Blackbox Security Auditing for Network Functions Virtualization (NFV)

Momen Oqaily

A THESIS

IN THE DEPARTMENT OF

CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

Doctor of Philosophy (Information and Systems)

AT CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

May 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By:	**Momen Oqaily**
Entitled:	**Blackbox Security Auditing for Network Functions Virtualization (NFV)**

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

Doctor of Philosophy (Information and Systems)

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

Signed by the Final Examining Committee:

Dr. Mustafa K. Mehmet Ali	Chair
Dr. Xinwen Fu	External Examiner
Dr. Abdelhak Bentaleb	External to Program
Dr. Chadi Assi	Examiner
Dr. Suryadipta Majumdar	Examiner
Dr. Lingyu Wang and Dr. Mourad Debbabi	Thesis Supervisor

Approved by

Dr. Jun Yan, Graduate Program Director

26/04/2024

Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Blackbox Security Auditing for Network Functions Virtualization (NFV)

Momen Oqaily, Ph.D.

Concordia University, 2024

Over the past decade, Network Functions Virtualization (NFV) has revolutionized networking by leveraging virtualization to separate Network Functions (NFs) from dedicated physical hardware. However, this architecture introduces unique security risks, such as stealthy attacks causing discrepancies between tenant-level NF specifications and cloud provider-level deployment. To safely utilize NFV, robust security auditing mechanisms are crucial to ensure compliance and detect breaches. Yet, existing methods face challenges: NFV tenants have limited access to cloud infrastructure, and providers are hesitant to share data due to confidentiality concerns. Relying solely on providers for auditing may overlook tenant-specific requirements and legitimate modifications by attackers. Furthermore, current solutions often require unrealistic infrastructure modifications. This thesis introduces novel auditing solutions for both tenants and providers of NFV, addressing these limitations. Firstly, an interactive anonymization tool called iCAT facilitates selective, privacy-preserving data sharing between tenants and providers. It utilizes an anonymization space to model various anonymization techniques, translating requirements from both parties into suitable primitives using NLP and ontology modeling. Secondly, a tenant-based, two-stage solution enhances auditing autonomy. The first stage utilizes tenant-side information to detect integrity breaches, while the second stage anonymizes provider-level data for tenant verification, offering control, transparency, and accuracy in breach identification. Additionally, a cryptographic approach is combined with side-channel watermarking to bolster tenant security. This lightweight solution enables continuous detection and classification of cloud-level attacks on service function chains, encoding cryptographic trailers as side-channel watermarks. This approach ensures verifiable attack detection without significant overhead, overcoming challenges such as limited side channel capacity and packet delay. By addressing these issues, the proposed solutions aim to enhance the security of NFV deployments and enable safer utilization of this innovative networking architecture.

Acknowledgments

The completion of this thesis is a collaborative effort, made possible with the invaluable help and support of numerous individuals, to whom I extend my sincere gratitude and appreciation.

I would like to express my deepest gratitude to Dr. Lingyu Wang for his unwavering guidance and steadfast support throughout my Ph.D. journey. His mentorship has been invaluable, providing me with not only profound academic insights but also the encouragement and motivation needed to navigate the challenges of doctoral research. Professor Wang's dedication to fostering intellectual growth and his commitment to excellence have played a pivotal role in shaping my academic pursuits. I am especially grateful for Professor Wang's understanding and support during a challenging period of health issues I faced. His compassion and flexibility helped me navigate these difficulties, highlighting his genuine concern for the well-being of his students.

I also extend my heartfelt appreciation to my co-supervisor Dr. Mourad Debbabi, for his support and guidance throughout my Ph.D. journey. Dr. Debbabi's expertise, insightful feedback, and dedication to my research have been invaluable assets, significantly contributing to the success of my doctoral studies. I am deeply grateful for Dr. Debbabi's commitment to excellence and his pivotal role in shaping the trajectory of my research. I am grateful to the members of my Ph.D. examination committee, Dr. Mustafa K. Mehmet Ali, Dr. Xinwen Fu, Dr. Abdelhak Bentaleb, Dr. Chadi Assi and Dr. Suryadipta Majumdar for the helpful comments they have provided along my Ph.D. journey.

I also want to extend my heartfelt gratitude to my colleagues at Audit Ready Cloud research group for an incredible seven-year journey filled with collaboration and exploration of the newest and hottest research topics. Working alongside such talented and dedicated individuals has been an enriching experience that has significantly shaped my professional growth. Specifically, I am immensely thankful to Dr. Makan Pourzandi, Dr. Yosr Jarraya, Dr. Suryadipta Majumdar, Amir Alimohammadifar, Dr. Meisam Mohammady, Dr. Mohammad Ekramul Kabir, Dr. Nour Boubakr, Dr. A.S.M Asadujjaman and Hinddeep Purohit for the camaraderie, support, and shared dedication to pushing the boundaries of knowledge. These years have been a remarkable chapter in my career, and I am grateful for the friendships and memories forged during our collective pursuit of excellence.

Finally, I would like to express my deepest gratitude to my family for their unwavering support throughout my Ph.D. journey. Your encouragement, understanding, and patience have been the pillars that sustained me during the challenges and triumphs of this rigorous academic pursuit. Your belief in my abilities, even during moments of self-doubt, has been a source of inspiration. Thank you for being my constant support system, for celebrating the milestones, and for standing by me during the arduous times. Your love and encouragement have made this journey not only academically rewarding but also emotionally fulfilling. To my friends, especially, Dr. Saed Alrabaee and Dr. Abdullah Al-Amaren I want to express my sincere gratitude for being steadfast companions on this academic odyssey. Your camaraderie, shared laughter, and unwavering encouragement have added immeasurable richness to the fabric of my Ph.D. experience. Your support, whether through late-night study sessions or uplifting conversations, has been invaluable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

NFV aims to outsource network functions (NFs) from dedicated proprietary appliances such that those functions can be implemented as software modules running on top of generic hardware [4]. The NFV market is projected to grow by 34.9% each year reaching 122$ billion by 2027 according to a forecast of ResearchAndMarkets [5]. For cloud service providers, this new paradigm allows them to more easily deploy a managed multi-tenant NFV environment on top of existing cloud infrastructure. For NFV tenants, the paradigm allows them to accelerate the provisioning and deployment of their network services at lower costs [5]. However, despite the great benefits of this technology, tenants usually show reluctance to virtualize their network services using NFV technologies. They anticipate dishonest behaviors by cloud providers, including violating data privacy and security requirements stated in the Service Level Agreements (SLAs) and deliberately concealing the violations by lying on the network services states [5]. Also, co-resident tenants, external attackers, or malicious insiders may compromise enterprises' network services to steal sensitive data or interrupt business operations [6]. Therefore, to bring the NFV technology into fruition, appropriate security auditing measures must be in place to detect any security violation or integrity breach that may arise [7].

