing connection between all VNF pairs. For instance, if n VNFs are in the chain, $(n * (n-1))/2$ active probing connections need to be established to cover all pairs of VNFs. For each connection, different parameters (i.e., probing rate and probing window, connection type) are also varied to profile the network behavior for different settings. After that, *NFVSense* collects RTT values in the probing responses accordingly and characterizes them by correlating the RTT values with the corresponding VNF chains. For this purpose, we adopt Network Performance Tomography (NPT) [10] of NFV which only requires VNF's usual packet forwarding behavior. Specifically, we integrate a performance measurement tool (e.g., IPerf [19]) into the VNF images. The pseudo-code of this step is described in *Algorithm 1*. In this pseudo-code, Lines 3-5 state to probe all the pairs of VNFs in the SFC according to the pre-defined probing rates and probing window, while Lines 6-8 find the number of hops and the RTT values between all VNF pairs and store the results with the probing in the NPT array.

Example 1. For the sake of illustration, we assume four NFV setups corresponding to four different integrity breaches scenarios as follows: i) $Scenario_0$: Initial integrity breach-free setup at $time_0$ before the attacker performs any modification; ii) $Scenario_1$: Malicious VM injection (marked as ① in Figure 3); iii) $Scenario_2$: Reduction in physical hosts (marked as ②) and iv) $Scenario_3$: Traffic redirection to a malicious VM (marked as ③). For each scenario, the NPT is collected between all pairs of VNFs in the chain while different parameters like probing workload, probing window, connection

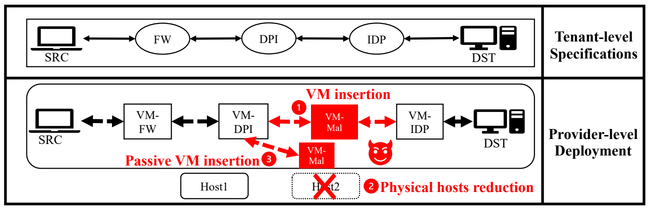


Figure 3: Example of different primitive scenarios for integrity breaches: i) $Scenario_1$: VM injection; ii) $Scenario_2$: Physical hosts reduction; and iii) $Scenario_3$: Passive VM injection.

type, etc. are also varied. Note that the methodology of *NFVSense* can be applied to other types of breaches as well, and also note that $Scenario_1$ to $Scenario_3$ are integrity breaches (i.e., the consequences) rather than attacks (hence *NFVSense* can identify such breaches regardless of the attacks causing the breaches).

Algorithm 1 Network Performance Tomography

```
1: Input: SubChai[], Rate[], Win[]
2: Output: NPT[]
3: for (s in range(Len(SubChai[])))  do
4:     for (r in range(Len(Rate[]))) do
5:         for (w in range(Len(Win[]))) do
6:             Hops = Len(SubChai[s])
7:             RTT = IPerf(SubChai[s], Rate[r], Win[w])
8:             NPT[s] = [SubChai[s], Hops, Rate[r], Win[w], RTT]
9:             Return (NPT[])
```

To process the collected data for profiling, *NFVSense*: i) identifies and filters out the outliers (e.g., extremely high RTT for the first few probing packets compared to the rest [55], or lost probing packets) in the network performance measurements using the Interquartile Range [9]; ii) merges all the features in the collected data, namely, the number of hops between source and destination of probes, VNF functionality type, probing rate and probing window; and iii) adds two ground-truth data labels to each probing record: *integrity breach* for indicating whether there is an integrity breach in a node pair and *breach type* (e.g., $Scenario_1$-$Scenario_3$ as mentioned in Example (1) from which those measurements are collected.

Performance Profiling. This step tends to profile the labeled RTT data by learning two separate ML models: a binary classification based *Identification Model* for identifying suspected breaches and a multi-class signature-based *Classification Model* for classifying the suspects. This separation of models: i) allows *NFVSense* to identify suspected integrity breaches beyond those example scenarios (i.e., $Scenario_0$-$Scenario_3$) as it mainly checks any deviation from the normal behavior; ii) improves the identification accuracy as it reduces the complexity of the trained models by reducing the dimension of the training dataset, and iii) improves the efficiency of *NFVSense* as the classification model is only triggered when there is a suspected breach. We elaborate on learning each of those models as follows:

- First, we learn the *identification model* to profile the normal behavior between all pairs of VNFs based on the integrity breaches-free setup. The objective of this model is to identify any deviation from the normal behavior (i.e., integrity breach-free setup) to identify integrity breaches.
- Second, we learn the *classification model* to learn the breach types by observing the patterns to extract each breach's signature. The objective is to differentiate between the suspected breaches based on their signature.

Identifying Suspected Breaches. This step implements the trained *identification* and *classification* ML models to identify and then classify suspected breaches.

- *NFVSense* identifies the suspected breaches in a VNF chain by using the *identification model*. Specifically, this model compares the similarity of the measured performance features from the probing packets with the learned performance profiles (i.e., the normal behavior), and any deviation from the normal behavior indicates a suspected breach. Note that to identify suspected breaches, instead of using a single probing packet, the model tests a stream of packets to compare them against the normal profile and hence attains a higher accuracy. *Algorithm 2* is the pseudo-code of this step. In this pseudo-code, line 3 defines variables to count the number of probe streams that are classified as normal and abnormal. Then Lines 4-6 are for probing all the pairs of VNFs in the SFC according to the predefined probing rates and probing window, and Lines 7-8 find the RTT values of the corresponding VNF pairs and parse them with other features. Finally, Lines 9-10 increase the predefined counter value based on the *identification model* results, while Lines 11-14 decide on the integrity breach status based on the counter value.

Algorithm 2 Integrity Breach Identification

```
1: Input: Sub-chains[], Rate[], Window[], BreachDetec()
2: Output: Decision[]
3: Normal=0, Abnormal=0
4: for (s in range(Len(Sub-Chains[])))  do
5:     for (r in range(Len(Rate[]))) do
6:         for (w in range(Len(Window[]))) do
7:             RTT = IPerf(Sub-chain[s], Rate[r], Window[w])
8:             Class=BreachDetec(RTT)
9:             IF (Class==1) {Normal++}
10:            ELSE {Abnormal++}
11:            IF(Normal $\geq$ Abnormal)
12:            Decision[j][i]=1
13:            ELSE
14:            Decision[j][i]=0
15:            Return(Decision[])
```

- After identifying a suspected breach, *NFVSense* classifies that breach by using the *classification model*. Specifically, using this model, *NFVSense* maps the deviated behaviors in the extracted integrity breach signatures into two levels: *physical resource level* and *virtual resource level*, where the integrity breaches related to changes in the physical resource level have distinct performance gaps compared to the breaches related to the virtual resource level, as discussed in Table 1.

