ProvTalk: Towards Interpretable Multi-level Provenance Analysis in Networking Functions Virtualization (NFV)

Azadeh Tabiban[1*], Heyang Zhao[1], Yosr Jarraya[2], Makan Pourzandi[2], Mengyuan Zhang[3], Lingyu Wang[1*]
[1] CIISE, Concordia University, {a_tabiba, z_heyang, wang}@encs.concordia.ca
[2] Ericsson Security Research, Ericsson Canada, {yosr.jarraya, makan.pourzandi}@ericsson.com
[3] Department of Computing, The Hong Kong Polytechnic University, mengyuan.zhang@polyu.edu.hk

***Abstract*—Network functions virtualization (NFV) enables agile deployment of network services on top of clouds. However, as NFV involves multiple levels of abstraction representing the same components, pinpointing the root cause of security incidents can become challenging. For instance, a security incident may be detected at a different level from where its root cause operations were conducted with no obvious link between the two. Moreover, existing provenance analysis techniques may produce results that are impractically large for human analysts to interpret due to the inherent complexity of NFV. In this paper, we propose ProvTalk, a provenance analysis system that handles the unique multi-level nature of NFV and assists the analyst to identify the root cause of security incidents. Specifically, we first define a multi-level provenance model to capture the dependencies between NFV levels. Next, we improve the interpretability through three novel techniques, i.e., multi-level pruning, mining-based aggregation, and rule-based natural language translation. We implement ProvTalk on a Tacker-OpenStack NFV platform and validate its effectiveness based on real-world security incidents. We demonstrate that ProvTalk captures management API calls issued to all NFV services, and produces more interpretable results by significantly reducing the size of the provenance graphs (about 3.6 times reduction via the multi-level pruning scheme and two times reduction via the aggregation scheme). Our user studies show that ProvTalk facilitates the analysis task of real-world users by generating more interpretable results.**

I. Introduction

Today, softwarized services are increasingly deployed over virtual resources (e.g., containers or VMs) sharing underlying physical infrastructures [1], [2]. In particular, NFV enables the replacement of proprietary devices with software network services and allows for more dynamic and agile network service deployment in the cloud [3]. This new paradigm converts traditional networking into a multi-level NFV stack by running virtual services over multiple levels of virtual resources. Each level of the NFV stack is operated by a different managerial component accessible through its API interface to create, modify or delete network services and their related resources. Such added complexity of NFV may increase the risk of vulnerabilities and misconfiguration in the deployed services [4], [5]. For instance, by exploiting `CVE-2020-12689` [6], an attacker can gain unauthorised access to the NFV management API, and compromise other clients' network services. The multi-level nature of NFV together with its sheer scale and complexity may also render pinpointing the root cause of security incidents more challenging. Existing solutions in NFV (e.g., [7], [8]) mainly focus on localizing failed components instead of identifying the activities leading to the incident.

Unique challenges of provenance analysis in NFV. Data provenance is a well-established technique used for root cause analysis that has been applied in other domains such as IoT (e.g., [9]), SDN (e.g., [10]), cloud (e.g., [11]) and operating systems (e.g., [12]). Most existing approaches rely on tracking system-level events (detailed in Section V-C and VI). The existing management-level solution [11] is limited to clouds with no support for multi-level/cross-level analysis (more comparison in Section V-C3). However, applying provenance analysis to each level of the NFV stack (e.g., cloud), while ignoring the relationships between levels, would be insufficient. For instance, a user request for creating a network service may lead to a series of operations to create virtual resources across different levels of the NFV stack. Without capturing the dependencies between such operations and resources, a provenance analysis would not be able to link a security incident to its root cause if they happen to be at different levels of the NFV stack.

There exist provenance analysis solutions for other multi-level systems (e.g., SDN). However, one unique aspect of NFV that distinguishes it from those systems is that different NFV levels are actually representing the same components (e.g., a virtual firewall) with different degrees of abstraction (e.g., as a service, a virtual network function, or a virtual machine) as detailed in Section II-A. In contrast, most existing multi-level provenance solutions (e.g., [13], [14], [15]) mainly focus on multiple systems working together while each plays a different role (e.g., SDN controller vs. applications [16], or the operating system of the host vs. applications [15]). Therefore, their provenance graphs do not need to be explicitly segregated into different levels that can be mapped back to each other. In other words, although those solutions can analyse the interactions between different systems, they do not support the need for analysing the information flows to/from different *abstractions* of the *same* resources that we face in NFV (which will be further illustrated through a concrete example in Section II-C).

*Corresponding author.

Network and Distributed Systems Security (NDSS) Symposium 2022
27 February - 3 March 2022, San Diego, CA, USA
ISBN 1-891562-74-6
https://dx.doi.org/10.14722/ndss.2022.23103
www.ndss-symposium.org

Moreover, the sheer scale of NFV environments implies impractically large and complex provenance graphs for human analysts to interpret. Some recent works (e.g., [17], [18]) focus on assisting the analyst by identifying the high-level abstraction of behavior corresponding to different parts of the provenance graph. However, these approaches still require some level of domain-knowledge. For instance, the analyst may be required to determine a label (e.g., program compilation) for each extracted subgraph in the training dataset [17]. Determining such labels can be especially challenging in a multi-tenant environment like NFV considering the complexity and interleaving nature of the users' behavior.

Key ideas. In this work, we argue that the uniqueness of NFV not only leads to novel challenges in provenance analysis but also brings about new opportunities. To explore such opportunities, we propose ProvTalk, an interpretable multi-level provenance analysis approach for assisting security analysts to investigate the root cause of security incidents in NFV. The key insight behind ProvTalk is that the multi-level aspect of NFV intrinsically provides higher level semantics corresponding to lower-level concepts (e.g., a virtual firewall is represented as a virtual network function or virtual machine at lower levels). By establishing such a cross-level mapping, we can trace a security incident back to its root causes located at a different level, while improving the interpretability of the provenance graph. Specifically, ProvTalk *links* the provenance graphs at different levels of the NFV stack through capturing the cross-level dependencies between different abstractions of the same network service, which also implicitly captures the cross-level dependencies among operations. Then, based on the captured dependencies, ProvTalk improves the interpretability of its results in three steps. First, ProvTalk removes irrelevant information from provenance graphs by propagating the pruning label of nodes across different levels based on the established dependencies between resources and operations. Second, through mining (system or user-related) frequent patterns, ProvTalk hides redundant nodes by visually aggregating them into a single node (which can be expanded to reveal the hidden details when necessary). Third, ProvTalk leverages a rule-based approach to automatically translate details of a provenance graph into a human-readable format to provide high-level guidance to analysts.

In summary, our main contributions are as follows.

- To the best of our knowledge, ProvTalk is the first provenance-based solution specifically designed for NFV. Our provenance model captures the unique multi-level nature of NFV environments, and provenance analysis using ProvTalk allows tracing a security incident back to its root cause potentially located at a different level.
- We propose three novel techniques to improve the interpretability of provenance graphs. The multi-level pruning and mining-based aggregation schemes can both reduce the size and complexity of provenance graphs, and the rule-based translation can provide useful guidance to analyzing provenance graphs. These techniques can not only ease the task of human analysts in applying ProvTalk to large scale NFV environments, but also be potentially applied to other multi-level virtual environments.
- We implement ProvTalk and integrate it into our Tacker-OpenStack [19] testbed as an attached middleware. We validate the effectiveness of ProvTalk based on real-world security incidents. Our experiments using both real-world data and testbed data show that ProvTalk produces more interpretable provenance graphs with significant reduction in their sizes, without losing the information vital for the investigation. We demonstrate that ProvTalk can capture all management API calls while incurring an insignificant storage, latency and computational overhead. Finally, our user studies show that ProvTalk can markedly ease the analysis task of real-world users.

The remainder of this paper is organized as follows: Section II provides NFV background and motivates our solution. Section III describes our methodology. Section IV details the implementation of ProvTalk, and Section V presents our experimental results and user studies. Section VI discusses limitations and future work. Section VII reviews related work. Section VIII concludes the paper.

II. Background and Motivating Example

This section provides background on NFV, defines our threat model and describes a motivating example for ProvTalk.

A. NFV Background

The left side of Fig. 1 illustrates a high-level view of the ETSI NFV reference architecture, where a user-specified service description is implemented through the virtual network function (VNF) block (which provides a high-level representation of network functions) and the NFV infrastructure (NFVI) block (which represents the underlying cloud infrastructure), while both blocks are provided by the same network operator [3]. The right side of Fig. 1 shows an example of a multi-level network service deployment with corresponding management modules, and depicts how the actual deployment of a network service would correspond to the ETSI architecture. Specifically, the VNF block in the ETSI architecture maps to the *NFV level*, where a network service is deployed as several VNFs forming a VNF forwarding graph (VNFFG). The NFVI block in the ETSI architecture is mapped to both the service function chaining (*SFC level*), where there are virtual resources such as port pair groups, and the *cloud level*, where there are virtual resources such as VM, port and subnet. Finally, users can manage those levels through management modules including Network Function Virtualization Orchestrator (NFVO), Virtual Network Function Manager (VNFM), Software Defined Networking Controller (SDN-C) and Virtualized Infrastructure Manager (VIM).

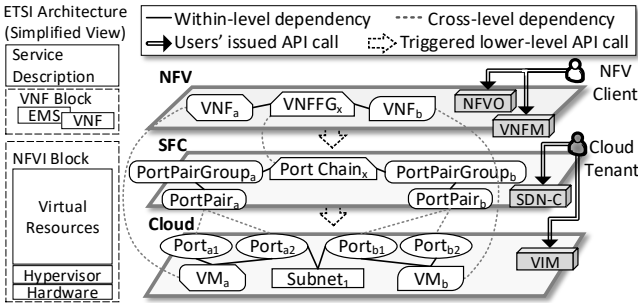


Fig. 1: High-level view of ETSI NFV reference architecture (left); Example of multi-level network service deployment showing the dependencies between resources (right).

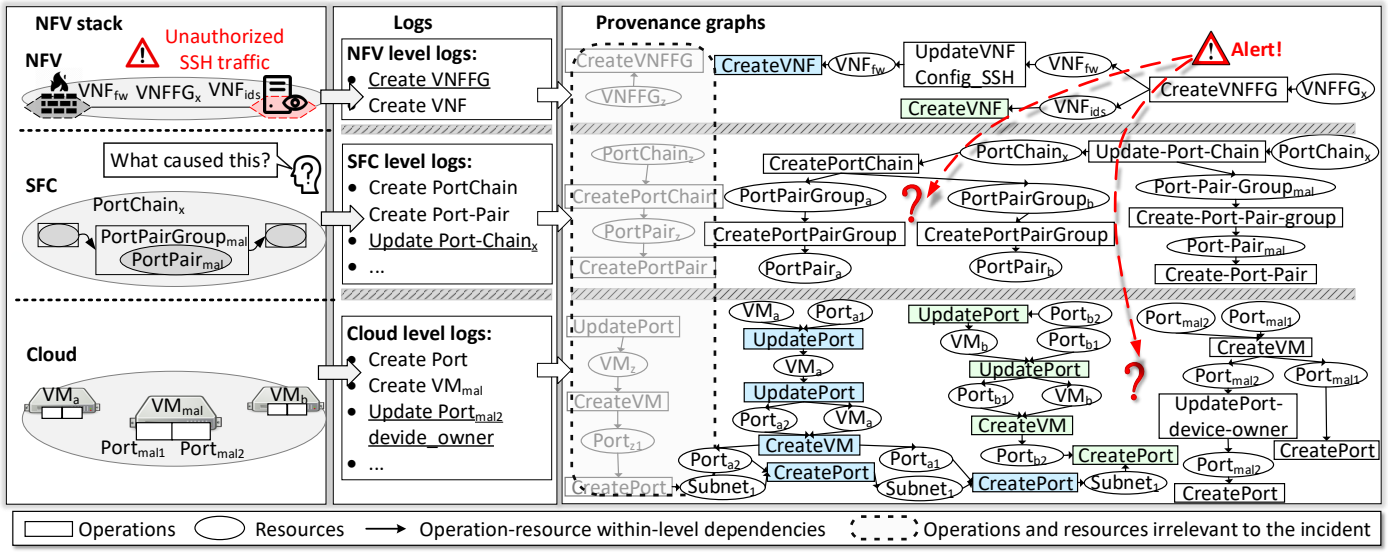


Fig. 2: Attack scenario detected at the NFV-level (left); log-based analysis (middle); excerpts of the provenance graphs (right).

Fig. 1 demonstrates how dependencies may exist between resources either at the same level or between different levels. For example, the solid lines at the NFV level indicate that the two VNFs, namely VNF_a and VNF_b, are part of (and connected through) a VNFFG named $VNFFG_x$ (similar dependencies exist between the port pair groups and port chains, the VMs and ports, as well as the ports and subnets). Furthermore, the dashed lines represent dependencies across different levels. For example, the dashed line between VNF_a (NFV-level) and VM_a (cloud-level) indicates that the creation of VNF_a will automatically trigger the creation of VM_a. Similarly, the creation of $VNFFG_x$ will automatically trigger the creation of $PortChain_x$. Finally, a dashed line links $PortPair_a$ (SFC level) to its two components ($Port_{a1}$ and $Port_{a2}$).