Existing works on NFV security can be largely divided into two main categories. First, the stateful verification approaches [8, 9, 10] ensure that flows are forwarded correctly according to the high-level service chaining policies to capture the problems before deployment. These approaches rely on access to the underlying cloud infrastructure and to the descriptions of the NFs. However, they do not consider the privacy of co-resident tenants who might share the same resources and hence be indirectly involved in the auditing process. Also, such methodology fails to match the highly dynamic nature of NFV where anomalies can exist and disappear before being detected. Second, the run-time verification approaches [1, 11, 12] perform verification on the correct behavior of NFs and the SFCs after deployment. Moreover, most existing tools need access to the underlying cloud infrastructure to perform deployment and implementation changes. However, in real life, the cloud provider typically does not give the tenants access to the underlying deployment as it will increase the attack surface. Moreover, having a per-tenant solution on shared resources would not be something acceptable to co-resident tenants.

1.2 Problem Statement

In this thesis, we propose security auditing solutions to ensure the security of virtualized networks while respecting specific properties related to this technology (i.e., constraints of access, performance, scalability, and deployability). Therefore, we look beyond what traditional direct observation-based approaches can achieve. Instead, we analyze the indirect impacts of the attacks on the deployed infrastructure. More specifically, to overcome the limited access constraints, we propose an interactive and customizable approach for the cloud provider to enable selective data sharing in a privacy-preserving manner. Second, we propose a BlackBox and lightweight mechanism for the tenant to verify the integrity of SFCs in a privacy-preserving manner. Finally, we propose a tenant-based lightweight solution to perform continuous detection and classification of cloud-level attacks on SFCs. In particular, this thesis work mainly addresses the following research questions:

1. How to enable the cloud provider to share selective data with the tenants in a privacy-preserving manner?

2. How can we verify that virtualized networks are forwarding traffic and providing the same functionality as desired by the tenant?

3. How can we perform the verification while respecting the virtualized networks' properties to reflect real-life scenarios?

1.3 Research Contributions

In the following, we provide a brief overview of the main research contributions of this thesis. The contributions encompass two entities: 1) For the cloud provider, we propose the interactive and customizable data anonymization solution; 2) For the cloud tenant, we propose the following solutions: (i) side channel-based auditing of the integrity of Virtual Network Functions (ii) continuous verification of Virtual Network Functions based on virtual trailers.

1.3.1 Interactive and Customizable Data Anonymization

Data anonymization is a viable solution for cloud owners to mitigate their privacy concerns. However, existing data anonymization tools are inflexible to support various privacy and utility requirements of both cloud owners and data users. leaving two main gaps (i) between the cloud owner and data users' requirements, and (ii) between those requirements and the existing tool's anonymization capabilities. In most cases, this limitation is due to a lack of understanding of those requirements as well as the non-customizability of the existing tools. To address this limitation, we propose a solution for the cloud provider, namely, iCAT+, which is an interactive and customizable anonymization approach. More specifically, we first automate the interpretation of cloud owners' and data users' textual requirements by deploying a Convolutional Neural Network (CNN) model for Natural Language Processing (NLP). Second, we introduce the concept of the anonymization space to model possible combinations of per-attribute anonymization primitives based on the level of privacy and utility that each primitive provides. Third, we design an ontology model that maps the translated requirements into their appropriate anonymization primitives in the defined anonymization space corresponding to the plain data. Chapter 3 details our work on interactive and customizable data anonymization.

1.3.2 Auditing the Integrity of Virtual Network Functions Chain

There is a growing trend of hosting chains of Virtual Network Functions (VNFs) on third-party clouds for more cost-effective deployment. However, the multi-actor nature of such a deployment may allow a mismatch to silently arise between tenant-level specifications of VNF chains and their cloud provider-level deployment. Most existing auditing approaches would face difficulties in identifying such an integrity breach. First, relying on the cloud provider may not be sufficient, since modifications made by a stealthy attacker may seem legitimate to the provider. Second, the tenant cannot directly perform the auditing due to limited access to the provider-level data. In addition, shipping such data to the tenant would incur prohibitive overhead and confidentiality concerns. In this work, we design a tenant-based, two-stage solution where the first stage leverages tenant-level side-channel information to identify suspected integrity breaches, and then the second stage automatically identifies and anonymizes selected provider-level data for the tenant to verify the suspected breaches from the first stage. The key advantages of our solution are: (i) the first stage gives tenants more control and transparency (with the capability of identifying integrity breaches without the provider's assistance), and (ii) the second stage provides tenants higher accuracy (with the capability of rigorous verification based on provider-level data). Chapter 4 further describes our idea of auditing functional integrity for virtualized networks.

1.3.3 Continuous Verification of Virtual Network Functions Services

Network functions virtualization enables tenants to outsource their service function chains (SFCs) to third-party clouds for better agility and cost-effectiveness. However, outsourcing may restrict tenants' ability to directly inspect the cloud-level deployment to detect attacks on SFC forwarding paths, such as network function bypass or traffic injection. Existing solutions requiring direct access to the cloud do not apply to outsourcing, and cryptographic trailer-based ones may be expensive for large flows. In this work, we propose a lightweight solution for tenants to perform continuous detection and classification of cloud-level attacks on SFCs. Our main idea is to "virtualize" cryptographic trailers by encoding them as side-channel watermarks. This provides the best of both worlds, i.e., verifiable attack detection and classification without the overhead. We tackle several key challenges such as encoding virtual trailers within limited side channel capacity and minimizing packet delay. Chapter 5 further describes our idea of auditing functional integrity for virtualized networks. In summary, the main contributions of this thesis are as follows:

- To the best of our knowledge, we are the first to propose an automated approach to translate the requirements (expressed in English) of both data owners and data users by implementing and mapping them onto the anonymization space through NLP and ontology modeling. Such automated translation and mapping of user requirements improve the usability for data users and data owners as well as reduce potential human errors.

- To the best of our knowledge, we propose the first tenant-based approach to auditing the integrity of VNF chains. The two-stage design of our solution leads to two key advantages: i) the first stage gives the tenant more control and transparency to audit the underlying deployment; ii) the second stage provides higher accuracy to the tenant who can perform rigorous verification based on selected provider-level data (which has been automatically identified and anonymized).

- We propose the novel concept of virtual trailer, which inherits the advantages of both cryptographic trailer-based solutions (i.e., verifiable attack detection) and side channel watermarking (i.e., lightweight). To realize this, we address several key challenges such as encoding virtual trailers within the limited capacity of a side channel, minimizing packet delay while computing virtual trailers and detecting/classifying attacks with less trailers. We believe this concept may potentially find other applications in a broader context.

- All the proposed solutions are implemented and integrated into major cloud platforms such as OpenStack [3] and Amazon Elastic Compute Service (EC2) [13]. Extensive experiments with both synthetic and real data are performed to demonstrate the effectiveness and efficiency of the proposed approaches.