Table 2: An example of event logs for different cloud services in OpenStack [32]

Level	Service	Project	Example Events
Physical	Computing	Nova	Add host, List Migrations
	Networking	Neutron	Create port, Delete subnet
	Switching	Open VSwitch	Add bundle, Delete bridge
Virtual	SFC	Tacker	Create VNF, Delete SFC

- Then *NFVSense* notifies the tenant about its findings to trigger Stage 2 (for further verification) to confirm or reject the decisions made in Stage 1.

Example 2. For $Scenario_2$ (refer to Figure 3), *NFVSense* identifies the suspected breaches as follows. As the packets delivery gaps between VM_{DPI} and VM_{IDP} are expected to be lower compared to the normal behavior since the virtual switch level consumes less time to deliver the traffic between these two VMs running on the same physical hosts. Consequently, this leads the *identification model* to raise an alarm of a suspected breach. Then using the *classification model*, *NFVSense* maps the suspected breach with the pre-defined breaches (i.e., $Scenario_1$ to $Scenario_3$ as described in Example 1). More specifically, the RTT values would be lower after the *reduction in physical host* ($Scenario_2$) compared to the *malicious VM injection* ($Scenario_1$), where the virtual switch level forwards the traffic to an extra VM (i.e., a malicious VM) in a VNF chain. On the other hand, $Scenario_2$ has higher RTT values compared to the *traffic redirection* breach ($Scenario_3$), where the virtual switch duplicates the traffic to the injected passive VM, VM_{Mal}. Hence, by this mapping, *NFVSense* determines the suspected breach as a potential *reduction in physical host* attack.

3.3 Stage 2: Selected Data Verification

In the following, we elaborate on the steps of Stage 2.

Selected Data Request. This step is to request the provider for specific data related to the suspected breach. More specifically, *NFVSense* considers the decision made by Stage 1 (i.e., the identified and classified suspected breaches) to determine which provider-level data is required to verify those decisions, and requests the provider to send those related logs accordingly. If the location of the suspected breach is identified and traced to a specific pair of VNFs based on the built performance profiles and measured performance metrics, then *NFVSense* requests only the provider-level logs which correspond to those VNF deployments at the tenant level. On the other hand, if the class of the suspected breach is not identified (i.e., a breach is suspected but cannot be classified), *NFVSense* queries the log of all cloud services for a specific time range (i.e., $T_{Abnormal} - T_{Normal}$), where $T_{Abnormal}$ is the time when the suspected breach is identified, and T_{Normal} is the last time when there was no breach. The details of locating the suspected breach are described in *Algorithm 3* as follows. Lines 3-4 of the pseudo-code define two variables to locate the breach, while Lines 5-6 go through the list of all VNF pairs in the SFC. Lines 7-11 identify the VNF pair where the number of hubs is equal to zero, and the integrity breach is identified.

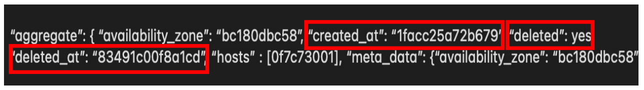
Algorithm 3 Integrity Breach Locator

```
1:  Input: Sub-chains[], Decision[][]
2:  Output: Loc[]
3:  Loc[2]=NULL
4:  NumofHops=1
5:  for (s in range(Len(Sub-Chains[]))) do
6:      for (r in range(Len(Sub-Chains[]))) do
7:          IF( NumofHops!=0)
8:          IF (Decision[j][i]==1 && Decision[j+1 [i+1]==0)
9:          Loc[0]=VNFlist[i]
10:         Loc[1]=VNFlist[j]
11:         NumofHops=j;
12:         Return(Loc[])
```

Example 3. Table 2 shows an example (from OpenStack [32], a popular cloud platform) of event logs specific to different services (e.g., computing, networking, etc.) at the provider level; part of which can be requested during this step for any suspected breach. Due to a suspected breach related to $Scenario_2$, between the VNF pairs VNF_x and VNF_y, the tenant will query the provider for the ports forwarding logs of the corresponding VMs VM_x and VM_y from the *Neutron* service in OpenStack. Similarly, other selected provider-level data can be requested for other types of breaches such as OpenVswitch and Nova logs for $Scenario_1$ and $Scenario_3$.

Anonymized Data Preparation. Though the previous step can minimize the overhead of sending all logs to the tenant, that can not ensure the privacy and confidentiality of both the provider and other tenants. For example, important network configuration information (i.e., potential bottlenecks and topology of the network) may be inferred from the logs and subsequently exploited by adversaries [37]. To address this concern, in this step, *NFVSense* leverages iCAT [37] to anonymize the logs, while this tool meets both the provider's privacy requirements along with the tenant's utility requirements. To that end, iCAT leverages natural language processing techniques to translate those requirements and find the most suitable anonymization primitives to meet the requirements. Hence the confidentiality concern of the provider in sharing data is addressed, while auditing at the tenant level is enabled.

Example 4. To verify the suspected breach in $Scenario_2$, the tenant's utility requirement is *the Sequence of the events must be preserved* (i.e., the events must be persevered chronologically in the anonymized output). On the other hand, the provider's privacy requirement is *all tenants' network topologies should be unidentifiable*. Considering those requirements, the leveraged anonymization tool first determines the suitable anonymization primitives (e.g., timestamp shifting, IPs truncation, etc.) and then produces an anonymized output (i.e., ports forwarding logs).



```
"aggregate": { "availability_zone": "bc180dbc58", "created_at": "1facc25a72b679" "deleted": yes
"deleted_at": "83491c00f8a1cd", "hosts" : [0f7c73001], "meta_data": {"availability_zone": "bc180dbc58"
```

Figure 4: A snippet of anonymized Nova logs related to an integrity breach.

Integrity Verification. *NFVSense* can leverage existing auditing tools or perform a manual inspection on the received anonymized data to validate the suspected breaches identified in Stage 1. To that end, we utilize a formal security verification tool, NFVGuard [36] to formally verify the alerts generated by *NFVSense.* NFVGuard verifies chain integrity using the formal method in two steps. First, VNF Forwarding Graph (i.e., VNFFG, a feature used to orchestrate and manage traffic through VNFs) configuration consistency properties are applied to verify consistency between the VNFFGs specification uploaded by the tenants (such as the size of VNFFG and the sequences of VNFs) and their corresponding implementation. Second, The VNFFG configuration consistency properties are applied to verify the consistency between the created SFCs and hardware implementation. If either property does not hold in the deployed NFV system, it means that there is a breach to the integrity of the underlying deployment. In practice, the tenant admins can further manually inspect the logs to verify the integrity of the SFC.