The above example shows how operations executed at different levels might affect resources and consequently introduce (within-level or cross-level) dependencies in the NFV stack. These dependencies are crucial to correctly identify the root cause of security incidents in NFV. To capture those concepts, we define a multi-level provenance model in Section III-A1.

B. Threat Model and Assumptions

Our in-scope threats include both external attackers who exploit existing vulnerabilities in the NFV stack, and insiders, such as NFV clients, cloud users and tenant administrators, who make the NFV stack exploitable either through mistakes or by malicious intentions. As our provenance model focuses on capturing management operations, we limit our scope to attacks that involve operations directed through the NFV management interfaces (e.g., command line and dashboard). Similar to most existing provenance solutions, we assume our solution is deployed by the owner of the system, and thus it has access to the full NFV stack, and we assume the NFV stack management modules, the provenance building mechanism and the provenance storage are all protected with existing techniques such as remote attestation [20], [21], hash-chain-based provenance storage protection [22] or type enforcement [23].

Out-of-scope threats include attacks that involve no management operations or resources captured in the provenance graph, and attacks that can completely bypass the NFV management interfaces. Moreover, as with most works on provenance analysis, we do not consider attackers who can temper (either through attacks or by using insider privileges) the infrastructure management system (e.g., breaching the integrity of the API calls and databases of services) or the provenance solution itself. Although our framework provides more interpretable information for easier analyses, it relies on the human analysts to pinpoint the root cause at the end. Finally, although our provenance results may lead to the discovery of existing vulnerabilities or misconfigurations, our focus is not on vulnerability analysis, intrusion detection, or configuration verification, and our solution is expected to work in tandem with those solutions.

C. Motivating Example

In this section, we provide a motivating example to show the benefit of applying ProvTalk. Fig. 2 illustrates an attack scenario (left) and the challenges faced by a security analyst (middle and right) in investigating the root cause.

Tedious log investigation. Upon receiving an alert from the virtual IDS (VNF_{ids}) about unauthorized SSH traffic, an analyst begins investigating the root cause of this incident. As shown in Fig. 2 (middle), the analyst may need to inspect thousands of log entries at all levels of the NFV stack. However, this can be cumbersome if done manually, since there is no apparent relationship between those entries.

Lack of cross-level dependencies. To establish the relationships between log entries, assume the analyst applies a provenance analysis tool (e.g., [24], [11]) to each level. As shown in Fig. 2 (right), the tool would generate a provenance graph that shows the virtual IDS (VNF_{ids}) becomes part of $VNFFG_x$ through *CreateVNFFG* operation, where it is preceded by a virtual firewall (VNF_{fw}). The analyst can also see that VNF_{fw} is configured to filter the SSH traffic (by *UpdateVNF-Config-SSH* operation), and thus, the security incident (unauthorized SSH traffic) is not supposed to happen. At this point, the analyst is unable to proceed further using the provenance graph at the NFV-level alone, since no other operations at this level can explain what caused the incident.

Size and complexity of multi-level analysis. For the sake of this example, now suppose the analyst manually establishes the cross-level dependencies based on his/her experiences. For instance, he/she can identify that $VNFFG_x$ is deployed as $PortChain_x$ at a lower (SFC) level. He/she could then link the provenance graphs at different levels to each other, and continue the investigation at the lower (SFC and cloud) levels. However, the sheer scale of NFV environments and the existing dependencies among a large number of resources (e.g., there may be hundreds of VMs attached to $Subnet_1$) mean that pinpointing the root cause among all the nodes and edges in the provenance graph is still very challenging. On the other hand, the provenance graph contains a lot of irrelevant or redundant information. For instance, in Fig. 2 (right), the grayed out portion of the provenance graph is actually irrelevant to the incident, since it corresponds to an irrelevant VNFFG ($VNFFG_z$). Moreover, the groups of green and blue nodes at the cloud-level are triggered by the platform after each NFV-level operation *CreateVNF*, and thus are redundant. Leaving such information as-is in the provenance graph can make the analysis time consuming and error-prone.

Example output of ProvTalk. ProvTalk is designed to address all the aforementioned challenges in an automatic fashion, so analysts can focus on the most relevant information for identifying the root cause. For example, Fig. 3a illustrates the result of ProvTalk corresponding to the above example. While we leave the details (e.g., the hexagons and hatched box) to Section III, the attack scenario is self-explanatory from the figure as follows. First, the attacker creates two ports (i.e. $Port_{mal1}$ and $Port_{mal2}$). Next, he/she updates the *device_owner* field of $Port_{mal2}$, and immediately creates VM_{mal} attached to this port so that he/she can exploit a vulnerability [25] for spoofing the IP address of the enterprise sending network traffic into $VNFFG_x$. Finally, the attacker inserts the port pair group of VM_{mal} into the port-chain corresponding to $VNFFG_x$ (via *Update-Port-Chain* operation). The attacker can then send malicious traffic using VM_{mal} inserted between the virtual firewall and IDS services to evade them.

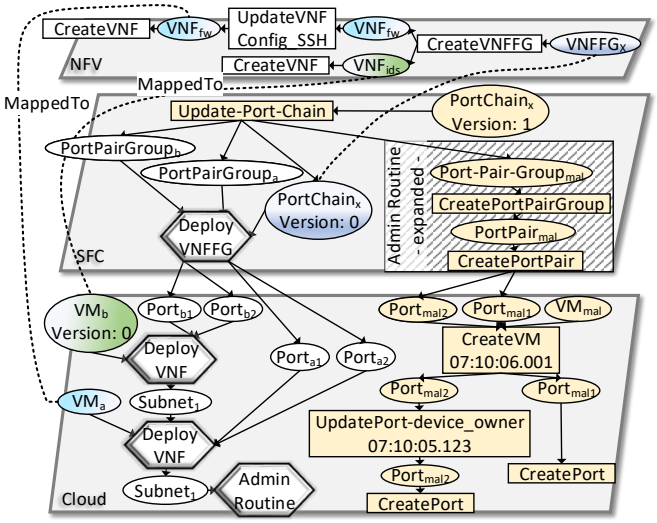
Comparison to existing works. In Fig. 3a, we can see the provenance graph generated by ProvTalk is explicitly divided into three disjoint levels, with nodes corresponding to a common resource "mapped to" each other (e.g., VNF_{fw} vs. VM_a). In contrast, Fig. 3b and 3c show that the provenance graphs of existing "multi-tier" provenance techniques (e.g., [15], [16]) usually do not have such explicit separation between levels, as they focus more on the information flow between multiple systems that play different roles (e.g., operating system vs. application in Fig. 3b, or application vs. control plane in Fig. 3c). Therefore, such works are not designed to support the need for capturing the information flow to/from different abstractions of the same resource in NFV.

III. ProvTalk

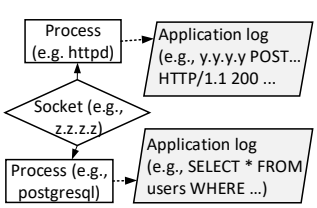
In this section, we detail different modules of our approach. Fig. 4 shows an overview of ProvTalk consisting of three main stages: provenance construction, training and investigation.

A. Provenance Construction

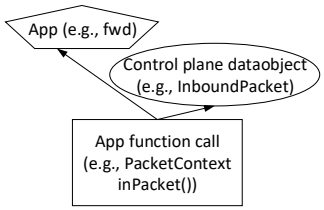
In this section, we define our NFV provenance model, and describe the provenance construction module.



(a) The result of ProvTalk for the attack scenario of Fig. 2 (details in Section III-C). Nodes visualizing the same resource have the same light colors.



(b) An example provenance graph of OmegaLog [15] for comparison.



(c) An example provenance graph of ProvSDN [16] for comparison.

Fig. 3: An example output of ProvTalk (a), and comparing it to existing works [15], [16] to highlight the different multi-level nature (b) and (c).

1) NFV Provenance Model: We define a platform-independent provenance model[1] based on the standard specification PROV-DM [26]. The example provenance graph in Fig. 3a follows this model to show two types of nodes: *entities* and *activities*. Entities (shown as ovals) represent virtual resources (e.g., VM_{mal}), and activities (shown as boxes) represent management operations (e.g., *CreateVM*). To avoid cycles, we adopt the node versioning method as in [27], [28], where a new version of an entity is created if an operation affects its represented resource. For example, Fig. 3a shows that a new node representing $PortChain_x$ (i.e., the node $\langle PortChain_x, Version1 \rangle$) is created after it gets updated by *Update-Port-Chain* operation. Directed edges denote the dependency between an operation and its generated or used resources. For instance, the edge from *Update-Port-Chain* to $PortPairGroup_{mal}$ shows that this operation uses $PortPairGroup_{mal}$ to update $PortChain_x$. Finally, we represent *cross-level dependencies* by edges labeled as *MappedTo*, which connect the entities related to the same resource at different levels, e.g., the edge between VNF_{ids} and VM_b in Fig. 3a.

2) Building the Provenance Graph: To capture all operations affecting virtual resources, we deploy our event interception mechanism as middlewares [29], [11] attached to managerial services at all levels of the NFV stack. These services include but are not limited to networking, compute, storage, image and NFV orchestration services. In Section V-D, we

[1] Additional details of our provenance model can be found in Appendix A.

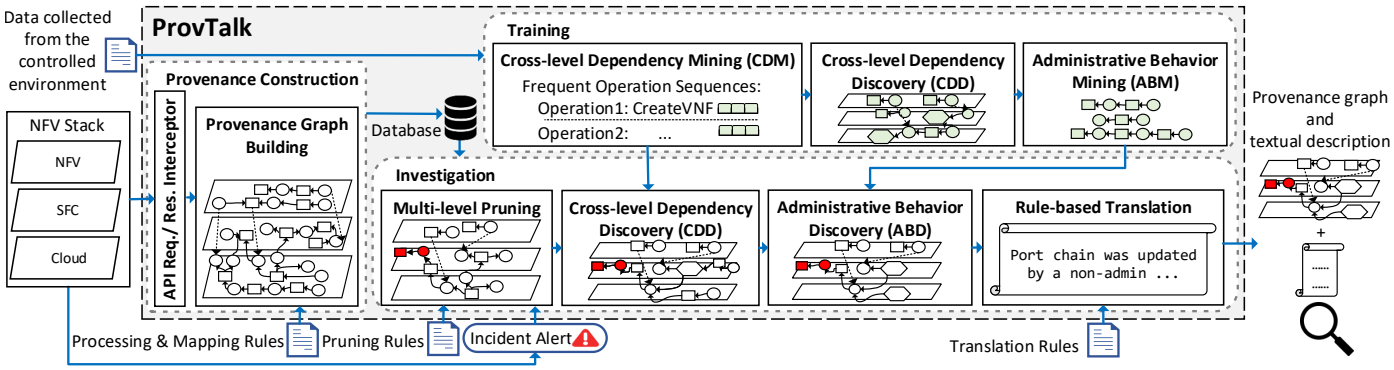


Fig. 4: The overview of ProvTalk.

show that intercepting all management API calls through the deployment of our middlewares is enough to satisfy the *completeness* property. Moreover, to process the intercepted API calls, we define a set of parsing and typing rules according to cloud and NFV documentations [30], [31], [32]. Specifically, the information required for building the provenance graph is embedded in the intercepted API calls. Therefore, upon intercepting each API call and based on the provided rules, ProvTalk parses the fields of that API call, identifies the type of the requested operation (e.g., *CreateVNF*), determines the affected virtual resources, and the ID of the user issuing that API call. Next, it creates nodes representing the operation and the affected resources with edges capturing their dependencies. ProvTalk also stores the extracted information (e.g. user ID) and the time of interception as node attributes. Then, it appends the created nodes and edges to the provenance graph stored in the backend graph database.

Additionally, to capture the cross-level dependencies between resources, we provide ProvTalk with a set of mapping rules. Upon the creation of each NFV-level resource, NFV platforms automatically store the ID of that resource and its lower-level associated resource in specific cells of the platform databases. Therefore, we define rules specifying the queries for extracting the IDs of those resources and creating *MappedTo* edges between their nodes. For instance, the ID and associated virtual service of a VM are stored in two columns in the same row of table *nova_instances*. Accordingly, we define the rule "Upon the interception of *CreateVNF* operations, ProvTalk should issue a query to nova_instances table to extract VM-ID from the same row storing the ID of the created VNF".