1.4 Relationships between the Research Topics

In the following, we describe the relationships between the three topics and how they were identified. First, we start with the goal of providing a solution for the cloud provider to enable selective data sharing with the tenants in a privacy-preserving manner. This will enable the tenants to challenge the cloud provider with audit requests, while allowing the cloud provider to fulfill such requests without worrying about privacy breaches. Second, while this approach achieves the main goal of providing more transparency to the cloud, it has a major limitation, i.e., the tenants must fully rely on the cloud provider for every audit request, and the provider would be overwhelmed by many audit requests coming from all the tenants. This limitation leads to the second research topic of developing a tenant-based auditing solution based on side channels, which returns some control back to the tenants, as they can perform auditing by themselves based on side channel information that is directly observed at the tenant-level. The solution also reduces the burden on the cloud provider, which will only be contacted for providing related evidence, after the tenants have already identified potential breaches. Third, while the second work achieves a balance between tenants' need for auditing and providers' need for reducing interaction, its reliance on side channel information means that its results are not always accurate, which can be a limitation to those tenants who desire provable security. Therefore, the last topic of this thesis aims to combine side channels with cryptographic approaches, such that we could have the best of both worlds, i.e., the lightweight nature of side channels and the rigor of crypto solutions. In summary, the three topics of this thesis are closely related, and they form complementary components of a common solution to achieve the ultimate goal of secure and privacy-preserving auditing for NFV.

1.5 Thesis Structure

This thesis is organized into six chapters. Chapter 1 introduces this thesis work. Chapter 2 reviews the related literature. Chapter 3 discusses the result of our interactive and customizable data anonymization tool. Chapter 4 presents our research work on auditing the integrity verification of virtualized network functions. Chapter 5 details our work on continuous integrity verification of virtualized networks using side-channel. Finally, we conclude our thesis in Chapter 6. Finally, table 1.1 summarizes the terminologies used in this thesis sorted alphabetically.

Table 1.1: List of acronyms used in thesis and their terminology

Acronym	**Terminology**
CNN	Convolutional Neural Network
DPI	Deep Packets Inspector
ETSI	European Telecommunications Standards Institute
FW	Firewall
GAN	Generative Adversarial Network
GBRAM	Goal-based requirements analysis method
GDPR	General Data Protection Regulation
GNN	Graph Neural Network
IDS	Intrusion Detection System
IPD	Inter Packets Delay
MAC	Message Authentication Code
ML	Machine Learning
NFP	Network Forwarding Path
NFV	Network Function Virtualization
NLP	Natural Language Processing
NPT	Network Performance Tomography
NS	Network Service
OS	Operating system
PU	Privacy UP
RTT	Round Trip Time
SFC	Service Function Chain
UD	Utility Down
VM	Virtual Machine
VNF	Virtual Network Function
VNF	Virtual Network Function Forwarding Graph
VT	Virtual Trailer

Chapter 2

Related Work

In this chapter, we review related works of prior research on our identified problem areas. Additionally, we provide a comparison between our proposed solutions and the existing works. The review of related works is structured as follows. First, in Section 2.1, we review the literature and show a qualitative comparison between existing works and our proposed solutions in the area of data anonymization. Second, in Section 2.2, we review the literature and show a qualitative comparison between existing works and our proposed solutions in the area of forwarding integrity verification. Finally, in Section 2.3, we review the literature and show a qualitative comparison between existing works and our proposed solutions in the area of continuous integrity verification.

2.1 Data Anonymization Tools

In this section, we discuss the existing works in the domain of machine learning (ML), anonymization, and the domain of privacy goals mining from privacy policies. We also demonstrate the existing data anonymization tools, and their limitations and provide a taxonomy based on the nature of each tool into- (i) cryptography-based anonymization tools, and (ii) replacement-based anonymization tools. Finally, we study the tools' capabilities under each taxonomy in terms of the features they provide, the fields that they cover, and the anonymization primitives they support.

2.1.1 ML-based anonymization

In [14], the authors propose a machine learning-based model that could remove sensitive personal health information. Unlike *iCAT*, the authors deploy an ML algorithm to act as an anonymization primitive to anonymize data. However, in *iCAT*, the ML algorithm is used to help in understanding the users' needs and translate them into the appropriate anonymization primitives only. Moreover, we link the requirements translation phase in *iCAT* to the privacy goals extraction from the privacy policies mining field, as we are aiming for the same interest in inferring requirements from the natural English text. On the other hand, in [15, 16], the authors introduce the goal-based requirements analysis method (GBRAM) and heuristics to extract goal specifications from the text. Then, they apply GBRAM to mine privacy goals from privacy policies. In [17], the authors report results from three experiments aimed at assessing the potential of crowd-sourcing requirements extraction to non-experts. The authors show the cost, efficiency, and effectiveness of that task and conclude that by using NLP techniques, the cost decreases, and the requirements coverage increases compared to manual extraction by trained experts. A combination of crowd-sourcing and NLP is implemented in [18], while the authors introduce and evaluate a method that combines crowd-sourcing and NLP to extract goals from privacy policies. Their analysis depicts that crowd workers can provide human interpretations that are still beyond the state of the art in NLP and the NLP can provide a cost-effective and more effective goal extraction. As per our best knowledge, unlike all existing works, *iCAT* deploys the extracted privacy goals from the privacy policy in data anonymization.

Table 2.1: Comparing existing network data anonymization tools with iCAT.

Tool	**Anonymized Fields**					**Anonymization Primitive**					
Name	**NF fields**	**IP**	**Port**	**Header**	**Payload**	**Pref-Pres**	**Hiding**	**Permutation**	**Truncation**	**Hashing**	**Shifting**
AnonToo [19]	✓	✓				✓	✓	✓		✓	
CANINE [20]	✓	✓	✓			✓	✓	✓	✓		✓
CoralReef [21]	✓	✓	✓			✓	✓		✓		
Flaim [22]	✓	✓	✓			✓	✓	✓		✓	✓
IPsumdump [23]		✓		✓		✓					
NFDump [24]		✓				✓					
SCRUB [25]		✓		✓	✓		✓	✓			✓
TCPanon [26]					✓		✓				
tcpdpriv [27]		✓		✓	✓	✓	✓	✓	✓		
TCPmkpub [28]		✓		✓	✓	✓	✓		✓		
TCPurify [29]		✓			✓		✓	✓	✓		
iCAT	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓

2.1.2 Cryptography-based Anonymization Tools

To distinguish *iCAT*, we present our taxonomy of existing data anonymization tools. Table 2.1 compares those tools according to the anonymized fields (e.g., IP, header, port, etc.) and the anonymization primitives they use. As shown in Table 2.1, except *iCAT*, none of those tools can support all the attributes or anonymization primitives (let alone the flexibility for customization), nor can take users' requirements to understand their privacy and utility needs. The cryptography-based anonymization tools are considered as the first taxonomy, whereas most of the existing tools under this taxonomy use cryptography-based anonymization primitives; such as prefix-preserving, shifting, hashing, and permutation. Existing tools in this category are used to anonymize network traces and mainly anonymize the TCP header. However, some of those tools support live interfaces anonymization to anonymize the data in a running-time manner. Moreover, tools under this taxonomy provide higher privacy output and are well known to be more user-friendly as the tool user does not require to have good knowledge about the anonymization primitives.