Example 5. In the case of $Scenario_2$, the tenant will look for system event logs in the computing service, *Nova* in OpenStack, resulting from deleting a physical server. Figure 4 shows a snippet of the Nova anonymized logs for the *deleted host* event. From this log entry (highlighted in red), *NFVSense* confirms the breach from the host creation and deletion time (i.e., *created-at* and *deleted-at*) and the success of the deletion operation (i.e., *deleted:Yes*).

4 IMPLEMENTATION AND EXPERIMENTS

This section describes the implementation and experiment results.

4.1 Implementation

We implement and integrate *NFVSense* into OpenStack/Tacker [33], which is a popular NFV management platform. Specifically, *NFVSense* is implemented in Python, by leveraging the NumPy and Pandas [31] libraries for gathering NFV performance measurements (RTT for this case), and scikit-learn [43] for implementing ML models. Finally, we use our designed anonymization tool iCAT [37] to generate the anonymized proofs for verification. All the VNFs are built based on the official Ubuntu 18.04 cloud image customized by Canonical to run on public clouds [52]. We follow the model stated in [55] to configure our VNFs and use Open vSwitch [41] to manage the connectivity between them.

4.2 Experimental Environment

This section describes our experiments datasets and environment.

Dataset Description. We consider both real and synthetic data to evaluate *NFVSense* effectiveness. We collect and generate four datasets, *DS*0-*DS*3, for four scenarios (as shown in Figure 3), respectively. In summary, a dataset of size 4.5 GB corresponding to three different integrity breach scenarios are used, while over 136K probing requests are generated on four scenarios, where four VNF images and two physical hosts are used. During probing, we vary the traffic to four different workloads, the probing period to three different windows with two connection types (i.e., stateful and stateless). In the following, we briefly explain how data is collected.

- *Real Data:* We implement an NFV testbed to collect real data from the NFV stack. We use Python scripts to automatically generate TOSCA templates in order to deploy NFV entities, such as VNFs and VNFFGs. In our implementation, we deploy three different VNF images for widely-used network services: (i) Tcpdump [47], a network data packet analyzer with the default configuration, (ii) Snort [45], a network intrusion, detection, and prevention system configured with the default rules, and (iii) IPtables [30], a firewall program for Linux configured with 50 rules (in a way that only the last rule matches with our probing traffic and the VNF checks all rules). On each VNF, we implement NPT for characterizing NFV performance measurements using IPerf3 [19], a tool that can produce standardized performance measurements for any network, and Bash scripting to perform active probing between all pairs of VNFs in a chain while varying different probing parameters (i.e., probing rate, probing window, connection type). For each probing packet, we log the source and destination nodes, the number of hops between them, and the resulting RTT.
- *Synthetic Data:* We provide an option in *NFVSense* that is only enabled when a tenant cannot gather a sufficient amount of training data for ML models due to the higher cost of continuously obtaining labeled data (e.g., via manual efforts) [50], or the overhead of substantial probing traffic using NPT [40]. This alternative option generates synthetic training data using generative adversarial network (GAN) models [46]. In contrast to most existing works that apply GAN to deceive systems (e.g., IDS [22], steganalyser [57]), we leverage GAN, similarly as in [17, 44], to generate synthetic network traffic data that would be similar to the real flow data obtained through active probing. More specifically, by generating realistic data, GAN solves the challenge of acquiring "labeled data" for training our ML model [17] along with ensuring a negligible overhead compared to the imposed overhead by the NPT step. To evaluate the effectiveness of GAN in generating synthetic data, we compare the performance of *NFVSense* with and without data synthesis in the next section. Note that the synthetic data generated by GAN is proposed as an auxiliary source of data, only when real data are scarce.

For the experimental setup, we deploy 31 types of VNFs (e.g., with autoscaling policies, dedicated subnet, floating IPs, etc.), and seven variations of VNFFGs to create sufficient diversity in the corresponding event sequences. We also randomize a few important parameters in the template description: 1) the number of virtual network ports per VNF, 2) the number of deployment units per VNF, 3) the node Flavor specification for each VDU, 4) the number of VNFs for each Network Forwarding Path (NFP), 5) the order of VNFs for each NFP, 6) the flow-classifier criteria for each NFP, and 7) the number of NFPs for each VNFFG.

Environment Setup. All the experiments are conducted on SuperServer 6029P-WTR running the Ubuntu 18.04 operating system equipped with Intel(R) Xeon(R) Bronze 3104CPU 1.70GHz and 128GB of RAM without GPUs. Moreover, the NFV stack implementation to collect the real datasets follows the four scenarios (shown in Figure 3). To evaluate the accuracy (AUC) of *NFVSense*, we implement, train, and deploy different ML models: Decision Tree (DT),

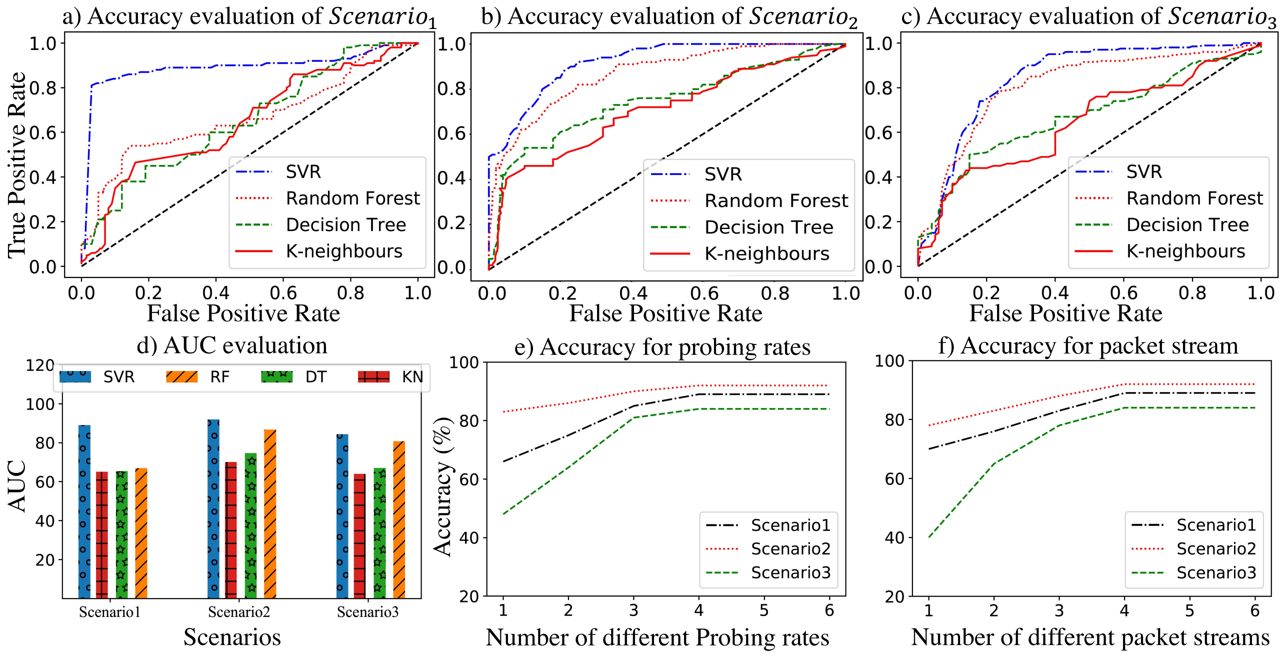


Figure 5: *NFVSense* Stage 1 evaluation.