B. Multi-level Pruning

The provenance graph may include a large number of nodes and edges that are irrelevant to the target incident. Most of the existing pruning techniques are designed for single-level provenance models [11], [9], [28], and they remove irrelevant nodes according to the analyst's provided pruning criteria. However, in the specific context of multi-level NFV environments, this approach has the limitation of identifying and pruning irrelevant nodes only at the same level as where the target incident is detected. In other words, those techniques do not factor in the dependencies between provenance graphs captured at different levels, which can be useful in identifying the potentially irrelevant nodes across different levels. The analyst could certainly provide additional pruning criteria for identifying irrelevant nodes at every level. However, this requires more effort from the analyst, and it also assumes he/she has a good understanding about all levels of the NFV stack and corresponding security assumptions. The reliance on such assumptions may make the pruning error-prone and result in pruning nodes that are indeed relevant to the attack.

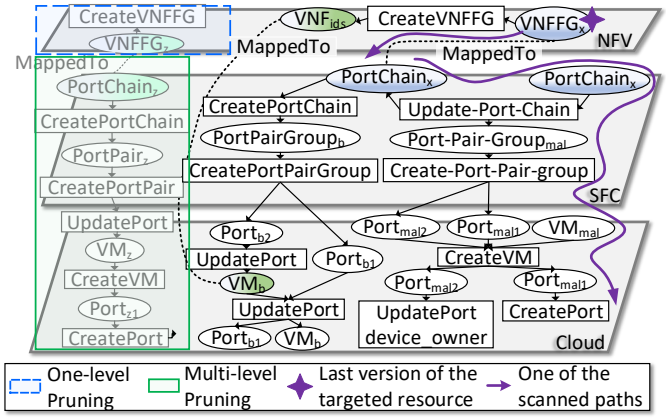


Fig. 5: Excerpt of the provenance graph related to the incident discussed in Section II-C contrasting the effectiveness of one-level [11] and multi-level pruning.

To address those issues, we propose a multi-level pruning mechanism to automatically identify and keep the potentially relevant nodes across different levels by leveraging cross-level dependencies (Section III-A1). Specifically, after labeling all the potentially relevant nodes at the same level as the target resource (i.e., the resource associated with the incident), ProvTalk passes the label assigned to each resource to its corresponding resources at other levels via the cross-level dependencies (*MappedTo* edges). Next, ProvTalk further follows all paths to pass the labels to all the reachable nodes. To further narrow down the analysis, ProvTalk allows the analyst to specify some additional constraints about the nodes passing the labels. Finally, ProvTalk discards nodes that are not labeled as they are not reachable from the target resource.

Example 1. Fig. 5 shows an excerpt of the provenance graph in our motivating example (Section II-C) contrasting one-level pruning (e.g., [11]) with multi-level pruning. Both approaches first identify the target resource *VNFFG*$_x$ (starred node), assign it a label and pass it to all reachable nodes meeting the provided criteria. Then, all non-labeled nodes are pruned. For the one-level pruning, as the cross-level dependencies are not considered, the label is only passed to all reachable nodes

at the same level (e.g. VNF_{ids}, *CreateVNF*). Thus, only the group of nodes at the NFV-level (inside dashed blue box) are pruned, leaving the graph at other levels with potentially irrelevant nodes. In contrast, ProvTalk leverages cross-level dependencies (i.e., *MappedTo* edges) to pass the label from the target resource to nodes at lower levels (e.g., $PortChain_x$ and the path specified by an arrow). Thus, the irrelevant nodes inside the green box can also be identified and pruned.

C. Aggregation

The pruned provenance graph may still include a large amount of redundant information. However, pruning those is not a viable option as they may contain valuable information about the root cause. For example, some operations such as *CreatePortPair* and *CreatePortPairGroup* (hatched box in Fig. 3a) are frequently issued by cloud admins as a part of routine maintenance tasks. However, as explained in Section II-C, they may also be part of the attack steps (e.g., operations issued by the attacker to insert a malicious VM into a port chain). Hence, if we hypothetically remove those operations due to their redundancy, the analyst would fail to pinpoint the root cause. Additionally, there is no systematic way of associating high-level semantics to those frequent operations, which could make the provenance graph easier to understand. For instance, the analyst cannot identify the cloud-level operations in Fig. 2 (groups of blue and green nodes) that were automatically triggered after each NFV-level operation *CreateVNF*.

Therefore, we propose an aggregation-based solution that *visually* aggregates the nodes corresponding to such operations into a compound node labeled with the corresponding NFV-level operation or administrative routine. Our aggregation technique is designed to be fully reversible such that each compound node can be easily expanded to show the original nodes, which allows the analyst to easily recover the potentially useful details of the aggregated nodes (hexagons and hatched box in Fig. 3a). Our approach consists of two mining-based schemes for *cross-level operations* and *administrative tasks operations*, which involve training and investigation stages. Similar to most approaches in this area (e.g., [33]), we collect training data from a controlled environment to ensure there is no involvement of malicious actors.

1) Cross-level Aggregation: The sequence of lower-level operations automatically triggered after an NFV-level operation are generally fixed, and thus frequently appear in the provenance graph causing redundancy. To avoid this, ProvTalk leverages a mining-based approach to model such sequences, and then applies the model to identify and aggregate the lower-level nodes corresponding to each NFV-level operation.

Cross-level Dependency Modeling (CDM). Since lower-level operations are generally triggered shortly after their corresponding NFV-level operations, we leverage this intuition to build a model of those lower-level operations generated within a small time interval. To this end, our API interceptors log operations triggered at all levels (i.e., NFV, SFC, and cloud) with a *timestamp* indicating the time when ProvTalk intercepts each operation. Next, based on those timestamps, ProvTalk extracts a sequence of lower-level operations triggered within t_{CDM} seconds after each logged NFV-level operation. The analyst may determine the interval t_{CDM} based on studies of the NFV platform (e.g., computational power).

However, since there may be many operations issued at almost the same time in a real-world NFV environment, the extracted sequences may include irrelevant lower-level operations (e.g., the operations triggered by other NFV-level operations). To address this issue, we model relevant operations by deriving frequent patterns of lower-level operations. Specifically, for each NFV-level operation, we feed the extracted sequences to our sequential pattern mining algorithm (an efficient self-supervised method for discovering the frequent patterns of ordered items in a controlled environment), which then outputs the list of mined patterns with their frequencies (i.e., support [34]). Finally, we identify the most frequent patterns related to each NFV-level operation, which are provided to the CDD module during the investigation stage.

Example 2. Fig. 6 depicts modeling the lower-level management operations triggered by the NFV-level operation, *CreateVNF*. ProvTalk extracts sequences of lower-level operations logged shortly after each *CreateVNF* (shown by rows of the *Sequences* table). We show two example scenarios causing different extracted sequences corresponding to the *CreateVNF* operation. Scenario1 describes cases where a single user issues a *CreateVNF* operation. Scenario2 describes cases where two NFV-level operations, *CreateVNF* and *DeleteVNF*, are issued at approximately the same time, and thus the extracted sequence includes an irrelevant operation, *DeleteVM* (triggered by *DeleteVNF*). To derive the operations triggered by *CreateVNF*, ProvTalk retrieves the most frequently observed patterns in *Sequences* table, which yields [*CreatePort*, *CreatePort*, ...] with the support value of 60%.

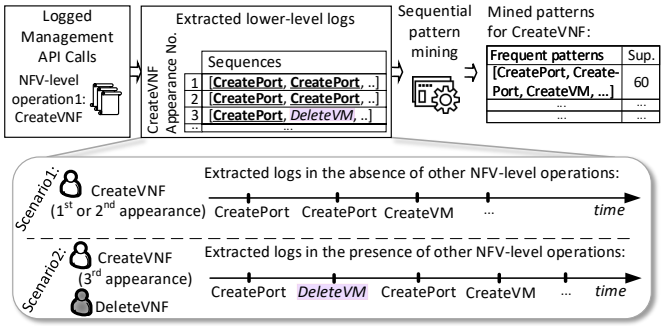


Fig. 6: An example of cross-level dependency mining for the NFV-level operation, *CreateVNF* (top); Scenarios leading to different extracted sequences of lower-level operations corresponding to the *CreateVNF* operation (bottom).

Cross-level Dependency Discovery (CDD). During the investigation stage, CDD identifies and aggregates the nodes related to the mined operations corresponding to the same NFV-level operation (e.g., *CreateVNF*). This can be challenging, since there usually exist many nodes representing the same type of operation (e.g., several *CreatePort* nodes in Fig. 2). Our key insight is that, since all the triggered operations correspond to the same NFV-level operation, we can expect some dependency among them. Additionally, due to cross-level dependencies between resources, if an operation affects a resource at the NFV-level, a triggered operation will affect its associated lower-level resource. Based on such intuition, Fig. 7 shows how CDD works. First, CDD identifies the node representing the resource affected by an NFV-level operation (e.g., a created *VNF* at

NFV-level) and its lower-level associated resource connected by a *MappedTo* edge (e.g., its corresponding *VM* at cloud-level). Next, to start an iteration, it tags the operation node connected to the resource node, removes that operation from the mined sequence, then tags the next connected resource node on the path. The iteration stops once the sequence is empty. Finally, CDD visually aggregates the tagged nodes into a compound node labeled as the corresponding NFV-level operation (see Appendix B for more details).

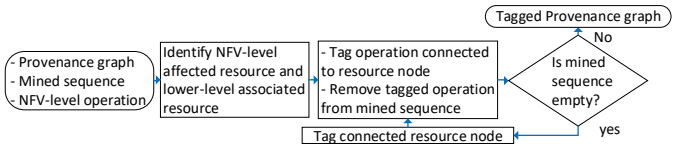
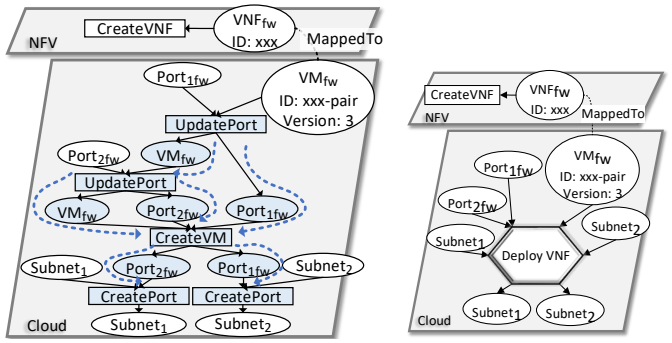


Fig. 7: Identifying the cross-level dependencies.

Example 3. Fig. 8 depicts an example aggregation related to *CreateVNF* operation. CDD identifies the lower-level node $\langle VM_{fw}, Version\ 3\rangle$ associated with VNF_{fw}. Next, it identifies the nodes representing the operations triggered by *CreateVNF* and their affected resources (Fig. 8a). The identified nodes are visually aggregated into a compound node (the hexagon in Fig. 8b), which is labeled as *DeployVNF*[2].



(a) (b)

Fig. 8: Example of cross-level dependency discovery (a) before and (b) after aggregation.

2) Administrative Behavior Aggregation: To further ease the interpretation, we aggregate the routine administrative operations (e.g., maintenance tasks) repetitively appearing in the provenance graph. ProvTalk mines the frequent sequences representing the paths (that are not aggregated by CDD) in the training stage, and aggregates the paths matching the mined sequences in the investigation stage.

Administrative Behavior Modeling (ABM). This module builds a model of routine administrative behavior based on frequent paths and using sequential pattern mining [34]. Fig. 9 shows the steps of our ABM module: 1) ABM retrieves all paths with the length of at most $l_{routine}$. The analyst can adjust $l_{routine}$ based on the requirements of the investigated platform, e.g., the regularity of life-cycle management of resources. 2) ABM converts the retrieved paths into string sequences. Our intuition is that nodes compose a path in a similar way that items compose an ordered sequence. Formally, a causal path can be translated into a sequence of items $[f(res\_node_i), f(op\_node_i), f(res\_node_{i+1}), ...]$ where f is the function for obtaining the string representation of a node. In this work, we use the resource or operation type attribute as the string representation of each node. For example, the path $(Port_1)\leftarrow(CreateVM)\leftarrow(VM_1)\leftarrow(StartVM)$ is converted into the following sequence: [*Port*, *CreateVM*, *VM*, *StartVM*]. 3) Finally, we apply the sequential pattern mining algorithm BIDE [34] to identify the most frequent patterns which are used by ABD during the investigation stage.

[2]We use the label *DeployVNF*, instead of *CreateVNF*, to make referring to compound nodes easier in the paper.

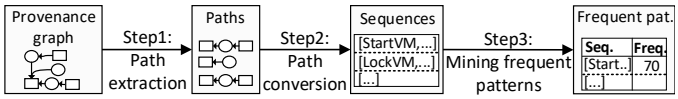


Fig. 9: Steps of ABM module.

Administrative Behavior Discovery (ABD). This module identifies and aggregates the paths with a corresponding sequence that matches the patterns mined by ABM. Specifically, ABD retrieves all paths with the length of at most $l_{routine}$. Next, it converts those paths into sequences of string elements as described in the previous step (the ABM module), while it also captures the IDs (a unique number automatically assigned to each node by the graph database) of the consisting nodes. Then, it identifies the sequences that are observed among the frequent patterns mined by ABM. If a matching sequence is identified, the ABD module finds the nodes corresponding to that sequence using their unique IDs and aggregates them into a single *Admin_Routine* compound node. Moreover, to increase the number of aggregated nodes represented by each compound node (i.e., the reduction power of our scheme), we merge *Admin_Routine* nodes with common aggregated nodes.