2.1.3 Replacement-based Anonymization Tools

The second taxonomy is the replacement-based anonymization tools, while the existing tools in this category deal mainly with log files and anonymized data by replacing the sensitive attributes (e.g., passwords, system logs, files paths, etc.) in the log with some values predefined by the user in the so-called rule-file or generated using deterministic cryptography algorithms. The rule file contains patterns used by the tool to perform pattern matching and the conversion state of the anonymization can be stored in a look-up table. This category of anonymization provides a higher utility output because it preserves some property of the original data (e.g., equality, format, order, etc.). However, this is also susceptible to de-anonymization attacks, known as semantic attacks (e.g., frequency analysis, injection, and shared text matching attacks). Moreover, those tools are generally not user-friendly and require knowledge about conducting tool-based search patterns and managing the conversion state of the anonymized data.

2.2 Forwarding Integrity Verification

2.2.1 NFV-based Security Solutions

Most of the existing solutions (e.g., [1, 11, 30, 31, 9, 32]), to audit integrity breaches in NFV rely on the provider-level data. FlowCloak [31] and vSFC [11] identifies VNF chain violations (e.g., path non-compliance and packet injection attacks). Similarly, SFC-checker [9] and ChainGuard [30], audit the forwarding behavior of the VNF chains in a network service. AuditBox [1] provides continuous assurance that packets follow the formally specified policy-mandated path using formal models. On the other hand, several works (e.g., [33, 34, 35]) focus on the performance and functionality of NFV network services. Unlike those works, our solution provides a tenant-based, two-stage approach which does not fully rely on the provider.

2.2.2 Performance-based Identification Solutions

There exist several performance-based identification solutions (e.g., [36, 37, 38]) for virtualized environments. For instance, Koh et al. [36] study the inter-VM interference performance characteristics by collecting runtime performance measures. Whereas, Mei et al. [37] study the network performance in a virtualized cloud environment while varying the network I/O workloads. The authors in [38] analyze the performance of virtual machines in an IaaS cloud environment to infer the network topology. There are a few other works (e.g., [39, 40, 41]) that analyze VNFs to find performance issues in NFV. Unlike those solutions, our solution, in Stage 1, uses the performance characteristics to identify suspected integrity breaches.

2.2.3 Network Tomography Solutions

There are several works (e.g., [42, 43, 44, 45]) that use network tomography for characterizing network behavior. Among them, traceroute [42] and iperf [43] use the node-to-node tomography approach to measure performance metrics (e.g., delay, loss rate) of a specific link directly through sending probing traffic from source to destination. On the other hand, using the end-to-end tomography approach, Chen et al. [44] calculate unknown link variables and Arifler et al. [45] identify the congested links. Similar to existing works, our solution uses network tomography to collect performance characteristics data, but uniquely for identifying integrity breaches in NFV.

2.2.4 Comparison Among Related Works

Table 2.2 summarizes the comparison between the most recent NFV security auditing works and NFVSense, Chapter 4. Our comparison is based on different properties that those solutions support. Specifically, the detection coverage (i.e., run-time or stateful), the auditing capabilities (i.e., dynamic or static), the deployment level (i.e., tenant-side or provider-side), the integrity breaches, and the provider level accessibility (i.e., blackbox or whitebox).

Table 2.2: Comparing our work with existing solutions.

Work	Run-time	Dynamic	Tenant-based	Path Modif.	Blackbox
vSFC [11]	✓	✓	x	✓	x
FlowCloak [31]	✓	✓	x	✓	x
AuditBox [1]	✓	✓	x	✓	x
SFC-Checker [9]	x	x	x	✓	x
EasyOrch. [46]	x	x	N/A	x	x
FlowTags [32]	✓	x	✓	✓	x
ChainGuard [30]	✓	x	x	✓	x
NFVSense	✓	✓	✓	✓	✓

2.3 Continuous Integrity Verification

2.3.1 SFC integrity

Traditional forwarding path verification protocols [47, 48, 49] cannot be directly applied to SFCs hosted in an NFV environment, since some of their underlying assumptions become unrealistic in the NFV context, e.g., forwarding paths are no longer fixed or known in advance in NFV, and network nodes (VNFs) are no longer transparent to packets as they might legitimately modify packets. More recent works tackle such issues to enable forwarding path verification in NFV. In [32], the authors study the issue of stateful and dynamic actions performed by VNFs, and propose a solution for VNFs to add tags to outgoing packets to bind packets with their origin. However, this scheme becomes ineffective when one or more switches are compromised. Therefore, FlowCloak [31] proposes an advanced packet tagging approach to randomize the tag generation such that the tags are probabilistically unknown by compromised switches. In contrast to our solution, FlowCloak requires modification to the internal logic of VNFs which may complicate its deployment. In [50], the authors propose a verification layer that is decoupled from the processing of VNFs and is embedded in VMs supporting those VNFs. Nonetheless, vSFC requires modification at the cloud level, whereas our solution is a tenant-level solution that regards the cloud as a Blackbox. SFC-Checker [9] proposes a static analysis-based framework to ensure the correct behavior of dynamic and stateful forwarding paths. EasyOrch [46] performs verification based on a formal model that provides the flexibility of specifying both a forwarding policy and the set of anomalies to verify. In contrast to our solution, those solutions are static and cannot detect run-time integrity breaches, as they either take snapshots of the network state to perform verification offline [9] or work before SFC deployment [46]. In [51], the authors propose a hardware-based solution to enclose both VNF processing and verification inside enclaves to preserve data confidentiality and VNF integrity against powerful adversaries, which are different from the focus of our solution (i.e., network links between VNFs). Closest to our work, AuditBox [1] provides runtime guarantees on the compliance with forwarding path policies using a hop-by-hop cryptographic trailer-based protocol, and by running each VNF inside enclaves. While our solution borrows a similar design of trailers as AuditBox, our virtual trailer concept can significantly reduce its communication overhead (as demonstrated through experiments in Section 5.6).

2.3.2 Digital watermarking

There is a rich literature on digital watermarking in different contexts (e.g., image, audio, video, and network packets) and for different applications (e.g., copyright protection, traffic analysis, and tampering identification). The literature may be categorized along different dimensions. For instance, *blind* (e.g., [52]) or *non-blind* (e.g., [53, 54]) watermarking indicate whether the embedding and extraction of the watermarks require sharing knowledge about the original data. our solution leverages a blind watermarking scheme to avoid the overhead of sharing additional information about the original data. Second, for applications such as copyright protection and traffic analysis, the watermarks will be subject to either natural network noises on the Internet, or malicious tempering. Therefore, the watermarks should be *robust* [55] and/or *invisible* [54, 53] (i.e., hide watermarks against adversaries who aim to locate the watermarks [56, 57]). On the other hand, a *fragile* watermarking scheme (e.g., [58]) is mainly used for tampering identification as well as localisation of tampered data, and hence the watermarks should be sensitive to modifications and are not necessarily invisible. our solution belongs to the fragile (and non-invisible) category, since its watermarks are used to detect and classify tampering. The key innovation of our solution is it introduces an additional abstraction layer, i.e., virtual trailers, over watermarks, in order to leverage the latter's cryptographic properties.