Random Forest (RF), k-nearest neighbors (kNN), and Support Vector Regression (SVR).

4.3 Experimental Results

We evaluate the accuracy (AUC) of Stage 1 for identifying the suspected breaches, and the corresponding resource consumption. Afterward, we evaluate the performance of Stage 2 in confirming (or rejecting) the findings of Stage 1 (i.e., suspected breaches) by using selective data.

Evaluation of Stage 1 in Identifying Suspected Breaches. The first set of experiments (Figure 5) is to evaluate the accuracy of Stage 1 in identifying suspected breaches. We use the dataset without a breach, $DS0$, to train the *identification model* to learn the normal behavior, and the three datasets having breaches ($DS1$-$DS3$) for validating and testing the model (10% of data are for validating and 90% of them are for testing). To avoid overfitting, we run our experiments up to 100 epochs. We also use an early stopping mechanism, while the trigger parameter is set to *early-stop-threshold* = 5, which means that during the training period, the accuracy is calculated after each epoch, and if there is no improvement in the accuracy of the validation set compared to the training set for consecutive five epochs, the training process stops. Finally, we set the length of packet trains and the number of different probing rates to four (a study on the selection of these parameters is shown in Figures 5.e and 5.f). Figures 5.(a-c) show the identification accuracy of Stage 1 for different integrity breach scenarios and different ML algorithms.

Among all four ML algorithms, the SVR achieves the best accuracy in all cases, as it tries to fit the best line within a threshold value that separates the normal behavior from the deviated one. The accuracy is also dependent on the integrity breach scenario as depicted in Figure 5.d. For $Scenario_1$ (i.e., malicious VM injection), the AUC value is about 89%, while for $Scenario_2$ (i.e., one physical host is deployed instead of two), the *AUC* value increases up to 91.92%. For $Scenario_2$, SVR achieves the best accuracy due to having the highest impact on the measured performance [5]. Finally, for $Scenario_3$ (an adversary inserts passive and malicious VM to the chain), Stage 1 achieves the AUC of 84.3% as depicted in Figure 5.c. The accuracy of the results related to this scenario is the lowest due to the performance profiles at low probing rates being very close to the scenario without a breach (i.e., normal behavior), and the effect of this scenario only appears at higher probing rates. This is because the observed delay on the end-to-end service results from the delay caused by the OvS duplicating the traffic to the passive malicious VNF, and it only creates a distinct difference when the amount of traffic to be duplicated is higher. Figures 5.e and 5.f show the accuracy of identifying the suspected breaches by the SVR model while varying the performance profiling parameters, i.e., the length of a packet stream and the probing rates.

During measurements for each scenario, we fix one parameter and vary the other. For varying the number of probing rates, we progressively increase the number of used probing rates (in KB/s), i.e., 64, (64,128), (64, 128, 256), and so on. We observe that the accuracy increases for all scenarios when we increase the number of probing rates and the length of the packet stream. When both the length of a packet stream and the number of probing rates are set to four, the accuracy becomes almost saturated (i.e., around 95% for $Scenario_2$). As a result, the model decision is made to identify suspected breaches based on the results of four probing samples that cover four different probing rates. Also, even though the performance profile of $Scenario_3$ is the closest to the scenario without a breach (as discussed above), our model successfully identifies it

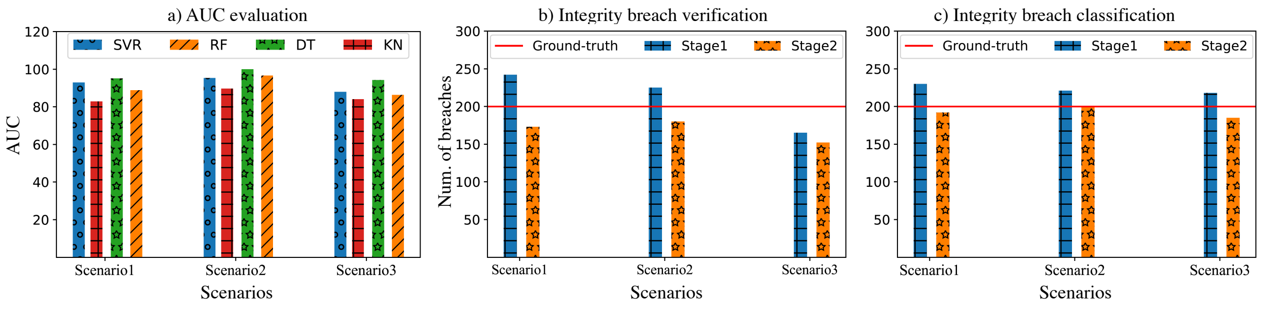


Figure 6: a) Stage 1 evaluation for integrity breach classification, b) evaluation of Stage 2 performance in correctly identifying integrity breaches from Stage 1, and c) evaluation of Stage 2 performance in correctly classifying the integrity breaches from Stage 1.

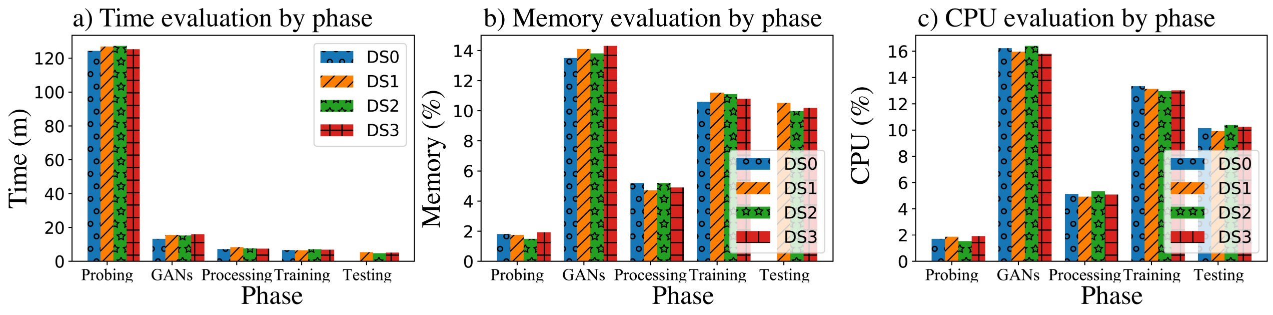


Figure 7: *NFVSense* resource consumption evaluation.

and the accuracy increases significantly when increasing both the number of probing rates and the length of the packet stream.