Example 4. Fig. 10 shows an example of administrative behavior aggregation. As we can see on the left, two paths are converted into sequences $Sequence_k$ and $Sequence_j$, and are initially aggregated into the blue and red shaded compound nodes. However, due to their common aggregated node, ABD merges them into one single compound node (right side).

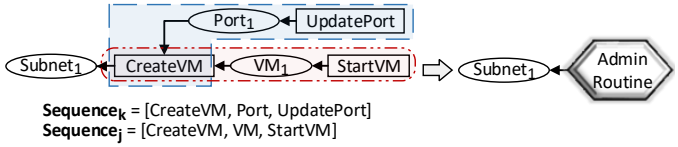


Fig. 10: Example of administrative behavior aggregation and merging compound nodes.

In Fig. 3a, we have shown our motivating example provenance graph after applying both aggregation schemes. As we can see, the resulted provenance graph is significantly smaller and more interpretable with the assigned labels.

D. Rule-based Translation

To further enhance the interpretability of the provenance graph and facilitate the analysis, we propose a rule-based technique to translate the captured information into a human-readable text. As we demonstrate in Section V-E, the generated text can provide useful guidance to the analyst in investigating the provenance graph and identifying the root cause. The analysts may also take advantage of the generated text for filing a report describing the result of their investigations. To this end, ProvTalk first backtracks from the node corresponding to the target resource (e.g., the node $VNFFG_x$ in Fig. 3a)

to retrieve all paths connected to this node. ProvTalk also allows the analyst to specify a time interval so that only paths generated during that time will be translated. Next, it extracts the information captured in each path, and generates a textual description reflecting those information and the incident alert.

As an example, Fig. 11 shows the automatically generated description of the highlighted path in Fig. 3a. The generated text is organized in three paragraphs: the first paragraph reflects the information extracted from the incident alert as well as the number of operations represented in the path. The second paragraph details the information extracted from the path, and the last paragraph includes the information necessary for identifying the described path in the provenance graph.

> By the detection time 21-01-06 11:44:07.769, there are 6 operations performed in the specified time interval 0:00:15.454 hours corresponding to the target entity VNFFGx created by admin user using 2 VNFs.
>
> User 12ddf created PORTmal2 at 07:10:03.403. He updated device_owner Portmal2 at 07:10:05.123. And created VMmal using that port(s) after 0:00:0.878 hours. Then, he created PortPairmal using that port(s) after 0:00:03.202 hours. And created PortPairGroupmal using that PortPair(s). He updated PortChainx, using that PortPairGroup(s) after 0:00:02.239 Hours.
>
> More details can be found in the provenance graph following this node path [215 - 211 - 208 - 207 - 204 - 202]. Node(s) (UpdatePortChain-ID: 215) worth a closer look: nonAdmin user (ID: 12ddf) modifies admin (ID: 53atb) modified resource(s).

Fig. 11: Example of auto-generated textual description.

Path Translation. ProvTalk automatically follows graph paths to extract and include the following four node attributes in sentences of the second paragraph): 1) *user*: ProvTalk treats the user issuing an API call as the *subject* of the sentence, and identifies it by the user ID attribute stored in operation nodes. 2) *operations*: ProvTalk treats operation type node attribute as the *verb* of the sentence. 3) *resources*: resources affected by each operation are treated as *objects*, and are identified by the entities from which there is an edge pointing to an activity (Section III-A1). 4) *timestamps*: ProvTalk includes the timestamp attribute (stored in operation nodes) as the *propositional phrase* in the sentence. Additionally, to smooth the transition between sentences, it uses pronouns and *transitional words*. ProvTalk also applies pre-defined sentence templates to automatically form the descriptions. To support aggregation, analysts can configure ProvTalk to treat the label of compound nodes as the *verb* of generated sentences.

To enable retrieving the information not already reflected in the generated text, the third paragraph describes the unique IDs of the corresponding nodes, using which the analyst can map the generated text back to the provenance graph. Additionally, ProvTalk can be configured to include specific parts of the extracted information that may deserve more attention.

IV. Implementation

We implement ProvTalk in a testbed based on Tacker [19], SFC [32] and OpenStack [2] (a popular platform supporting NFV for telecommunication service providers [35], [36]). We note that only our API interception mechanism is platform-specific (i.e., OpenStack/Tacker), while the modular design of our approach makes it readily adaptable to other multi-level virtualized platforms (see Section VI for detailed discussion). To intercept provenance metadata from the REST API calls issued to different services (i.e., Tacker, SFC, and Openstack services, e.g., Nova and Neutron), we implement our provenance construction module as Python WSGI middleware [37], [38], which stores the provenance graph in Neo4j [39] database, and uses Cypher language [40] to query the database. We implement pruning and aggregation modules in Python, and use BIDE algorithm [34] to mine frequent sequences. We set the interval t_{CDM} to 15 seconds (Section III-C1) and $l_{routine}$ to 10 (Section III-C2). We also use the Th_h=0.8 and Th_l=0.5 as the thresholds in our aggregation (explained in Appendix B). Our translation module exports the provenance graphs into JSON format [41], and uses SimpleNLG Python realiser [42] for generating sentences. To visualize the provenance graphs and enable the interaction with ProvTalk, we provide a frontend graphical user interface. We use Cytoscape [43] to visualize the provenance graph, and support the aggregation and expansion of compound nodes.

V. Evaluation

To evaluate ProvTalk, we seek to answer the following questions:

RQ1: How effective is the provenance model at capturing real-world attacks in NFV environments?

RQ2: To what extent can ProvTalk reduce the size of the provenance graph? What is the effect of accuracy on the performance? How does it compare to the existing techniques?

RQ3: What is the overhead introduced by ProvTalk in terms of latency, computation and storage? How does it compare to the existing OS-level provenance techniques?

RQ4: How complete and sound is ProvTalk in terms of capturing and analysing all management API calls?

RQ5: How helpful is the enhanced interpretability in root cause analysis for real-world users?

Experimental Setup and Dataset. We run ProvTalk on an Ubuntu 18.04 server equipped with Intel Xeon Bronze 3104 CPU @1.70GHz and 128GB of RAM. We conducted our experiments based on both our testbed and a real research cloud dataset. To generate diverse sequences of operations in our dataset, we deploy 31 different types of VNFs while randomly varying their parameters (e.g., the number of virtual ports), and seven variations of VNFFGs with varying parameters (e.g., the number of VNFs per VNFFG). Table I shows statistics about the datasets generated in our NFV testbed.

TABLE I: Statistics of our NFV testbed datasets.

	Training Datasets				Testing Datasets			
# of API calls (in thousands)	6	9	12	15	3	6	9	12
# of nodes (in thousands)	11	16	21	28	5	10	15	20
# of VMs (in hundreds)	6	8	10	11	3	6	9	11
# of VNFs (in hundreds)	3	5	6	6	3	4	6	7

A. Effectiveness

To answer RQ1, we automatically reproduce in our testbed 10 attack scenarios that involve NFV management operations (as discussed in e.g., [44], [45], [47]) via a Bash script. Table II summarizes those scenarios, the most relevant operations and the vulnerabilities that are exploited through the requested operations for launching the attack. We evaluate the effectiveness of ProvTalk on these attacks as we know the precise ground truth (attack steps) published in the existing works[3] and the publicly reported vulnerabilities [49]. For all the 10 attacks,

[3]Most of these works (e.g., [44], [45]) focus on *security verification* (rather than provenance analysis), and thus we cannot directly compare our results with these solutions.

TABLE II: Attack scenarios used to evaluate the effectiveness of ProvTalk (the shaded rows indicate the incident and root cause are located at different levels).

Root Cause	Detected Incident	Most Relevant Management Operation Types	Vulnerability
Stealthy node injection into a VNFFG [44]	Unauthorized Access	Create-Port-Pair, Create-Port-Pair-Group, Update-Port-Chain	CVE-2017-2673
Bypassing anti-spoofing rules in network [29]	Unauthorized Access	Create-VNFFG, Create-Port, Create-VM, Update-Port	CVE-2015-5240
Firewall VNF misconfiguration	CPU DoS	Create-VNFFG, Update-VNFFG, Update-VNF	CVE-2017-7400
Malformed security group rule addition	Host Unavailability	Create-Security-Group, Create-Security-Group-Rule, Create-VM	CVE-2019-9735
Overlapping security group rule addition	Host Unavailability	Create-Security-Group, Create-Security-Group-Rule, Create-VM	CVE-2019-10876
Update of security group is not applied [45]	Data Leakage	Add-Security-Group, Start-VM, Delete-Security-Group-Rule	CVE-2015-7713
Neutron proper authorization failure [46]	Port Scanning	Create-Router, Create-Port, Create-VM	CVE-2014-0056
Wrong VLAN ID [47]	Data Leakage	Create-Network, Update-Network	Not specified
Failing to delete VMs in resize state	Disk DoS	Create-VM, Resize-VM, Delete-VM	CVE-2016-7498
Excessive VM creation on the same host [48]	Disk DoS	Create-VM	Not specified

we successfully trace back to the root cause of the reported incident using ProvTalk. Note that due to the novelty of NFV, very few papers exist on NFV-specific attacks or vulnerability exploits which limit our choice of attacks (although given the prevalence of NFV 5G telecommunication, we envision an extensive research on this matter in the future). Due to page limitation, we only chose 10 cases among those vulnerabilities to present in this paper. We showcase the effectiveness of our approach based on four cases: two vulnerabilities in Table II presented in the motivating example (first and second rows) and two cases presented below (third and seventh rows). We choose those scenarios because their incidents are detected at a different level from where the root cause operations are conducted, which make the analyses more challenging.

1) Cloud-level Alert, NFV-level Root cause: In this scenario (see Table II, third row), a VNFFG with a virtual firewall is protecting an end-to-end network service. The analyst receives a high CPU utilization alert from VM_b. Using the provenance graph generated by ProvTalk (Fig. 12), the analyst can observe that the VM generating the alert at cloud-level (VM_b) corresponds to VNF_1 (1). He/She can also see that VNF_1 was included in $VNFFG_z$ (2). Moreover, according to the provenance graph, $VNFFG_z$ was updated by the admin for chaining a preceding virtual firewall (3). However, shortly after adding the firewall, another user changed its configurations so that it will not filter syn-flood traffic (4). As updating the configuration to allow syn-flood traffic right after the insertion of a firewall in a VNFFG is not a routine behavior and is conducted by a non-admin user, the analyst can attribute this to a potential privilege escalation by that user.

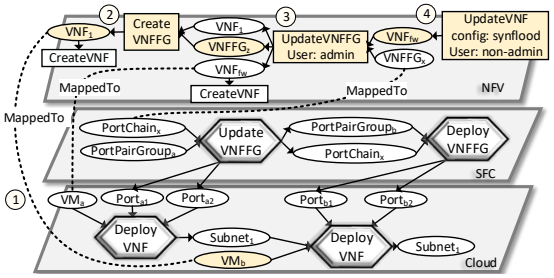


Fig. 12: Root cause of the CPU DoS Identified by ProvTalk.

2) NFV-level Alert, Cloud-level Root Cause: In this scenario (see Table II, seventh row), the analyst receives a port scanning alert from a virtual IDS service, VNF_{ids}. Using the multi-level provenance graph generated by ProvTalk (Fig. 13), he/she can easily identify that VNF_{ids} is associated with VM_b, which is created in $Subnet_1$ (1). He/She can also observe that an attacker from a different cloud tenant creates another port in $Subnet_1$ attaching it to $Router_1$ (2), and then creates a VM attached to this port (3). As this chain of operations shows that a different tenant could enter the network of the target resource, the analyst suspects that an attacker exploited a vulnerability in the NFV platform [50] to send malicious traffic to $Subnet_1$, which is detected by VNF_{ids}.

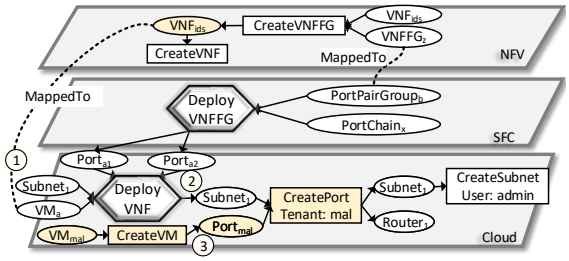
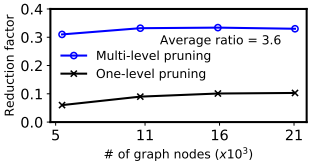


Fig. 13: Root cause of port scanning identified by ProvTalk.

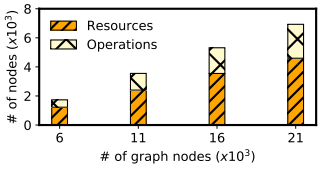
B. Graph Reduction Performance

To answer RQ2, we measure the reduction in the size of the provenance graph after applying our pruning and aggregation.