Timing-based side channels are shown to have a larger capacity to share more information compared to other side channels [59]. In particular, packet timing-based flow watermarking modulates the IPDs of target network flow to embed watermarks and achieve the goal of linking flows for different applications, such as detection of stepping stone attacks, and compromising anonymity systems. For instance, the authors in [52] propose an IPD-based probabilistically robust watermarking scheme, which embeds watermark bits through slightly adjusting the independently and randomly selected IPDs. In [60], the authors propose an enhanced scheme where the watermarker can adaptively choose values of watermark parameters according to packet timing and packet size features of target flows. In [61], the authors propose a scheme that resists timing perturbations through grouping-based flow watermarking. In [62], the authors propose a flow watermarking technology based on packet matching and IPDs. In [63], the authors propose a blind flow watermarking system, which modulates fingerprints into the timing patterns of network flows through slightly delaying packets into secret time intervals only known to the fingerprinting parties. More recent works leverage machine learning to design more robust watermarking schemes, or employ watermarking to authenticate machine learning models. For instance, Fang et al. [64] apply deep learning techniques to obtain more robust flow-based watermarking schemes to ensure high consistency between the encoder and the decoder, and Xu et al. [65] design a watermarking scheme for graph data to verify the ownership of Graph Neural Networks (GNN) models. Although those works are similar to our solution in that they also deal with network data, they mostly fall in the robust (and often invisible) watermarking category, since they mostly aim to correlate flows with unique watermarks, which need to survive malicious modifications and the noises natural to Internet traffic. This is different from our solution which employs a fragile and visible watermarking scheme to detect integrity breaches.

2.3.3 Comparison Among Related Works

Finally, Table 2.3 summarizes the main advantages of ChainPatrol, Chapter 5, over existing works on SFC integrity along several dimensions:

- **Timeliness** (runtime vs. offline): Offline solutions work on static snapshots taken periodically from the deployment. Runtime solutions like our solution continuously detect and classify SFC-based attacks based on real-time traffic.

- **Methodology**: Those existing solutions can be largely classified as either cryptography-based or formal method-based. our solution combines the security guarantee provided by cryptographic trailers with side channel-based watermarking for a lightweight solution.

- **Blackbox**: Most of the existing solutions are white-box approaches since they require access or modification to the underlying cloud. our solution is a tenant-based Blackbox approach that does not require such accesses since it works on inter-packet delays that are directly observable/mutable at the tenant level.

- **No instrumentation**: Most of the existing solutions require some modification or amendment to the cloud-level deployment of VNFs. our solution does not require any instrumentation of VNFs, since it is a tenant-level solution and its agents work as independent proxies attached to VNFs.

- **No modification to cloud infrastructure**: Most of the existing solutions require some modification to the cloud infrastructure. our solution requires no such modification and can be independently deployed by the tenant.
- **Out-of-scope attacks**: Compared to many of the existing works, our solution covers more attack types as it inherits the strength of a cryptographic trailer-based approach [1].

Table 2.3: Comparing our solution with existing solutions

Work	**Timeliness**	**Methodology**	**BlackBox**	**No Instr.**	**Unmodified Infra.**	**Out-of-scope**
vSFC [11]	Runtime	Crypto-based	No	No	No	-
FlowCloak [31]	Runtime	Crypto-based	No	No	Yes	Dropping, replay
AuditBox [1]	Runtime	Crypto-based	No	No	No	-
FlowTags [32]	Runtime	Crypto-based	No	No	Yes	Forwarding misbehaviour
ChainGuard [30]	Runtime	Crypto-based	No	No	No	Dropping, modification
SFC-Checker [9]	Offline	Formal Methods	No	Yes	Yes	Dropping, modification, replay
EasyOrch. [46]	Offline	Formal Methods	No	Yes	Yes	Dropping, modification, replay
Fulvio et al. [46]	Offline	Formal Methods	No	Yes	Yes	Dropping, modification, replay
ChainPatrol	**Runtime**	**Crypto + side channel**	**Yes**	**Yes**	**Yes**	**-**

Chapter 3

Interactive and Customizable Data Anonymization

3.1 Introduction

Network data has recently become an increasingly valuable asset that enables various applications for different stakeholders in many sectors [66]. At the same time, the reluctance in sharing those data, especially from the fear of sensitive information leakage, is also well-known (e.g., [67, 68]). Moreover, this reluctance is exacerbated by potential financial implications of privacy regulations (e.g., the General Data Protection Regulation (GDPR) [69]), by an increasing trend of emerging attacks (e.g., frequency analysis and data injection attacks [70]) and high profile data breaches and misuse incidents[1] [2], and by the growing availability of large-scale data analytics that might further empower the attackers.

To this end, data anonymization is a widely adopted solution for mitigating data owners' concerns [71]. On the other hand, since data users are often not interested in the plain data itself, but in its semantics [72], anonymized data also could be useful for data users to

[1]https://www.identityforce.com/blog/2021-data-breaches.

[2]https://www.techworld.com/security/uks-most-infamous-data-breaches-3604586/

attain their goals. Nonetheless, a key challenge in applying anonymization solutions is that the effectiveness of such solutions critically and solely depends on how well the data owner makes the right choices of anonymization primitives. Such choices must achieve the significant trade-off between utility and privacy so that the data owners' sensitive information is properly hidden, whereas the desired information by the data receivers are well-preserved. On the other hand, such a choice is highly dependent on a proper understanding of all the requirements from both data owners and data users. To fulfill those requirements at a satisfactory level, a data owner needs to:

- understand his/her own privacy requirements as well as the utility requirements of potential data users;
- understand the capabilities (in terms of anonymization primitives) of the available anonymization tools;
- map all the privacy/utility requirements correctly and consistently onto the capabilities of the anonymization tools; and
- select the right combination of anonymization tools for different data attributes to achieve the desired trade-off between utility and privacy.

However, since most data owners may not be familiar with all the privacy concepts and solutions, they would find those tasks challenging, if not infeasible. Additionally, they may not have much incentive to understand the utility requirements of data users. Moreover, due to the lack of an efficient and automated translator, data users may also fail to translate their requirements in an accurate way to present their demands to the data owners. As a ramification, the data owners may simply decide to withhold the datasets, as indicated in several studies (e.g., [73]). Such tendency practiced by data owners of withholding the datasets is understandable since fulfilling those tasks would demand a systematic knowledge of the search space, i.e., all possible combinations of anonymization primitives, and their mapping to the privacy/utility requirements. Moreover, most existing anonymization tools (e.g., Loganon [74], Camouflage [75], etc.) only support a limited number of choices, providing one-size-fits-all solutions, and manually mapping the privacy and utility requirements onto the tools' anonymization capabilities. In summary, there are two major gaps:

- The first gap is between the capability of a typical data owner and the expectations of current solutions from data owners to identify the right combinations of anonymization primitives for given privacy/utility requirements;
- The second gap is between the wide range of privacy/utility requirements and the limited support of anonymization capabilities by existing tools.