Evaluation of Stage 1 in Classifying Suspected Breaches. The second set of experiments (Figure 6.a) is to evaluate the accuracy of Stage 1 in classifying the suspected breaches. To that end, we use 20% of the data to train the signature-based *classification model*, while 5% for validation, and 75% for testing. Similar to the previous set of experiments, we implement four ML algorithms and run our experiments up to 100 epochs with an early stopping mechanism (i.e., early-stop-threshold = 5). Figure 6.a shows that the Decision Tree (DT) model achieves the best AUC compared to other ML models for all three scenarios as it reduces the variance by creating its predictions based on the training data. The DT model achieves the AUC value of 95.1% for $Scenario_1$, 100% for $Scenario_2$, and 94.2% for $Scenario_3$. Note that Stage 1 attains the highest accuracy for $Scenario_2$ as there is a distinct gap between the data points of this scenario compared to other scenarios.

Evaluation of Stage 2 in Verifying Selective Data. Figures 6.b and 6.c show the outcomes of Stage 2 in verifying the suspected breaches. The logs are anonymized by using iCAT [37] (an interactive and customizable anonymization tool as discussed in Section 3.3), while Nova logs are used to evaluate $Scenario_2$ (i.e., physical hosts reduction), and Neutron and OpenvSwitch logs are used to evaluate $Scenario_1$ and $Scenario_3$ (i.e., active and passive VM insertion). Figure 6.b shows the number of breaches identified at two stages (separately) compared to the ground truth. We can see that Stage 1 identifies a slightly larger number of breaches compared to the ground truth (i.e., 242 and 225 for $Scenario_1$ and $Scenario_2$, respectively), which denotes false positives. However, due to Stage 2, we can successfully filter out those false positives from Stage 1. Thus, in $Scenario_1$ and $Scenario_2$, 173 and 183 (respectively) of the suspects are identified by Stage 2 to correspond to real integrity breaches. In $Scenario_3$, Stage 1 identifies a slightly lower number of breaches compared to the ground truth (i.e., 165 for $Scenario_3$), which denotes few missed integrity breaches. Stage 2 filters out the false positives and identifies 152 of those suspects corresponding to real ones. Figure 6.c shows the number of classified suspected breaches at two stages (separately) compared to the ground truth. In this context, even though some false positives are generated by Stage 1, Stage 2 filters the false positives and shows that there are only a few misses by Stage 1, which joins the accuracy obtained in the previous evaluation experiment. In summary, Stage 2 can successfully filter out false positives generated by Stage 1. Furthermore, the total number of integrity breaches reported by Stage 2 indicates that the identification and classification of suspected breaches by Stage 1 are fairly accurate concerning the ground truth (e.g., for $Scenario_2$, 183 out of 200 breaches are identified by Stage 1). Note that, the total number of false negative decisions is also significantly low (e.g., for $Scenario_2$, the number of false negatives is 17 out of 200 breaches), but *NFVSense* cannot verify this false negative decision using its Stage 2.

Evaluation of Resource Consumption. This set of experiments (Figure 7) is to evaluate the time, CPU, and memory consumption by Stage 1. Figure 7.a shows the time consumption of Stage 1 by

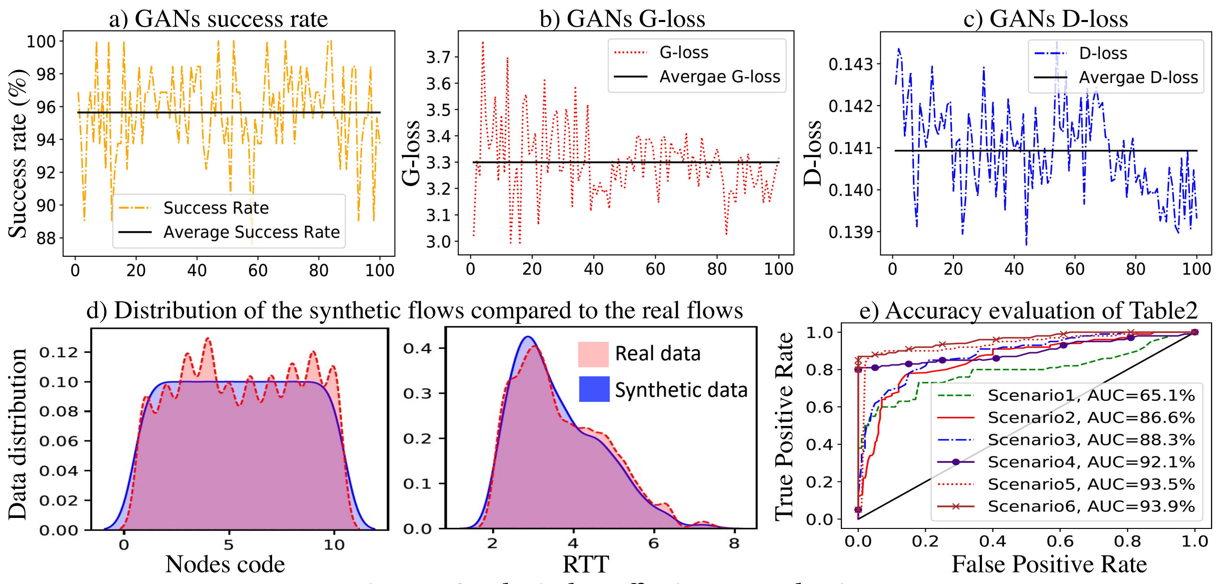


Figure 8: Synthetic data effectiveness evaluation.

Table 3: Datasets partitioning for synthetic data validation.