Comparing multi-level pruning with one-level pruning. To measure the effectiveness of multi-level pruning, we apply both multi-level pruning and one-level (NFV-level) pruning on provenance graphs of different sizes for the same incident. Fig. 14a shows that the reduction factor (i.e., the number of pruned nodes over that of all nodes) of multi-level pruning scheme is significantly higher (on average, by almost 3.6 times) than the one-level pruning scheme in all datasets. Furthermore, Fig. 14b shows that multi-level pruning removes a larger number of nodes representing resources than those representing operations. The reason is that most operations affect several resources at the same time, therefore, pruning an operation will automatically prune its many related resources.



(a) Multi-level vs. one-level effectiveness.

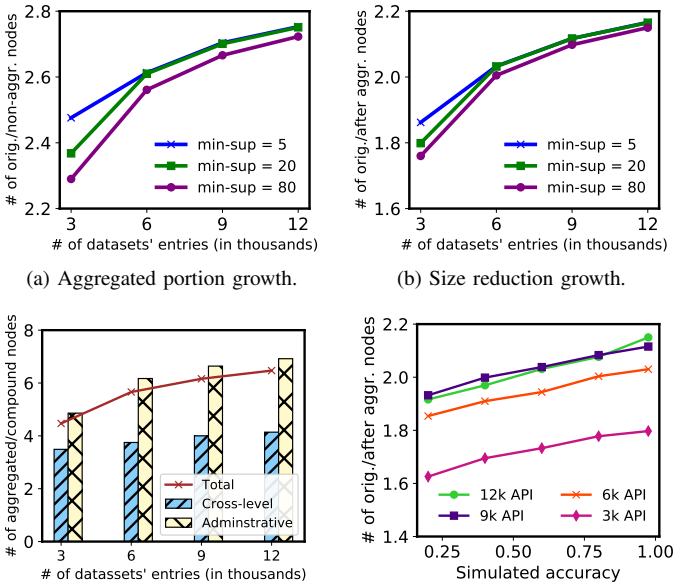


(b) Multi-level pruned resources vs. operations.

Fig. 14: Evaluating the effectiveness of multi-level pruning.

Aggregation. We measure the effectiveness of our aggregation for datasets of different sizes, while varying the minimum support values (aka *min-sup*) [34]. The min-sup value indicates

the minimum acceptable frequency of mined patterns in our training data, and thus, by varying the min-sup value from low to high, we can evaluate the performance for moderately to highly conservative scenarios respectively. Fig. 15a shows the ratio between the number of nodes in the original graph and that of the non-aggregated nodes. Fig. 15a shows that, on average, the number of nodes in the original provenance graph is around 2.6 times larger than the number of non-aggregated nodes and the reduction ratio increases with the size of the dataset. The reason is that, in larger provenance graphs, there usually exist more diverse sequences of operations, which increases the level of redundancy, and hence, allows for more aggregation. Moreover, smaller min-sup values lead to a lower ratio of non-aggregated nodes in smaller datasets, in contrast with larger datasets where reduction is similar for all measured values. The reason is that smaller datasets are more likely to lack the patterns mined via greater min-sup values, i.e., some of the most frequent mined patterns in the training data.



(a) Aggregated portion growth.

(b) Size reduction growth.

(c) Accuracy effect on size reduction.

(d) Aggregation power growth.

Fig. 15: Evaluating the effectiveness of aggregation.

Fig. 15b shows the ratio between the number of nodes in the original provenance graph and in the one after aggregation (which consists of non-aggregated and compound nodes). On average, our aggregation schemes decrease the size of the provenance graph by half, which shows the usefulness of our approach in providing a smaller provenance graph for investigation with the added higher-level semantics. Furthermore, the ascending trend of the curve shows that, as the size of the original provenance graph grows, our aggregation schemes is more effective in terms of reducing the graph size. One reason is that in larger provenance graphs there usually exist more overlapping compound nodes subsequently aggregated into a single compound node (Section III-C). In other words, as the size of the provenance graph grows, larger groups of nodes are aggregated into a single compound node, which leads to a greater reduction power. We can also see that lower min-sup values lead to a greater size reduction in smaller testing datasets, which is aligned with our results of Fig. 15a.

To gain more insights into the effectiveness of the aggregation, we evaluate the aggregation power of our approach (i.e., the number of aggregated nodes represented by compound nodes). Fig. 15c shows that the ratio between the total number of aggregated nodes and that of the compound nodes grows with the size of the provenance graph. On average, 5.69 aggregated nodes are represented by each compound node. To evaluate the performance of each aggregation scheme, we measure their contribution to reducing the size of the provenance graph separately. As we can see in Fig. 15c, for all datasets, the ratio between the number of compound nodes and the aggregated nodes they represent is higher for the administrative aggregation scheme. This can be partially explained by the merging we conduct on overlapping compound nodes related to the administrative behavior (Section III-C). Note that we do not merge cross-level compound nodes, as each of those nodes corresponds to a particular NFV-level operation and are labeled accordingly. Moreover, the ratio between aggregated and compound nodes under both schemes increases with the size of the provenance graph, which shows the better performance of both schemes for larger datasets.

Effect of accuracy on the reduction power. To evaluate how much the reduction power of ProvTalk may be affected by inaccuracies of our aggregation module, we simulate different accuracy values (of the frequent pattern mining step of our aggregation module) ranging from 20% up to 97%. To do so, we first manually verify the paths that are identified to be related to cross-level dependencies or administrative behaviors in our testing datasets. Then, we configure ProvTalk to randomly select a number of validated paths and leave them non-aggregated in the provenance graph. We determine the number of these *ignored* paths according to the desired accuracy value and the total number of paths extracted from the provenance graph. Fig. 15d shows the variation in the size reduction caused by different accuracy values in our four testing datasets. On average, the lowest accuracy value of 20% decreases the reduction power by almost 0.19 times only (i.e., preserving the total reduction power of around two times). It is also worth noting that since our aggregation module provides a more compact instance of the provenance graph (rather than discarding the nodes or the attack detection), low accuracy values would only affect the reduction power without losing vital information or generating false alarms.

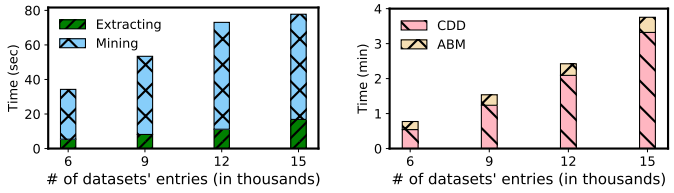
C. Efficiency

To answer RQ3, we measure the efficiency of ProvTalk based on our NFV testbed and real-world dataset.

1) Scalability Evaluation with NFV Testbed: To evaluate our approach in environments with a large number of diverse management API calls and deployed virtual services, we run ProvTalk on datasets generated in our NFV testbed (Table I).

Training time consumption. We measure the time required by the training stage of ProvTalk. Worthy to note that this is a one-time cost since the lower-level operations triggered by NFV-level API calls and administrative tasks do not change very frequently. Fig. 16a shows the time required by the CDM module for extracting the logged lower-level operations and running sequential pattern mining algorithm over the extracted operations. While the time required by both steps grows almost linearly with the size of the dataset, the total time does not exceed 80 seconds for the largest dataset. Fig. 16b depicts the

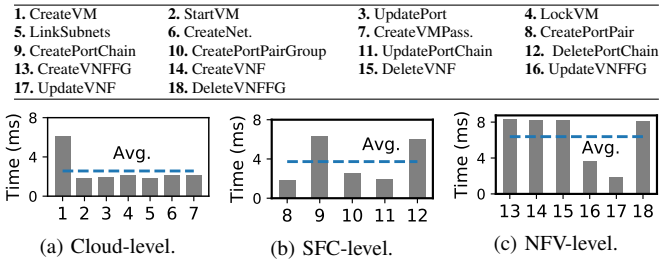
time required by the CDD and ABM modules. Although the required time increases with the size of the datasets, it remains under four minutes in total for the largest dataset.



(a) CDM module steps delay. (b) CDD and ABM modules delay.

Fig. 16: Evaluating the training stage overhead.

Runtime delay. We measure the delay caused by ProvTalk in the runtime execution of NFV management operations. Fig. 17 shows that, on average, ProvTalk adds around a two-millisecond delay for logging the information of most cloud-level operations. Longer delays (around eight milliseconds) mostly correspond to operations which also have a longer execution time (e.g., *CreateVM* has an execution time of above 10 seconds [51]). The average delay increases to 4 and 6 milliseconds for SFC-level and NFV-level operations (which also have a longer execution time). In summary, ProvTalk incurs a negligible overhead of around 0.04% additional delay to NFV management operations.



(a) Cloud-level. (b) SFC-level. (c) NFV-level.

Fig. 17: Runtime delay imposed to API calls at different levels.

Storage cost. We also evaluate the storage consumption of ProvTalk. Fig. 18a shows that the storage required by ProvTalk remains significantly lower than the logs generated by the platform. The reason is that the generated logs contain a large amount of information (e.g., listing deployed services) that are less important for security analysis, and thus are not included in the provenance graphs. Fig. 18b compares the storage required by ProvTalk and processed logs (containing only the information that we store as the attributes of nodes). Since the processed logs do not capture the relationships between operations necessary for the analyses, the size of the provenance graphs is greater than the processed logs, while it remains around 10 megabyte in the largest dataset with 20,000 API calls. To evaluate the storage cost of ProvTalk for environments with different available services, we measure the storage required for provenance graphs that are recorded at each level separately. Fig. 18c shows that the storage related to OpenStack API calls is significantly higher than the others, which can be caused by the higher number of API calls at this level (triggered both by cloud users and the platform after NFV-level operations). On the other hand, SFC operations consume the least amount of storage, due to the limited type and number of API calls that are attributed to this level.

CPU consumption. We measure how the rate of incoming API calls impacts the CPU usage. To simulate setups with different workloads, we vary the rate of received API calls per hour so that the time elapsed between every two consecutive API calls would be a fraction of the time elapsed between the same API calls in our real data. Fig. 18d shows that the CPU usage during the construction of the provenance graph increases almost linearly with the rate of received API calls. The rate of about 1,000 API calls per hour is comparable to the workload of our research cloud which incurs less than 1.25% CPU consumption. For enterprises with higher rates of API calls (e.g., 3,000 API calls per hour), the CPU consumption remains under 3.5%, which shows the scalability of our approach.

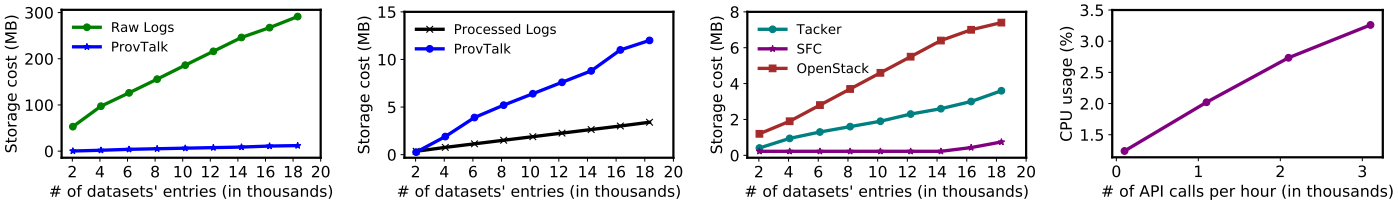
Comparing with OS-level provenance approaches. We also evaluate the benefit of tracking management API calls over their corresponding OS-level events. Table III shows the size of the OS-level provenance graph generated following each management API call as well as the provenance graph constructed by ProvTalk (two bottom rows). To build the OS-level provenance graph, we deploy a widely used open source tool, SPADE [52], in our controller host. We show an excerpt of the OS-level provenance graph generated after *CreateVNF* operation in Fig. 19. We can see that the OS-level provenance graphs would be impractically large in NFV environments (reaching millions of nodes and edges) even under a moderate workload of a research cloud with thousands of API calls issued during only a few days (Section V-C2). Additionally, in contrast with OS-level events, API management interfaces are usually accessible to a broad range of NFV/cloud customers (e.g., AWS CloudTrail) [53], which may include potentially malicious users. Therefore, by tracking management operations, ProvTalk enables an effective analysis on a vast group of security incidents in NFV, while avoiding the cost of large OS-level provenance graphs. Moreover, tracking management API calls provides a higher-level perception of changes in the NFV stack, and thus enables easier root cause identification.

TABLE III: Comparing the size of OS-level provenance graphs generated following each management API call with ProvTalk.