To fill in both gaps, our key idea is to design an automated, interactive, and customizable data anonymization framework that can translate privacy/utility requirements to identify the right combination of anonymization primitives that satisfy those privacy/utility requirements. More specifically, we first adopt a Convolutional Neural Network (CNN) model to automatically translate both data owners' and data users' requirements expressed in natural language (i.e., English) into their corresponding anonymization primitives. Second, we identify the different data attributes and generate possible combinations of anonymization primitives, namely, *the anonymization space*. Third, we build an ontology that develops rules for mapping the translated requirements to the anonymization space that can provide privacy-utility guarantees. Finally, we apply those mapping rules to translate requirements to the anonymization space and provide a set of anonymization combinations that satisfy both parties requirements. In summary, our main contributions are:

1. We propose an automated approach to translate the requirements (expressed in English) of both data owners and data users by deploying a CNN model, and mapping them onto the anonymization space through the NLP and the ontology modeling.

Such automated translation and mapping of user requirements improve the usability for data users and data owners as well as reduce the potential human errors (i.e., the effectiveness of the translation of the requirements is around 98% for data users and 99% for data owner).

2. To the best of our knowledge, our notion of *anonymization space* is the first model that systematically characterizes and organizes existing anonymization primitives based on their relative capabilities in terms of privacy and utility. This model provides data owners a comprehensive and yet intuitive understanding of the available anonymization choices, and it also, for the first time, allows the data users to be actively involved in making their own decisions.

3. We design and implement an automated tool, *iCAT*, that integrates popular anonymization primitives into a single framework, and selects and configures the proper primitives that satisfy the requirements of data owners and data users by utilizing the proposed anonymization space. Compared to most existing anonymization tools, *iCAT*, interactively provides more flexibility (access to the entire anonymization space) and better usability (automated requirement translation), and can support the largest combination of attributes and anonymization primitives.

4. We experimentally evaluate the effectiveness of *iCAT* using both synthetic and real (e.g., Google cluster dataset [76]) data, while the usability of *iCAT* is appraised through a user study involving participants from both industry and research labs.

3.2 Motivation and Preliminaries

In this section, we further illustrate our motivation using an example. Additionally, we define our threat model and discuss the considered anonymization primitives.

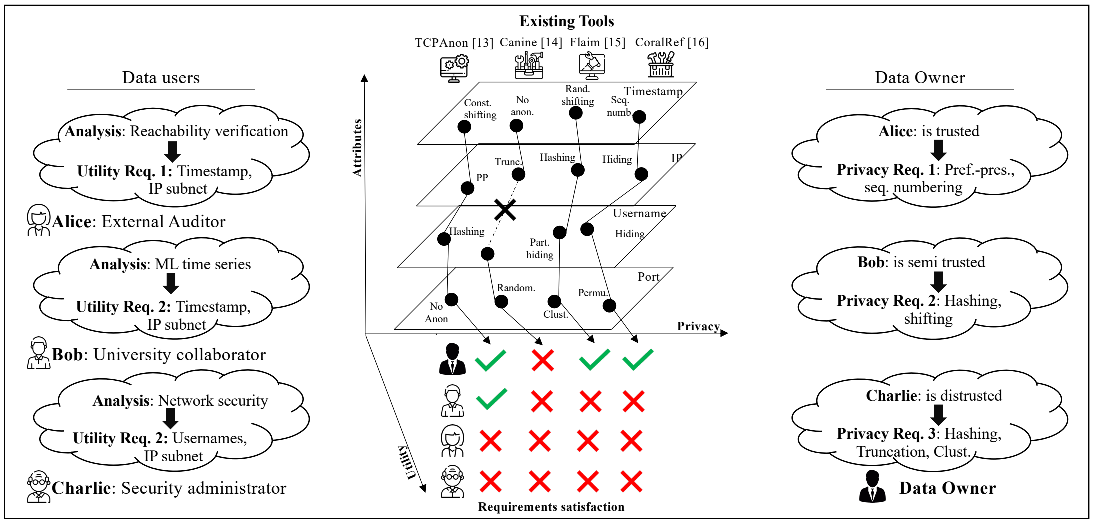

Existing Tools
TCPAnon [13]
Canine [14]
Flaim [15]
CoralRef [16]
Data users
Analysis: Reachability verification
Utility Req. 1: Timestamp, IP subnet
Alice: External Auditor
Analysis: ML time series
Utility Req. 2: Timestamp, IP subnet
Bob: University collaborator
Analysis: Network security
Utility Req. 2: Usernames, IP subnet
Charlie: Security administrator
Attributes
Privacy
Utility
Requirements satisfaction
Data Owner
Alice: is trusted
Privacy Req. 1: Pref.-pres., seq. numbering
Bob: is semi trusted
Privacy Req. 2: Hashing, shifting
Charlie: is distrusted
Privacy Req. 3: Hashing, Truncation, Clust.
Data Owner

Figure 3.1: The motivating example.

3.2.1 Motivating Example

Figure 3.1 depicts a scenario where data owners (on the right) would like to anonymize their data using anonymization tools (in the middle) before handing over the data to the data users (on the left). Specifically, first, we consider three data users (Alice: an external auditor, Bob: a university collaborator, and Charlie: a security administrator) who intend to conduct different analysis tasks, i.e., reachability verification, ML time series analysis, and network security, respectively. Second, we consider a data owner who has different trust levels for those users: trusted, semi-trusted, and distrusted. Third, we consider four different existing anonymization tools (i.e., TCPanon [26], Canine [20], Flaim [22] and CoralRef [21]) which might be employed for this situation, but only if they guarantee proper anonymization to meet the desired trust levels. (i.e., *privacy requirements*) of the data owner as well as preserve data quality for the planned analysis tasks (i.e., *utility requirements*) of data users. However, matching requirements and identifying the most appropriate anonymization tools might become a non-trivial task for data owners and users for the following reasons.

- Even though each data user (e.g., Alice) might have an understanding of his/her analysis tasks (e.g., reachability verification), identifying their correct and concrete utility requirements (e.g., preserving both sequences of timestamps and subnets in IPs will be needed for reachability verification) might not always be feasible (as confirmed later by our user-based experiments in Section 3.4). This is mainly because many iterations of interactions between a data owner and a data user have to be performed to identify the correct utility requirements.

- Even though a data owner might be capable of understanding his/her trust level of each data user (e.g., Alice is trusted), relying on data owners for identifying concrete privacy requirements (e.g., Alice can only be given prefix-preserving and sequentially numbered data) might not be practical mainly due to the fact that real-world data owners are usually not so considerate and might simply go with whatever is suggested by a handy anonymization tool [17].
- As shown in the middle of Figure 3.1, most existing tools (e.g., [26], [20], [22] and [21]) only implement a small set of anonymization primitives (e.g., constant shifting, hashing) suitable for a subset of the data attributes (e.g., timestamp, IP). Furthermore, those tools are not customizable enough to accommodate specific combinations of privacy and utility requirements. As a result, most tools fall short to satisfy the requirements of data owners and data users.