Source	$Setup_1$		$Setup_2$		$Setup_3$		$Setup_4$		$Setup_5$		$Setup_6$	
Scenario 0/1	Real	Synth.	Real	Synth.	Real	Synth.	Real	Synth.	Real	Synth.	Real	Synth.
	2k	0	2k	2k	4k	0	4k	20k	4k	30k	4k	40k

phase. Note that the scenario without a breach dataset (*DS*0) has no testing time as it is only used for training the suspected breach identification model, and this evaluation is based on the ML models that achieve the best accuracy (i.e., SVR for *identification model* and DT for *classification model*). This figure depicts that the most costly operation by Stage 1 is to characterize network performance data using NPT, where we collect performance statistics about the NFV deployment to train the *identification model*. This observation indicates that using GAN-generated synthetic data may reduce the data collection time. On the other hand, Figures 7.b and 7.c show each dataset's memory and CPU consumption, respectively. Though the probing step consumes a higher time, it consumes the lowest CPU and memory resources. Also, synthesizing by GANs is the most expensive step in terms of CPU and memory consumption. However, the CPU consumption (17%) and the memory consumption (15%) for GANs are still not very high and might be affordable.

Synthetic Data Effectiveness. The final set of experiments (Figure 8) is to measure the effectiveness of our synthetically generated data (using GAN). To that end, we evaluate the success rate of synthetic data that is indistinguishable from the real one, and the losses of the discriminator and the generator models, and conduct a comparison between the real and synthetic data distributions. Figure 8.a depicts a summary of the effectiveness evaluation of GAN for 100 epochs, while on average, the success rate of the synthetic data is around 95.7%. On the other hand, the average generator and discriminator losses are only 3.3% and 0.14%, as shown in Figures 8.b and 8.c, respectively. Such high accuracy indicates a successful generation of indistinguishable synthetic data, while Figure 8.d also illustrates the similarity of the distribution of generated data (red-shaded areas) with the real one (blue-shaded areas). Additionally, in Figure 8.e, we evaluate the SVR model only on one integrity breach dataset, *DS*2 (i.e., physical host reduction scenario), to evaluate the performance of the synthetic data in the identification accuracy of the *NFVSense* using various partitioning setups between synthetic and real data (as shown in Table 3). The figure demonstrates that the synthetic data provides almost the same accuracy as the real one. As an example, despite having different ratios of the real and synthetic data, the two equal-sized datasets (i.e., $Setup_2$ and $Setup_3$ in Table 3) show almost the same identification accuracy. The identification accuracy also increases with the increased amount of training data and hence, augmenting the real data (i.e., 4K) with 20K of synthetic data ($Setup_4$ in Table 3) increases the accuracy significantly (92.1%), while this is 93.5% for $Setup_5$. Thus, these experimental results indicate that utilizing synthetic data where collecting large-scale real data is infeasible might be an alternative for the *NFVSense* users.

5 DISCUSSIONS

Auditing over Specific Attack Detection. Unlike crypto-based solutions (e.g., AuditBox [25]), which are more specific and dependant on specific attacks/intrusions, NFVSense aims to audit integrity breaches in VNF chains, which could have been caused by different attacks. However, it can only classify the type of breaches in Stage 1 when the signature of such breaches is already available in the models. To accommodate the retraining of the model with new breach types, NFVSense would require human intervention to manually label the data with the new signatures.

Requirements for Retraining Our Models. *NFVSense*'s ML models are not specific to any topology, as they are trained based on the pairs of hops and not on the entire topology. Therefore, the same model can be used for different topologies as long as the same VNF images are used. On the other hand, the ML models require retraining when the number of VNFs in a chain is increased.

Various Options for Stage 2 Verification. The design of *NFVSense*, particularly its Stage 2, is as such that a wide range of existing tools (e.g., VS [16]) can be leveraged for the verification step. In this paper, we mainly leveraged formal methods (e.g., NFVGuard [36]) and manual inspection as examples. Nonetheless, other verification tools can be leveraged.

Compatibility with Other NFV Platforms. *NFVSense* is designed based on the generic NFV architecture and deployment model [11], and all its modules are mostly platform-agnostic (except the learned models). Therefore, *NFVSense* is a generic solution, which can consequently be adapted to other NFV platforms (e.g., OSM [38] and OPNFV [34]), by learning platform-specific models.

Online Training. The ML model of *NFVSense* is trained at deployment time (i.e, $time_0$) where the NFV setup is first created and thus assumed to be attack free. Therefore, such ML models are specific to the actual setup and runtime configurations of the NFV. Unlike existing works relying on offline training, we do not need to test all possible combinations of VNFs (i.e., considering all possible types of hardware and topologies in advance). However, once legitimate changes are made to the topology/hardware, the ML models have to be retrained. Hence, we plan to consider online training for the ML models as a future direction to support dynamic changes.

Table 4: Comparing our work with existing solutions.

Work	Run-time	Dynamic	Tenant-based	Path Modif.	Blackbox
vSFC [61]	✓	✓	x	✓	x
FlowCloak [6]	✓	✓	x	✓	x
AuditBox [25]	✓	✓	x	✓	x
SFC-Checker [51]	x	x	x	✓	x
EasyOrch. [53]	x	x	N/A	x	x
FlowTags [12]	✓	x	✓	✓	x
ChainGuard [14]	✓	x	x	✓	x
NFVSense	✓	✓	✓	✓	✓

6 RELATED WORK

This section reviews existing related works and highlights their limitations.

NFV-based Security Solutions. Most of the existing solutions (e.g., [6, 12, 14, 25, 51, 61]), to audit integrity breaches in NFV rely on the provider-level data. FlowCloak [6] and vSFC [61] identifies VNF chain violations (e.g., path non-compliance and packet injection attacks). Similarly, SFC-checker [51] and ChainGuard [14], audit the forwarding behavior of the VNF chains in a network service. AuditBox [25] provides continuous assurance that packets follow the formally specified policy-mandated path using formal models. On the other hand, several works (e.g., [13, 26, 55]) focus on the performance and functionality of NFV network services. Unlike those works, NFVSense provides a tenant-based, two-stage approach which does not fully rely on the provider.

Performance-based Identification Solutions. There exist several performance-based identification solutions (e.g., [5, 20, 27]) for virtualized environments. For instance, Koh et al. [20] study the inter-VM interference performance characteristics by collecting runtime performance measures. Whereas, Mei et al. [27] study the network performance in a virtualized cloud environment while varying the network I/O workloads. The authors in [5] analyze the performance of virtual machines in an IaaS cloud environment to infer the network topology. There are a few other works (e.g., [29, 39, 42]) that analyze VNFs to find performance issues in NFV. Unlike those solutions, *NFVSense*, in Stage 1, uses the performance characteristics to identify suspected integrity breaches.