OS-level	CreateVNF	DeleteVNF	CreatePortChain	LockVM
# of nodes	2582	2548	13065	234
# of edges	12692	10116	38585	615
Storage (mb)	3.3	2.4	9.9	0.27
ProvTalk	**CreateVNF**	**DeleteVNF**	**CreatePortChain**	**LockVM**
# of nodes	16	16	35	2
# of edges	19	19	42	2

2) Experiments with Real-world Data: We evaluate the applicability of our approach by using 5 days of OpenStack logs collected from a real research cloud hosted at a major telecommunications vendor with hundreds of hosts and users. Although logs generated by the platform lack sufficient information [54], we build the provenance graph using those logs as we were not allowed to install API interceptors in this cloud (detailed in Appendix D). Those logs consist of 1,882 API calls affecting the deployment configuration of 354 VMs. Note that the number of the extracted API calls is smaller compared to our NFV testbed dataset, due to the unavailability of Tacker-level logs in that environment. Accordingly, ProvTalk generates a provenance graph of 2,157 nodes in 99.8 seconds which consumes only 2.53 megabytes storage. Thus, ProvTalk imposes negligible storage costs in real-world virtualized environments.

We conclude that unlike existing techniques localizing



(a) PG vs. raw logs storage cost. (b) PG vs. processed logs storage cost. (c) per level of PG storage cost. (d) PG construction CPU usage.

Fig. 18: Evaluating the storage and computation cost of ProvTalk (PG denotes provenance graph).

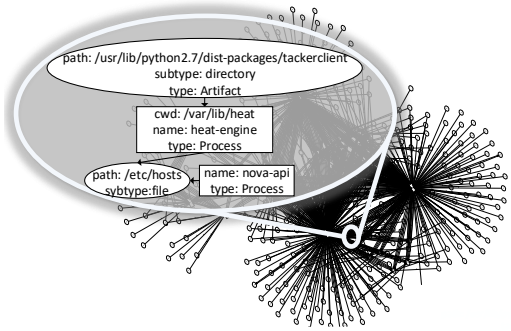
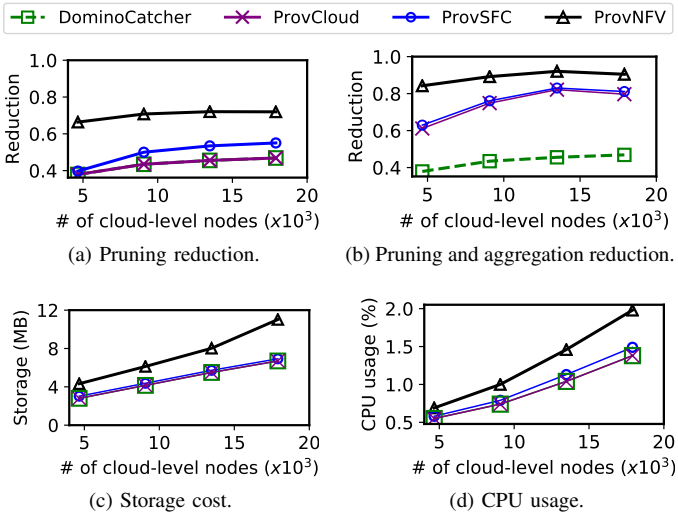


Fig. 19: Excerpt of the OS-level provenance graph generated by SPADE [52] upon issuing the *CreateVNF* operation. We magnified a subgraph to illustrate an example of the relationships between processes and data objects in an NFV controller.

failed components (e.g., [8], [7]), ProvTalk facilitates identifying the root cause activities, while incurring reasonable costs.

3) Comparing with DominoCatcher: We compare the overhead and performance of ProvTalk with DominoCatcher [11]. While DominoCatcher can only support the cloud level, ProvTalk can be applied to different virtual environments (e.g., cloud only, cloud and SFC, and cloud/SFC/NFV). Therefore, while all the experiments are based on cloud-level provenance graphs (the only level captured by DominoCatcher), we have applied ProvTalk under three scenarios, i.e., cloud-level only (*ProvCloud*), considering the cross-level dependencies between cloud/SFC levels (*ProvSFC*), and cloud/SFC/NFV levels (*ProvNFV*).

Size reduction. Fig. 20a shows the reduction of cloud-level provenance graphs of our testing datasets after applying the pruning schemes of DominoCatcher and ProvTalk (under aforementioned scenarios). The reduction factor is the number of pruned nodes over the total number of the cloud-level nodes. As Fig. 20a shows, the pruning schemes of ProvCloud and DominoCatcher have a similar reduction factor as neither of them captures cross-level dependencies. In contrast, ProvSFC and ProvNFV provide around 13% and 60% further reduction, respectively, which confirms the added benefits of our multi-level pruning schemes. In addition, Fig. 20b compares the total reduction enabled by both the pruning and aggregation techniques of ProvTalk with DominoCatcher. The administrative aggregation of ProvCloud enables around 71% further reduction compared with DominoCatcher. In constrast, ProvNFV has twice the reduction factor of DominoCatcher. Note that there is a smaller reduction caused by ProvSFC. The reason is that ProvSFC first prunes the portion of cloud-level nodes that corresponds to the routine administrative behavior, and thus a smaller number of nodes will remain to be aggregated by ProvSFC.



(a) Pruning reduction. (b) Pruning and aggregation reduction.

(c) Storage cost. (d) CPU usage.

Fig. 20: Comparing the effectiveness and overhead of ProvTalk and DominoCatcher [11].

Overhead. Fig. 20c shows that ProvNFV consumes only around 5 megabytes additional amount of storage compared with DominoCatcher. The additional storage cost increases as the number of NFV-level operations and resources grows (from 300 to 700 VNFs in our testing datasets). The storage required by DominoCatcher is similar to that by provenance graphs of ProvCloud, as both schemes capture only the dependencies between cloud-level operations. Note that ProvSFC requires an insignificant additional amount of storage (around 0.2 megabyte) compared with ProvCloud due to the small number of API calls related to SFC-level. Fig. 20d shows that, on average, ProvNFV consumes around 0.5% higher CPU resources than DominoCatcher and other implemented levels of ProvTalk due to capturing the dependencies across all levels. The CPU consumption of ProvSFC is close to DominoCatcher due to the lower number of operations at SFC-level. In conclusion, we can see that the low storage and CPU consumption of ProvSFC in addition to its effectiveness (Fig. 20a, 20b) demonstrate the benefit of our solution for virtual environments with only the cloud and SFC levels. Although ProvNFV has a slightly higher storage and CPU cost, it leads to significant reduction (twice that of DominoCatcher).

D. Correctness

To answer RQ4, we evaluate how completely and soundly ProvTalk can capture and process changes in the NFV stack.

Completeness. Our analysis shows that all operations directed through NFV management interfaces are passed as API calls to

the endpoint services to be applied to the NFV stack. To ensure completeness property, we deploy ProvTalk as a middleware attached to all those services so that it can capture all management API calls (i.e., 100% coverage). Table IV shows the number of unique types of management API calls (according to Tacker-OpenStack documentation [55]) that are issued to most commonly used services. Moreover, we conduct an exhaustive study of all database tables in the NFV platform and devise an entity-relationship (ER) model to ensure all cross-level relationships are captured by constructed provenance graphs.

TABLE IV: The number of types of management API calls.

Services	Tacker	Nova	Neutron	Glance	Swift	Heat
Unique API calls	73	313	251	31	16	58
Coverage (%)	100	100	100	100	100	100

Soundness. ProvTalk stores the interception time of API calls at all services to preserve the temporal order of events during backtracking on the provenance graph. Moreover, our pruning scheme preserves the soundness property by leveraging the cross-level dependencies as follows. At the same level as where the incident is detected, ProvTalk prunes only the nodes identified to be irrelevant according to the defined pruning policies (which is consistent with [11], [28]). Our multi-level pruning leverages cross-level dependencies to only prune the nodes at other levels that correspond to the irrelevant nodes identified by the existing techniques. Also, our aggregation schemes ensure the soundness property by preserving all the relationships between the aggregated nodes and the rest of the provenance graph. In other words, all edges pointing to/from each group of aggregated nodes will point to/from their representative compound node, and the compound nodes can be expanded to show the aggregated nodes in the original form.

E. User Studies

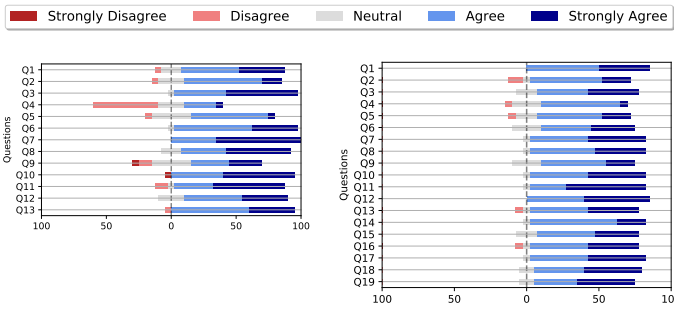
To answer RQ5, we conducted two user studies[4] based on standard practices [56] in which the participants have to identify the root cause of an incident using ProvTalk. In our first study (static outputs), participants are provided with the outputs of ProvTalk that we obtained in advance. In our second study (live interaction), participants could directly interact with ProvTalk, trigger each module, and/or analyse the output of customized queries (e.g., nodes related to operations requested by non-admin users).

TABLE V: Statistics of participants. PG means provenance-based analysis. (A), (L) and (N) mean advanced, little and no knowledge, respectively. Numbers of participants are shown in the first row of the tables related to each study.

1st (Static outputs)	Industry (14)					Academia (7)	
Background (NFV-PG)	A-A	A-L	A-N	L-L	L-N	A-L	L-L
Participants (%)	14	19	5	19	9	5	29
Scores	3.89	4.19	4	4.1	4.38	3.92	4.26
2nd (Live interaction)	**Industry (6)**			**Academia (11)**			
Background (NFV-PG)	A-A	A-L	A-N	A-L	L-L	L-N	N-N
Participants (%)	12	17	6	35	12	12	6
Scores	4.78	3.91	4.36	4.14	4.44	4.28	4.05

Participants. To conduct both studies, we recruited participants from a telecommunication industrial organization and graduate students working in cybersecurity from our university. Table V shows the statistics of participants in each study and the average score for all provided statements.

In the beginning of both studies, we provided a brief review of an attack story (our motivating example in Section II-C), and asked the participants to express their level of agreement with our provided statements. Our web-based platform showed the outputs generated by ProvTalk in split-screen together with the statements. Appendix E shows the list of statements common between our two studies. Table VI shows the additional statements specific to our second study (live interaction). Participants could express their agreement level by choosing either *Strongly agree*, *Agree*, *Neutral*, *Disagree* or *Strongly disagree*. To quantify the results, we calculated the average scores by assigning an integer between one and five to each option (five represents *Strongly agree* and one represents *Strongly disagree*).



(a) Based on the first study. (b) Based on the second study.

Fig. 21: Participants' agreement with the statements.

Results. According to Table V, ProvTalk achieves scores above 3.8 among all groups. Fig. 21 shows the distribution of participants' agreement with statements of our studies. Based on our results, pinpointing the root cause is challenging for most participants using the multi-level provenance graph (Q1-3). The results from Q4 and Q5 demonstrate the advantage of our multi-level pruning over the existing schemes in making the provenance graph easier to understand. The results (Q6-9) affirm the analysis becomes easier with our aggregation schemes by decreasing the size of the provenance graph and assigning expressive labels to compound nodes (i.e., adding semantics). Additionally, most participants find the expansion of compound nodes helpful for detailed analyses (Q10). The generated textual description facilitates the analyses for most participants (Q11-12), and they can easily associate the descriptions with their corresponding parts of the provenance graph for detailed analyses (Q13). We show the average quantified score for the aforementioned statements in Appendix E.

Added benefit of live interaction. There is a slight difference between agreement levels in the two studies, with a more significant gap in some cases, especially Q4 and Q9. We believe that the improved results in our second study (live interaction) is due to the added capabilities of participants to directly play with all modules or make customized queries (as shown by Q14-15). Additionally, interactive features such as zooming and node dragging render the effectiveness of ProvTalk more visible to participants (as shown by Q16-Q19).

Based on the results of our studies, we conclude that ProvTalk can effectively facilitate the analysis for both expert

[4]Those studies have been approved by Research Ethics/Office of Research of our university.

and non-expert users.

TABLE VI: Statements specific to our second study. Scores are between one and five (score five represents *Strongly agree*).

Statement	Code	Score
Live interaction with ProvTalk enabled searching for certain relationships between nodes (e.g., those satisfying given properties), which made the analysis easier.	Q14	4.17
Live interaction with ProvTalk enabled highlighting certain relationships between nodes (e.g., those satisfying given properties), which made the analysis easier.	Q15	4.17
Hovering over nodes shows all of their properties, which enables easier access to more information.	Q16	4.23
Zooming feature enhanced the visibility of different nodes.	Q17	4.41
Node dragging feature made understanding the relationships between nodes easier.	Q18	4.35
Aggregation/compression of any node of my choice made the analysis easier.	Q19	4.37

VI. Discussion

We discuss limitations and future directions of ProvTalk.

Integration with system-level solutions. ProvTalk can potentially be integrated with system-level provenance solutions to provide mutual benefits (e.g., ProvTalk can guide system-level solutions to focus on specific resources, whereas the latter can corroborate the findings of ProvTalk with low-level details). Moreover, we envision that the system calls captured within virtual resources (e.g., using SPADE system [52]) can be integrated with the provenance graph generated by ProvTalk in a manner consistent with previous works (e.g., [15], [14]).