To address the aforementioned challenges, we propose *iCAT*. Intuitively, *iCAT*:

- automatically translates data users' utility requirements in terms of data attributes;
- automatically translates data owners' privacy requirements in terms of anonymization primitives; and
- defines the entire "anonymization space" (that maps data attributes and their related anonymization primitives) instead of covering a subset of it. We elaborate on *iCAT* in Section 3.3.

3.2.2 Threat Model

We define the parties who are involved in the data anonymization process and their trust relationships in a more realistic approach as follows:

- **data owner:** who has useful datasets that can be used for different purposes and is interested in protecting the privacy of his/her data to avoid any data misuse. The data owner has different trust levels to the data users, which determines the exposure of data that s/he allows.

- **Data users:** who have different intentions of using the data (e.g., auditing, research purposes, etc.), are interested in having the maximum data utility to achieve valid results. The data users trust the data owners and are willing to share their use cases with them.

In the following, we elaborate on both in-scope and out-of-scope threats.

In-scope threats. We assume that both data owners and users are willing to follow the procedure to express their requirements, while the data user is interested in obtaining output with a higher utility if the tool provides him/her with such an opportunity. Moreover, we consider the case where the data user might tamper with the learning and requirement translation process to obtain a higher utility output.

Out of scope threats. The intention of *iCAT* is not to mitigate any weakness or vulnerability of the underlying anonymization primitives (e.g., frequency analysis, data injection attacks, or data linkage attacks). Consequently, those primitives are used as a BlackBox in our data anonymization module and can be replaced by other, better primitives when available. Also, we do not consider the case where a data user uses the tool with the data owner's privileges, where s/he has more capabilities. Finally, any integrity breach of the translation in NLP techniques is beyond the scope of this work.

3.2.3 Anonymization Primitives

In spite of the existing many data anonymization primitives in the literature, most current tools only support a limited number of primitives. Table 3.1 provides a list of such common anonymization primitives, examples of plain data, and the corresponding anonymized data obtained using those primitives. However, this list is not meant to be exhaustive, and our model and methodology can be simply extended to include other anonymization primitives.

Table 3.1: Examples of plain data inputs and their corresponding anonymized outputs for widely-used anonymized primitives.

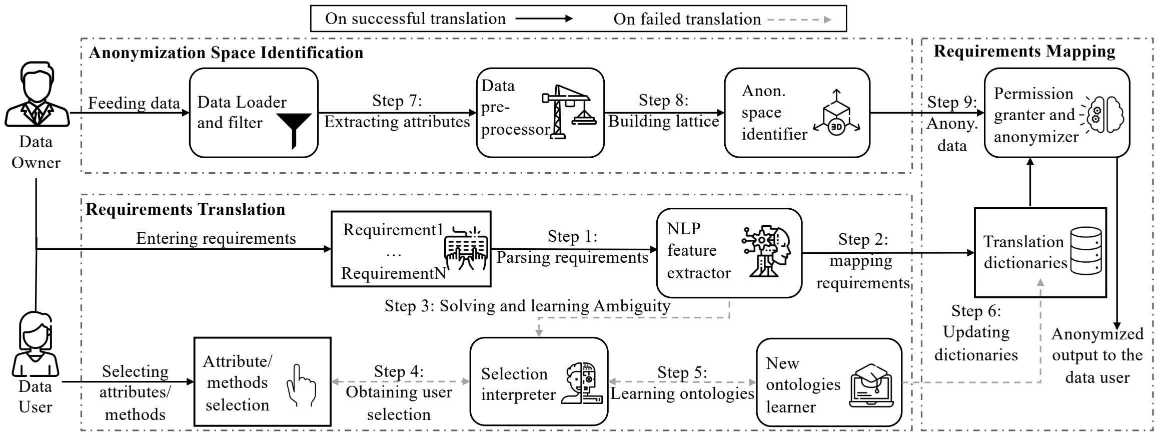
Primitive	**Plain Data Input**	**Anonymized Output**
Prefix-preserving	IP1:**12.8.3**.4 ; IP2:**12.8.3**.5	IP1:**51.22.7**.33 ; IP1:**51.22.7**.19
Truncation	IP1:12.8.**3.4** ; IP2:12.8.**3.5**	IP1:12.8.**X.X** ; IP2:12.8.**X.X**
Const. Substitution	Version:2.0.1	Version: VERSION
Const. Shifting	Time1: **2019**-03-31; Time2: **2019**-03-30	Time1: **2022**-03-31; Time2: **2022**-03-30
Random Shifting	Time1: **2019**-03-31; Time2: **2019**-03-30	Time1: **2003**-03-31; Time2: **2015**-03-30
Sequ. Numbering	Time1: 2019-03-31; Time2: 2019-03-30	Time1: T1; Time2: T2
Partial Hiding	Time1: 2019-**03-31**; Time2: 2019-**03-30**	Time1: 2019-**X-X**; Time2: 2019-**X-X**
Hashing	ID:40018833	ID: H3%s2*D9
Clustering	Port1:2**25**; Port2: 2**77**	Port1:**200**; Port2: **277**
Permutation	Port1:2**25**; Port2: 2**77**	Port1:2**77**; Port2: 2**25**
Randomization	Port1: **225**; Port2: **277**	Port1:**423**; Port2: **29**

3.3 Methodology

In this section, we provide overviews of the *iCAT* approach, as well as our anonymization space, and preference up and utility down (*PU/UD*) rules.

3.3.1 Approach Overview

Figure 3.2 depicts three major processes of *iCAT* including their corresponding steps: (i) requirement translation (Steps 1-3), (ii) anonymization space creation (Steps 4-6), and (iii) requirement mapping (Steps 7-9). In the following, we elaborate on each of them.



On successful translation
On failed translation
Anonymization Space Identification
Requirements Mapping
Data Owner
Feeding data
Data Loader and filter
Step 7: Extracting attributes
Data pre-processor
Step 8: Building lattice
Anon. space identifier
Step 9: Anony. data
Permission granter and anonymizer
Requirements Translation
Entering requirements
Requirement1 … RequirementN
Step 1: Parsing requirements
NLP feature extractor
Step 2: mapping requirements
Translation dictionaries
Step 3: Solving and learning Ambiguity
Step 6: Updating dictionaries
Anonymized output to the data user
Data User
Selecting attributes/ methods
Attribute/ methods selection
Step 4: Obtaining user selection
Selection interpreter
Step 5: Learning ontologies
New ontologies learner

Figure 3.2: An overview of *iCAT*.

- **Requirement translation**: In Step 1, *iCAT* accepts the requirements as plain text in English to ameliorate the burden of both the data owners and users. In Step 2, it parses those requirements into a combination of anonymization primitives and data attributes using NLP. In Step 3, *iCAT* deploys a feedback option to allow the users for a manual interpretation, in case the NLP fails to translate any requirement.

- **Anonymiztion space creation**: In Step 4, *iCAT*, performs filtering and pre-processing on the data (received from data owners) to remove undesired columns and rows. In Step 5, it extracts the total number of attributes (e.g., six columns) and their types from the processed input data (e.g., IP address, string, timestamp, etc.). In Step 6, *iCAT* creates the anonymization space based on the attributes number and data types generated from the previous steps.