Network Tomography Solutions. There are several works (e.g., [4, 8, 19, 23]) that use network tomography for characterizing network behavior. Among them, traceroute [23] and iperf [19] use the node-to-node tomography approach to measure performance metrics (e.g., delay, loss rate) of a specific link directly through sending probing traffic from source to destination. On the other hand, using the end-to-end tomography approach, Chen et al. [8] calculate unknown link variables and Arifler et al. [4] identify the congested links. Similar to existing works, NFVSense uses network tomography to collect performance characteristics data, but uniquely for identifying integrity breaches in NFV.

Table 4 summarizes the comparison between the most recent NFV security auditing works and *NFVSense*. Our comparison is based on different properties that those solutions support. Specifically, the detection coverage (i.e., run-time or stateful), the auditing capabilities (i.e., dynamic or static), the deployment level (i.e., tenant-side or provider-side), the integrity breaches, and the provider level accessibility (i.e., blackbox or whitebox).

7 CONCLUSION

With the widespread adoption of VNF chains for various network services in NFV, their proper deployments on a third-party cloud are very often questionable due to unintentional misconfigurations or malicious attacks. As a result, it is essential to audit the integrity of those VNF chains in NFV. In this paper, we proposed a new tenant-based, two-stage solution, namely, *NFVSense*, that tackled the existing challenges in auditing the integrity of VNF chains hosted on third-party clouds. Specifically, its first stage identified suspected integrity breaches from tenant-level side-channel information (i.e., RTT) and its second stage verified those suspects by identifying and anonymizing selected provider-level data for the tenant to confirm actual breaches. We implemented and integrated *NFVSense* into OpenStack/Tacker and conducted extensive experiments to demonstrate its efficiency and accuracy. However, there still exist a few limitations in *NFVSense*. For instance, currently, *NFVSense* can only sense and verify integrity breaches after the fact. Preventing breaches will be considered as future work. Another future direction for *NFVSense* is to modify the design of the learning phase to ensure that *NFVSense* is enriched with real-time data (e.g., online learning) that reflects the changing state of a running network service.

Acknowledgment. We thank the reviewers for their valuable comments. This work was supported by the Natural Sciences and Engineering Research Council of Canada and Ericsson Canada under the Industrial Research Chair in SDN/NFV Security and the Canada Foundation for Innovation under JELF Project 38599.

REFERENCES

[1] Jay Aikat, Aditya Akella, Jeffrey S Chase, Ari Juels, Michael K Reiter, Thomas Ristenpart, Vyas Sekar, and Michael Swift. 2017. Rethinking security in the era of cloud computing. *IEEE S&P* (2017).

[2] Ammar Latif, Ash Khamas, Sundeep Goswami, Vara Prasad Talari, and Dr Young Jung. 2022. Telco Meets AWS Cloud: Deploying DISH's 5G Network in AWS Cloud. Available at: https://aws.amazon.com/blogs/industries/telco-meets-aws-cloud-deploying-dishs-5g-network-in-aws-cloud/.

[3] Marco Anisetti, Claudio A Ardagna, Filippo Gaudenzi, Ernesto Damiani, Nicla Diomede, and Patrizio Tufarolo. 2018. Moon cloud: a cloud platform for ICT security governance. In *2018 IEEE (GLOBECOM)*. IEEE.

[4] Dogu Arifler, Gustavo de Veciana, and Brian L Evans. 2004. Network tomography based on flow level measurements. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2. IEEE, ii–437.

[5] Dominic Battré, Natalia Frejnik, Siddhant Goel, Odej Kao, and Daniel Warneke. 2011. Inferring network Topologies in Infrastructure as a Service Cloud. In *CCGRID*. IEEE.

[6] Kai Bu, Yutian Yang, Zixuan Guo, Yuanyuan Yang, Xing Li, and Shigeng Zhang. 2018. FlowCloak: Defeating middlebox-bypass attacks in software-defined networking. In *IEEE INFOCOM*. IEEE.

[7] Monchai Bunyakitanon, Aloizio Pereira da Silva, Xenofon Vasilakos, Reza Nejabati, and Dimitra Simeonidou. 2020. Auto-3P: An Autonomous VNF Performance Prediction & Placement Framework based on machine learning. *CN* (2020).

[8] Aiyou Chen, Jin Cao, and Tian Bu. 2010. Network tomography: Identifiability and Fourier domain estimation. *IEEE TSP* (2010).

[9] Frederik Michel Dekking, Cornelis Kraaikamp, Hendrik Paul Lopuhaä, and Ludolf Erwin Meester. 2005. *A Modern Introduction to Probability and Statistics: Understanding why and how.* Springer Science & Business Media.

[10] Nick Duffield. 2006. Network tomography of binary network performance characteristics. *IEEE TIT* 52, 12 (2006), 5373–5388.

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

[12] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. 2014. Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags. In *USENIX NSDI*.

[13] Seyed Kaveh Fayazbakhsh, Michael K Reiter, and Vyas Sekar. 2013. Verifiable network function outsourcing: requirements, challenges, and roadmap. In *MNFV*.

[14] Johannes M Flittner, Matthias Scheuermann. 2017. ChainGuard: Controller-independent verification of service function chaining in cloud computing. In *IEEE SDN*.

[15] Jinho Hwang, K K_ Ramakrishnan, and Timothy Wood. 2015. NetVM: High performance and flexible networking using virtualization on commodity platforms. *IEEE TNSM* (2015).

[16] Fariha Tasmin Jaigirdar, Carsten Rudolph, and Chris Bain. 2021. Risk and compliance in IoT-health data propagation: A security-aware provenance based approach. In *ICDH*.

[17] Steve TK Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, and Bimal Viswanath. 2020. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *IEEE S&P*.

[18] Peipei Jiang, Qian Wang, Muqi Huang, Cong Wang, Qi Li, Chao Shen, and Kui Ren. 2021. Building In-the-Cloud Network Functions: Security and Privacy Challenges. *Proc. IEEE* 109, 12 (2021), 1888–1919. https://doi.org/10.1109/JPROC.2021.3127277

[19] Jon Dugan et al. 2021. active measurements of the maximum achievable bandwidth on IP networks. Available at: https://iperf.fr/iperf-doc.php.

[20] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. 2007. An analysis of performance interference effects in virtual environments. In *ISPASS*.

[21] Sudershan Lakshmanan Thirunavukkarasu. Master's thesis 2020. Caught-in-Translation: Detecting Cross-level Inconsistency Attacks in NFV.

[22] Zilong Lin, Yong Shi, and Zhi Xue. 2018. IDSGAN: Generative adversarial networks for attack generation against intrusion detection. *arXiv preprint arXiv:1809.02077* (2018).

[23] Linux. 2021. Traceroute. Available at: t.ly/tq0k.

[24] LinuxPerf. 2021. Profiling with performance counters. Available at: t.ly/miMd.