Other platforms. While we focus on OpenStack-Tacker platform in this work, our approach can be extended to other NFV platforms with an initial effort of adapting to their management operations. For instance, in AWS cloud, we could map the NFV-level of our provenance model to AWS CloudFormation services [57]. Our model can also be mapped to container-based platforms such as Kubernetes-Tacker [58]. Additionally, as we show in Section V-D, the only engineering effort required for collecting the information of our interest is to plug in API interceptors suitable for the studied platforms (e.g., AWS Lambda in AWS cloud [59]).

More complex modeling approaches. Since ProvTalk focuses on assisting, instead of replacing human analysts, we are less concerned with the accuracy of our aggregation approach. Nevertheless, in our future work, we will investigate the effect of using different learning and embedding techniques (e.g., [60], [61]) on the reduction power. Moreover, we plan to extend ProvTalk to automatically detect anomalous behavior [12], [62] and translate it into a human-readable text [63]. Additionally, we will apply adversarial machine learning to study potential attacks (e.g., adversarial event sequences) against our solution.

Limitations around limited coverage and applicability. ProvTalk does not cover system-level events inside individual virtual resources, and it can potentially be integrated with system-level provenance solutions in order to address the scalability issue of the latter in a complex NFV system. To maintain the applicability of ProvTalk, analysts will need to periodically update the models trained by CDM and ABM modules. Finally, ProvTalk is designed to facilitate the investigation of management operations by human analysts instead of replacing them.

VII. Related Work

NFV incident investigation. Existing role/permission-based techniques (e.g., [64]) focus on incident prevention, and cannot be applied to investigate the attacks bypassing such techniques [6]. Several incident investigation solutions have been proposed for NFV platforms (e.g., [7], [8]) to enable locating malfunctioning components through alert correlation techniques. ChainGuard [65] and SFC-Checker [66] verify the correct forwarding behavior of service function chains. vSFC [67] enables identifying a wide range of security threats (e.g., packet injection attacks). Unlike ProvTalk, these solutions do not directly identify the root cause operations.

Provenance-based solutions. Provenance-based security analysis has been extensively studied in the literature [24], [68], [28], [69], [70], [15], [62], [18]. The authors in [28], [71], [69] improve the capture mechanism by building the provenance graphs based on the information captured by Linux Security Module hooks. LPM [23] leverages data provenance to ensure authenticated system communications. CamQuery [28] increases the efficiency of provenance analyses through tracing both userspace and in-kernel executions. As a management-level solution, ProvTalk may work in tandem and complement those system-level techniques.

Past frameworks for layered provenance (e.g., [13], [14], [15]) integrate application logs into the OS-level provenance of hosts to enable more accurate analysis by removing irrelevant dependencies. The authors in [70] and [12] leverage OS-level provenance to triage alerts and detect exploited instances of a program instead of root cause analysis. There exist network provenance-based techniques [72], [73] focusing on network traffic and reference packet events instead of management operations initiated by users (which is the focus of ProvTalk). ProvThings [9] proposes a provenance-based approach for auditing the IoT applications across different devices. In SDN environments, ForenGuard [10] provides flow-level forensics and ProvSDN [16] monitors the access to sensitive data for unprivileged applications. In [74], the authors identify the absence of events in distributed systems. In [62], the authors produce a behavioral model of distributed applications to identify anomalous events in a cluster. Bates et al. [75] propose a provenance-based access control mechanism ensuring cloud storage security. The authors in [76] propose a tenant-aware solution to enhance OpenStack access control mechanism. DominoCatcher [11] tracks cloud management operations in single-level provenance graphs. Our experiments show that our approach can reduce the size of cloud-level provenance graphs twice as much as DominoCatcher does (see Section V-C3 for detailed results). Moreover, unlike our work, none of these solutions can support tracking information flow to/from different visualizations of the same resources that we face in NFV. Finally, summarization solutions (e.g., [17]) abstract user behaviors using audit logs. However, such abstraction is already intrinsically provided by NFV stack. ProvTalk leverages the distinct visualizations of resources that already exist at different levels of the NFV stack to infer the semantics of lower-level events and facilitate the analyses.

VIII. Conclusion

In this paper, we presented ProvTalk, the first multi-level provenance solution for NFV. ProvTalk leveraged data

provenance concept to find the management operations leading to attacks in NFV platforms and provided efficient pruning, aggregation and translation mechanisms for users to pinpoint the root cause of security incidents. We integrated ProvTalk to Tacker-OpenStack and demonstrated the efficacy of our approach based on real attack scenarios. Moreover, based on our experiments on performance and storage cost, our system substantially reduces the size of the provenance graph with insignificant runtime and storage overhead. Finally, our user studies results show that our approach remarkably facilitates the identification of root cause of security incidents.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their valuable comments and suggestions. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada and Ericsson Canada under the Industrial Research Chair (IRC) in SDN/NFV Security.

REFERENCES

[1] VMware, "VMware vSphere," 2020. Accessed May 16, 2021. https://www.vmware.com/ca/products/vsphere.html.

[2] OpenStack, "Open Source Cloud Computing Infrastructure." Accessed February 04, 2021. https://www.openstack.org/.

[3] European Telecommunications Standard Institute, "Network Functions Virtualisation (NFV); Architectural Framework," Tech. Rep. ETSI GS NFV 002, V1.2.1, 2014.

[4] A. F. Murillo, S. J. Rueda, L. V. Morales, and Á. A. Cárdenas, "SDN and NFV Security: Challenges for Integrated Solutions," in *Guide to Security in SDN and NFV*, pp. 75–101, Springer, 2017.

[5] F. Reynaud, F.-X. Aguessy, O. Bettan, M. Bouet, and V. Conan, "Attacks against Network Functions Virtualization and Software-Defined Networking: State-of-the-art," in *NetSoft*, pp. 471–476, IEEE, 2016.

[6] R. Hat, "CVE-2020-12689: Keystone Credential Modification," 2020. Accessed February 04, 2021. https://access.redhat.com/security/cve/cve-2020-12689/.

[7] C. Sauvanaud, K. Lazri, M. Kaâniche, and K. Kanoun, "Anomaly Detection and Root Cause Localization in Virtual Network Functions," in *ISSRE*, pp. 196–206, IEEE, 2016.

[8] D. Kushnir and M. Goldstein, "Causality Inference for Failures in NFV," in *INFOCOM WKSHPS*, pp. 929–934, IEEE, 2016.

[9] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and Logging in the Internet of Things," in *NDSS*, 2018.

[10] H. Wang, G. Yang, P. Chinprutthiwong, L. Xu, Y. Zhang, and G. Gu, "Towards Fine-grained Network Security Forensics and Diagnosis in the SDN Era," in *CCS*, pp. 3–16, ACM, 2018.

[11] A. Tabiban, Y. Jarraya, M. Zhang, M. Pourzandi, L. Wang, and M. Debbabi, "Catching Falling Dominoes: Cloud Management-Level Provenance Analysis with Application to OpenStack," in *CNS*, pp. 1–9, IEEE, 2020.

[12] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. Gunter, *et al.*, "You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis," in *NDSS*, 2020.

[13] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "MPI: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning," in *USENIX Security*, pp. 1111–1128, 2017.

[14] K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. L. MacLean, D. W. Margo, M. I. Seltzer, and R. Smogor, "Layering in Provenance Systems," in *USENIX ATC*, 2009.

[15] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis," in *NDSS*, 2020.

[16] B. E. Ujcich, S. Jero, A. Edmundson, Q. Wang, R. Skowyra, J. Landry, A. Bates, W. H. Sanders, C. Nita-Rotaru, and H. Okhravi, "Cross-App Poisoning in Software-Defined Networking," in *CCS*, pp. 648–663, ACM, 2018.

[17] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics," in *NDSS*, 2021.

[18] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan, "HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows," in *IEEE S&P*, pp. 1137–1152, 2019.

[19] OpenStack, "Tacker Documentation." Accessed February 04, 2021. https://docs.openstack.org/tacker/.

[20] M. Li, W. Zang, K. Bai, M. Yu, and P. Liu, "MyCloud: Supporting User-Configured Privacy Protection in Cloud Computing," in *ACSAC*, pp. 59–68, ACM, 2013.

[21] N. Schear, P. T. Cable II, T. M. Moyer, B. Richard, and R. Rudd, "Bootstrapping and Maintaining Trust in the Cloud," in *ACSAC*, pp. 65–77, ACM, 2016.

[22] R. Hasan, R. Sion, and M. Winslett, "The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance," in *FAST*, pp. 1–14, 2009.

[23] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer, "Trustworthy Whole-System Provenance for the Linux Kernel," in *USENIX Security*, pp. 319–334, 2015.

[24] S. T. King and P. M. Chen, "Backtracking Intrusions," in *SOSP*, pp. 223–236, 2003.

[25] OpenStack, "OSSA-2015-018: Neutron Firewall Rules Bypass Through Port Update." Accessed February 04, 2021. https://security.openstack.org/ossa/OSSA-2015-018.html.

[26] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, *et al.*, "PROV-DM: The PROV Data Model," Tech. Rep. REC-prov-dm-20130430, W3C, 2013. W3C Recommendation. Accessed February 4, 2021. https://www.w3.org/TR/prov-dm/.

[27] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware Storage Systems," in *USENIX ATC*, pp. 43–56, 2006.

[28] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, and M. Seltzer, "Runtime Analysis of Whole-System Provenance," in *CCS*, pp. 1601–1616, ACM, 2018.

[29] A. Tabiban, S. Majumdar, L. Wang, and M. Debbabi, "PERMON: An Openstack Middleware for Runtime Security Policy Enforcement in Clouds," in *CNS*, pp. 1–7, IEEE, 2018.

[30] Opentsack, "API Reference." Accessed February 04, 2021. https://developer.openstack.org/api-ref/.

[31] Openstack, "NFV API reference." Accessed February 04, 2021. https://developer.openstack.org/api-ref/nfv-orchestration/v1/.

[32] OpenStack, "Service Function Chaining Extension for OpenStack." Accessed February 04, 2021. https://docs.openstack.org/networking-sfc/latest/.

[33] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime Provenance-based Detector for Advanced Persistent Threats," in *NDSS*, 2020.

[34] J. Wang and J. Han, "BIDE: Efficient Mining of Frequent Closed Sequences," in *ICDE*, pp. 79–90, IEEE, 2004.

[35] Verizon, "Verizon Network Infrastructure Planning," tech. rep., 2016. Accessed February 04, 2021. https://m.iotone.com/files/pdf/vendor/Verizon_SDN-NFV_Reference_Architecture.pdf.

[36] OpenStack, "OpenStack Foundation Report, Accelerating NFV Delivery with OpenStack," tech. rep. Accessed February 04, 2021. https://object-storage-ca-ymq-1.vexxhost.net/swift/v1/6e4619c416ff4bd19e1c087f27a43eea/www-assets-prod/marketing/OpenStack-NFV-A4.pdf.

[37] WSGI, "Python WSGI Middleware." Accessed February 04, 2021, https://wsgi.readthedocs.io/en/latest/libraries.html.

[38] Y. Luo, W. Luo, T. Puyang, Q. Shen, A. Ruan, and Z. Wu, "OpenStack Security Modules: A Least-invasive Access Control Framework for the Cloud," in *IEEE CLOUD*, pp. 51–58, 2016.

[39] Neo4j, "Neo4j Graph Platform." Accessed February 04, 2021. https://neo4j.com/.

[40] Neo4j, "Cypher Query Language." Accessed February 04, 2021. https://neo4j.com/developer/cypher-query-language/.

[41] Neo4j, "Export to json." Accessed February 04, 2021, https://neo4j.com/labs/apoc/4.1/export/json/.

[42] A. Gatt and E. Reiter, "SimpleNLG: A Realisation Engine for Practical Applications," in *ENLG*, pp. 90–93, 2009.

[43] Cytoscape, "Cytoscape: Open Source Platform for Complex Networks." Accessed February 04, 2021. https://cytoscape.org/.

[44] S. L. Thirunavukkarasu, M. Zhang, A. Oqaily, G. S. Chawla, L. Wang, M. Pourzandi, and M. Debbabi, "Modeling NFV Deployment to Identify the Cross-level Inconsistency Vulnerabilities," in *CloudCom*, pp. 167–174, IEEE, 2019.

[45] S. Majumdar, Y. Jarraya, M. Oqaily, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi, "LeaPS: Learning-based Proactive Security Auditing for Clouds," in *ESORICS*, pp. 265–285, Springer, 2017.

[46] Y. Wang, T. Madi, S. Majumdar, Y. Jarraya, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi, "TenantGuard: Scalable Runtime Verification of Cloud-Wide VM-Level Network Isolation," in *NDSS*, 2017.