[25] Guyue Liu, Hugo Sadok, Anne Kohlbrenner, Bryan Parno, Vyas Sekar, and Justine Sherry. 2021. Don't Yank My Chain: Auditable {NF} Service Chaining. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*.

[26] Guido Marchetto, Riccardo Sisto, Jalolliddin Yusupov, and Adlen Ksentini. 2018. Virtual network embedding with formal reachability assurance. In *IEEE CNSM*.

[27] Yiduo Mei, Ling Liu, Xing Pu, Sankaran Sivathanu, and Xiaoshe Dong. 2011. Performance analysis of network I/O workloads in virtualized data centers. *IEEE TSC* 6, 1 (2011), 48–63.

[28] Vaishnavi Moorthy, Revathi Venkataraman, and T Rama Rao. 2020. Security and privacy attacks during data communication in software defined mobile clouds. *Computer Communications* (2020).

[29] Priyanka Naik, Dilip Kumar Shaw, and Mythili Vutukuru. 2016. NFVPerf: Online performance monitoring and bottleneck detection for NFV. In *NFV-SDN*.

[30] Netfilter Org. 2021. IPTables. Available at: https://www.netfilter.org/.

[31] Numpy. 2021. The fundamental package for scientific computing with Python. https: https://scikit-learn.org/stable/.

[32] OpenStack. 2021. OpenStack. Available at: https://www.openstack.org/.

[33] OpenStack. 2021. Tacker. Available at: t.ly/8dh7.

[34] OPNFV Group. 2021. Available at: https://www.opnfv.org/.

[35] OProfile. 2022. Linux system profiler. Available at: t.ly/rqN0.

[36] Alaa Oqaily, LT Sudershan, Yosr Jarraya, Suryadipta Majumdar, Mengyuan Zhang, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. 2020. NFVGuard: Verifying the Security of Multilevel (NFV) Stack. In *CloudCom*.

[37] Momen Oqaily, Yosr Jarraya, Mengyuan Zhang, Lingyu Wang, Makan Pourzandi, and Mourad Debbabi. 2019. iCAT: An Interactive Customizable Anonymization Tool. ESORICS, Springer, 658–680.

[38] OSM Group. 2021. Open Source MANO. Available at: https://osm.etsi.org/.

[39] Manuel Peuster and Holger Karl. 2016. Understand your chains: Towards performance profile-based network service management. In *EWSDN*.

[40] Manuel Peuster and Holger Karl. 2017. Profile your chains, not functions: Automated network service profiling in devops environments. In *NFV-SDN*.

[41] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. 2015. The design and implementation of open vswitch. In *NSDI. USENIX*.

[42] Raphael Vicente Rosa, Christian Esteve Rothenberg, and Robert Szabo. 2015. VBaaS: VNF benchmark-as-a-service. In *EWSDN*.

[43] Scikit-learn. 2021. Machine Learning in Python. https: https://numpy.org/.

[44] Mustafizur R Shahid, Gregory Blanc, Houda Jmila, Zonghua Zhang, and Hervé Debar. 2020. Generative Deep Learning for Internet of Things Network Traffic Generation. In *PRDC*.

[45] Snort Org. 2021. snort. Available at: https://www.snort.org/.

[46] Soumith Chintala, Emily Denton, Martin Arjovsky, Michael Mathieu. 2021. How to Train a GAN? Tips and tricks to make GANs work. Available at: https://github.com/soumith/ganhacks.

[47] Tcpdump. 2021. Tcpdump. Available at:http://www.tcpdump.org/index.html.

[48] Nguyen Canh Thang and Minho Park. 2019. Detecting compromised switches and middlebox-bypass attacks in service function chaining. In *ITNAC*. IEEE.

[49] Sudershan Lakshmanan Thirunavukkarasu, Mengyuan Zhang, Alaa Oqaily, Gagandeep Singh Chawla, Lingyu Wang, Makan Pourzandi, and Mourad Debbabi. 2019. Modeling NFV deployment to identify the cross-level inconsistency vulnerabilities. In *CloudCom. IEEE*.

[50] Ke Tian, Steve TK Jan, Hang Hu, Danfeng Yao, and Gang Wang. 2018. Needle in a haystack: Tracking down elite phishing domains in the wild. In *IMC*. 429–442.

[51] Brendan Tschaen, Ying Zhang, Theo Benson, Sujata Banerjee, Jeongkeun Lee, and Joon-Myung Kang. 2016. Sfc-checker: Checking the correct forwarding behavior of service function chaining. In *NFV-SDN*.

[52] Ubuntu. 2021. Cloud Images. Available at: https://cloud-images.ubuntu.com/.

[53] Fulvio Valenza, Serena Spinoso, and Riccardo Sisto. 2019. Formally specifying and checking policies and anomalies in service function chaining. *Journal of Network and Computer Applications* 146 (2019), 102419.

[54] Steven Van Rossem, Wouter Tavernier, Didier Colle, Mario Pickavet, and Piet Demeester. 2019. Profile-based resource allocation for virtualized network functions. *IEEE TNSM* 16, 4 (2019), 1374–1388.

[55] Steven Van Rossem, Wouter Tavernier, Didier Colle, Mario Pickavet, and Piet Demeester. 2020. VNF Performance Modelling: From stand-alone to chained topologies. *CN* 181 (2020).

[56] VMware. 2020. VMware Expands Its VMware Ready for Telco Cloud Program to Accelerate the Deployment of 5G Services. Available at:t.ly/BIIW.

[57] Denis Volkhonskiy, Ivan Nazarov, and Evgeny Burnaev. 2020. Steganographic generative adversarial networks. In *Twelfth international conference on machine vision (ICMV 2019)*, Vol. 11433. SPIE, 991–1005.

[58] Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2020. Hybrid-shield: Accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks. In *CCD*.

[59] Si Yu, Gui Xiaolin, Lin Jiancai, Zhang Xuejun, and Wang Junfei. 2013. Detecting vms co-residency in cloud: Using cache-based side channel attacks. *Elektronika* (2013).

[60] Xiaoli Zhang, Qi Li, Jianping Wu, and Jiahai Yang. 2017. Generic and agile service function chain verification on cloud. In *IWQoS*.

[61] Xiaoli Zhang, Qi Li, Zeyu Zhang, Jianping Wu, and Jiahai Yang. 2020. VSFC: Generic and agile verification of service function chains in the cloud. *IEEE/ACM ToN* (2020).

[62] Ying Zhang, Wenfei Wu, Sujata Banerjee, Joon-Myung Kang, and Mario A Sanchez. 2017. SLA-verifier: Stateful and quantitative verification for service chaining. In *INFOCOM*.