[47] S. Bleikertz, C. Vogel, T. Groß, and S. Mödersheim, "Proactive Security Analysis of Changes in Virtualized Infrastructures," in *ACSAC*, pp. 51–60, ACM, 2015.

[48] T. Madi, M. Zhang, Y. Jarraya, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi, "QuantiC: Distance Metrics for Evaluating Multi-Tenancy Threats in Public Cloud," in *CloudCom*, pp. 163–170, IEEE, 2018.

[49] C. Details, "OpenStack Vulnerabilities." Accessed February 04, 2021, https://www.cvedetails.com/vulnerability-list/vendor_id-11727/Openstack.html.

[50] OpenStack, "OSSA-2014-008: Routers can be cross plugged by other tenants." Accessed February 04, 2021, https://security.openstack.org/ossa/OSSA-2014-008.

[51] S. Majumdar, G. S. Chawla, A. Alimohammadifar, T. Madi, Y. Jarraya, M. Pourzandi, L. Wang, and M. Debbabi, "Prosas: Proactive security auditing system for clouds," *IEEE Transactions on Dependable and volume= , number=, pages=, Secure Computing*, 2021.

[52] A. Gehani and D. Tariq, "Spade: Support for provenance auditing in distributed environments," in *Middleware*, 2012.

[53] O. Kodym, L. Kubáč, and L. Kavka, "Risks associated with logistics 4.0 and their minimization using blockchain," *Open Engineering*, vol. 10, no. 1, pp. 74–85, 2020.

[54] S. Majumdar, A. Tabiban, Y. Jarraya, M. Oqaily, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi, "Learning Probabilistic Dependencies among Events for Proactive Security Auditing in Clouds," *Journal of Computer Security*, vol. 27, no. 2, pp. 165–202, 2019.

[55] Tacker-OpenStack API, Accessed May 10, 2021, 2021. https://docs.openstack.org/api-ref/.

[56] A. Assila, H. Ezzedine, *et al.*, "Standardized usability questionnaires: Features and quality focus," *Electronic Journal of Computer Science and Information Technology: eJCIST*, 2016.

[57] A. W. Services, "Mapping AWS Services to the NFV Framework," 2021. Accessed May 16, 2021. https://aws.amazon.com/cloudformation/.

[58] OpenStack, "Kubernetes as VIM in Tacker," 2021. Accessed May 16, 2021. https://specs.openstack.org/openstack/tacker-specs/specs/queens/Kubernetes-as-VIM.html.

[59] A. W. Services, "Amazon virtual private cloud." Accessed February 04, 2021, https://aws.amazon.com/vpc.

[60] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[61] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning Distributed Representations of Graphs," *arXiv preprint, arXiv:1707.05005*, 2017.

[62] W. U. Hassan, L. Aguse, N. Aguse, A. Bates, and T. Moyer, "Towards Scalable Cluster Auditing Through Grammatical Inference over Provenance Graphs," in *NDSS*, 2018.

[63] A. Dwaraki, S. Kumary, and T. Wolf, "Automated Event Identification from System Logs Using Natural Language Processing," in *2020 International Conference on Computing, Networking and Communications (ICNC))*, IEEE, 2020.

[64] "Openstack wsgi." Accessed February 04, 2021, Middleware Architecture. https://docs.openstack.org/keystonemiddleware/latest/middlewarearchitecture.

[65] M. Flittner, J. M. Scheuermann, and R. Bauer, "ChainGuard: Controller-Independent Verification of Service Function Chaining in Cloud Computing," in *NFV-SDN*, pp. 1–7, IEEE, 2017.

[66] 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*, pp. 134–140, IEEE, 2016.

[67] X. Zhang, Q. Li, J. Wu, and J. Yang, "Generic and Agile Service Function Chain Verification on Cloud," in *IWQoS*, pp. 1–10, IEEE/ACM, 2017.

[68] D. J. Pohly, S. E. McLaughlin, P. D. McDaniel, and K. R. B. Butler, "Hi-Fi: Collecting High-Fidelity Whole-System Provenance," in *ACSAC*, pp. 259–268, 2012.

[69] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical Whole-System Provenance Capture," in *SoCC*, pp. 405–418, ACM, 2017.

[70] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage," in *NDSS*, 2019.

[71] X. Han, T. Pasquier, T. Ranjan, M. Goldstein, and M. Seltzer, "FRAPpuccino: Fault-Detection Through Runtime Analysis of Provenance," in *HotCloud*, 2017.

[72] A. Chen, Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo, "The good, the bad, and the differences: Better network diagnostics with differential provenance," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 115–128, 2016.

[73] A. Chen, A. Haeberlen, W. Zhou, and B. T. Loo, "One primitive to diagnose them all: Architectural support for internet diagnostics," in *Proceedings of the Twelfth European Conference on Computer Systems*, pp. 374–388, 2017.

[74] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo, "Diagnosing Missing Events in Distributed Systems with Negative Provenance," in *ACM SIGCOMM*, pp. 383–394, 2014.

[75] A. Bates, B. Mood, M. Valafar, and K. R. B. Butler, "Towards Secure Provenance-based Access Control in Cloud Environments," in *CODASPY*, pp. 277–284, 2013.

[76] D. Nguyen, J. Park, and R. Sandhu, "Adopting Provenance-based Access Control in OpenStack Cloud IaaS," in *NSS*, pp. 15–27, Springer, 2014.

APPENDIX A
PROVENANCE MODEL

We provide additional description about our model in Table VII. We show a summary of nodes defined in our provenance model, their related NFV concepts and their mapping into the PROV-DM model. Subtypes refine nodes classification with respect to NFV concepts.

APPENDIX B
CROSS-LEVEL DEPENDENCY DISCOVERY

To provide additional detail for our cross-level dependency discovery, we provide Algorithm 1, which elaborates the steps of the CDD module. For every NFV-level operation, CDD identifies and tags the nodes representing its affected resource, as well as the lower-level node connected through a *MappedTo* edge. It also tags the node representing the operation connected to the lower-level tagged node and removes that operation from the mined sequence (Line 1-3). Then, it starts an iteration over the remaining mined operations where it searches for paths of the type *(op2)←(res)←(op1)*, where *op1* is a tagged node and the operation type stored at node *op2* is in the mined sequence. It tags *res* and *op2* nodes, and removes *op2* from the mined sequence (Line 8-9). At the end of the iteration, based on the pre-specified threshold values and the number of operations

TABLE VII: Mapping of the common concepts in NFV stack to the PROV-DM Model.

NFV Concept	Description	PROV-DM	Subtype
NFV Client	Customers of network services with specific privileges and service requirements.	Agent	NFV user admin, other tenants.
Cloud Tenant	A group of users owning an isolated set of virtual resources.	Agent	Tenant Admin, other tenants
NFV Client Operation	Management API calls for updating the life-cycle or state of network services.	Activity	Create-VNF, Update-VNFFG, etc
Cloud Provider Operation	Management API calls for updating the life-cycle or state of virtual resources.	Activity	Create-VM, Update-Port, etc
Network service component	The states of individual or chain of services such as IDS.	Entity	VNF, VNFFG, etc.
Network service configuration	The states of a deployed network service configuration, e.g., virtual firewall rules.	Entity	VNF descriptors, etc.
Cloud Resource	The states of a virtual infrastructure resource, e.g., a running/stopped VM.	Entity	VMs, virtual ports, etc.
Cloud Resource Configuration	The states of a virtual infrastructure configuration, e.g., VM virtual hardware.	Entity	Security groups, Flavors, etc.
Input for changing configurations	An input data causing a change to the configuration state.	Entity	Security group rules, etc.

TABLE VIII: Statements common in our two studies. To quantify the results, we convert participants' agreement level to scores between one and five (score five represents *Strongly agree*). ScoreS and ScoreL represent the scores of our first and second studies, respectively.

Module	Statement	Code	ScoreS	ScoreL
Multi-level Provenance	Given the incident and the provenance graph at each level, it is almost impossible to find the root cause.	Q1	4.2	4.41
	Given the incident alert and the connections between levels, I could find the path from the incident to the root cause.	Q2	3.85	3.94
	It is time-consuming to recognize the root cause among all graph nodes.	Q3	4.4	4.24
Pruning	One-level pruning made it easier to identify the attack-related graph nodes.	Q4	2.85	3.71
	Multi-level pruning made it easier to identify the attack-related graph nodes (w.r.t one-level pruning).	Q5	3.65	3.94
Aggregation	Cross-level aggregation made understanding the relationship between nodes at different levels easier.	Q6	4.3	4.11
	The graph seems less complex after cross-level aggregation.	Q7	4.65	4.41
	The labels inside the compound nodes are helpful in understanding the provenance graph.	Q8	4.35	4.35
	It was easier to find the root cause after aggregating and labeling administrative behavior-related nodes.	Q9	3.6	4
	Expanding the compound nodes can provide useful details in an on-demand basis.	Q10	4.4	4.41
Rule-based Translation	Seeing this generated textual description would have made identifying the root cause much easier.	Q11	4.3	4.58
	The generated text is similar to what was described about the attack story in the beginning.	Q12	4.15	4.52
	It is easy to map the summary with the provenance graph.	Q13	4.25	4.23

remaining in the sequence, CDD aggregates the tagged nodes while labeling them with either the corresponding NFV-level operation (line 10-11), or adding *partially-mismatched* prefix to the label (line 12-13), or it does not aggregate them (line 14-15). Note that the analyst can configure the threshold values based on their platform. For instance, environments with a higher variety of services would have more various sets of lower-level operations, and therefore, he/she should provide higher threshold values in those cases.

Algorithm 1 Cross-level Dependency Discovery

```
Inputs:
    graph ← Multi-level Provenance Graph
    MinedOps ← Mined Sequences of Operations
    nfvAPIs ← NFV API Calls
    Th_l, Th_h ← High and Low Threshold Values
Outputs:
    Provenance graph with aggregated nodes
1:  foreach nfvAPI_i ∈ nfvAPIs do
        %the resource and operation connected with MappedTo edge
2:      First_Operations,graph ← MapTagger(graph, nfvAPI_i)
3:      LeftOps ← OperationRemover(MinedOps, first_Operations)
4:      while iteration < MinedOps_len do
5:          if iteration > MinedOps_len then
6:              break
7:          else
                %tagging other connected operations and resources
8:              graph, FoundOperations ← Tagger(graph, LeftOps)
9:              LeftOps ← OperationRemover(LeftOps, FoundOperations)
                %finalizing or removing tags from aggregation candidates
10:     if LeftOps_len < Mined_len*Th_l then
11:         graph←Aggregator(graph)
12:     if LeftOps_len ∈ [MinedOps_len*Th_l, MinedOps_len*Th_h] then
13:         graph ← MismatchAggregator(graph)
14:     if LeftOps_len> MinedOps_len*Th_h then
15:         graph←TagUndoer(graph)
16: return graph
```

Appendix C
Graphical User Interface

We show a screenshot of our interface with the magnified excerpt of a provenance graph in Fig. 22. A brief summary of recent incident alerts is displayed in our interface (not shown in Fig. 22). By selecting each incident, users can see the provenance graph corresponding to each incident, and then invoke the pruning and aggregation options. Users can click on the compound nodes (blue hexagon in Fig 22) to expand them and visualize the aggregated nodes (appeared in the blue rectangle). Users can also examine the information about the operations and affected resources (the green box in Fig. 22), by hovering over their corresponding nodes.

Appendix D
Log Processor

To demonstrate that our approach can potentially be applied to environments where intercepting API calls may not be feasible, we implement a log processor for extending ProvTalk to work with infrastructure logs (e.g., logs generated by *Neutron-Server* services by default). A challenge is that some details of the API calls are not captured by their corresponding log entries [54]. For instance, *Neutron-Server* does not log all resources affected by most API calls such as the virtual subnet attached to a created port. To collect some missing details, we devise methods for automatically inferring them through correlating the log entries of different services based on the ID of resources as index. For example, by correlating *Neutron-Server* and *DHCP-agent* logs, we extract the virtual subnet that is attached to a port. We implement this method as a log processor module using Python to automatically extract and correlate our required information from different services' logs. This additional module can potentially extend the scope

of application for ProvTalk to cover other virtual environments as long as there exists the logging capability.

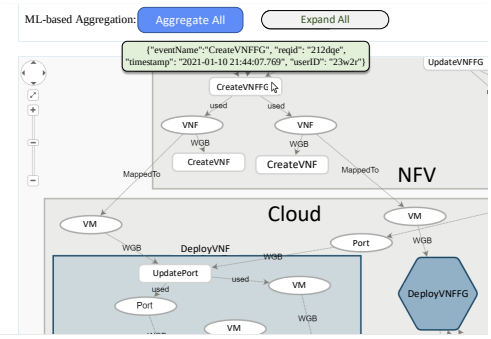


Fig. 22: Example screenshot of ProvTalk showing the aggregated and expanded cloud-level nodes and the information shown while hovering a node.

APPENDIX E
DETAILED USER PERCEPTION

Table VIII shows the list of the statements common between our two user studies as well as the average quantified agreement level of participants with each statement.