- **Requirement mapping**: In Steps 7 and 8, based on the attribute type of each requirement, *iCAT* maps those requirements into anonymization primitives in the anonymization space corresponding to the input data. In Step 9, based on the intersection between the data owner and data user requirements, *iCAT* provides a set of anonymization combinations to the data user which also meets the data owner requirement.

These steps will be further detailed in Sections 3.3.3, 3.3.7, and 3.3.8, respectively.

3.3.2 An Overview of Anonymization Space

This section first describes the need for an anonymization space and then define our proposed anonymization space.

The Need for an Anonymization Space. There is a need to determine a systematic approach to represent and organize all the possible choices of anonymization primitives for applying on a given dataset to offer the freedom of choice to data owners for heightening the privacy level even after ensuring the acceptable utility level for users. More specifically, first, assigning a trust level for each data user and translating this trust level into a privacy requirement is not a straightforward process due to the limited capabilities available by existing tools. Second, the existing anonymization primitives (i.e., shown but not limited to in Table 3.1) provide a wide range of anonymization possibilities and lead to cover a large number of trust levels as listed below.

- Each data attribute may be anonymized using a different collection of the anonymization primitives (e.g., IPs may work with prefix preserving, truncation, hashing, etc., while IDs with clustering, hashing, etc., and both can be either completely hidden or given as simple text without any anonymization).

- Or, different anonymization primitives applied to an attribute may yield different levels of, and sometimes incomparable, privacy and utility (e.g., for IPs, hashing provides more privacy/less utility than prefix preserving, whereas they are both incomparable to truncation or randomization).

- Or, the data owner and data users' requirements typically involve multiple attributes, as demonstrated in Figure 3.1, and sometimes in a complex fashion, e.g., the data owner might say "I can only give you the data with the IPs hashed, or with the IDs clustered, but not both", while a data user asks "I know I may not get the data with the IPs truncated and the IDs hashed, but what would be my next best option?"

Definition of the Anonymization Space. To meet the above-mentioned needs, we propose a novel concept, namely, *anonymization space*, by considering each data attribute as a *dimension*, and each combination of anonymization primitives that can cover all the attributes as a *point* inside the *anonymization space*. Since anonymization primitives are not always comparable in terms of privacy/utility, inspired by Denning's Axioms [77], we consider the collection of anonymization primitives applicable to each attribute to form a lattice[78] on their relationships in terms of privacy and utility. The product of all those lattices is the *anonymization space*. The formal definition and an example are as follows.

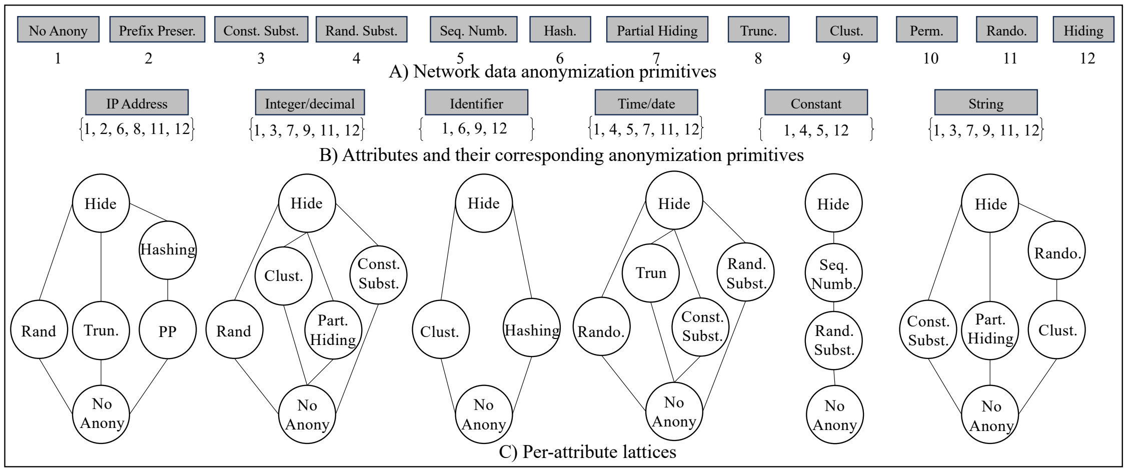


Figure 3.3: *An example of anonymization space:* **A**) examples of anonymization primitives with their indices, **B**) examples of data attributes and their applicable anonymization primitives, and **C**) the per-attribute anonymization lattices.

Definition 3.3.1 ***(Anonymization Space)*** *Given $\mathbb{A} = \langle a_1, a_2, \ldots, a_n \rangle$ as a set of attributes to be anonymized, and given $F_i = \{f_1, f_2, \ldots, f_m\}(1 \leq i \leq n)$ as the anonymization primitives set applicable to a_i, we define:*

- *The attribute anonymization lattice $\mathcal{L}_i(1 \leq i \leq n)$ as a lattice $\langle F_i, \prec \rangle$ where for any $f_1, f_2 \in F_i$, we have $f_1 \prec f_2$ iff f_1 provides better utility and more stringent privacy than f_2 when applied to a_i, and*
- *The anonymization space corresponding to $\mathbb{A}$ is denoted by $\prod_{i=1}^{n} \mathcal{L}_i$.*

Example 3.3.1 Figure 3.3.A (top) shows some examples of anonymization primitives, Figure 3.3.B (middle) shows their applicability (using their indices) to six attributes and Figure 3.3.C (bottom) shows the six attribute anonymization lattices. Due to space limitations, we omit the anonymization space representation (which would have a size of $20,736$ different anonymization combinations).

3.3.3 An Overview of *PU/UD* Rules

The preference up and utility down (PU/UD) rules are to address one of the major challenges in mapping privacy or utility requirements into anonymization primitives to ensure acceptable utility levels for all data users without infringing the data owner's privacy. Inspired by the Bell–LaPadula (BLP) model [79], in PU/UD rules, we adopt a concept of jointly enforcing the privacy and utility requirements through a simple access control mechanism. In this mechanism, data users actively participate in the anonymization process to maximize their utility levels, while this mechanism itself ensures the data owner's privacy concerns.

Due to considering each point (i.e., the collection of anonymization primitives) in the anonymization space as a privacy/utility *level*, the data owner's privacy requirement can be mapped to a level in such a way that anything above this level can satisfy the privacy requirement. This mapping is automated, and its implementation details will be discussed in Section 3.3.6. Since this approach yields the privacy higher, we define this as the *privacy-up* rule. Simultaneously, a data user's requirement can be mapped to a level in this anonymization space below which any level would satisfy the utility requirement, namely, the *utility-down* rule. Hence, these *privacy-up* and *utility-down* rules can be combinedly called *PU/UD* rules. The formal definition and an example are as follows.

Definition 3.3.2 ***(PU/UD rules)*** *Given the set of attributes $\mathbb{A}$, the corresponding anonymization space $\mathbb{AS} = \prod_{i=1}^{n} \mathcal{L}_i$, then:*

- *The Privacy Up (PU) lattice denoted by $L_p \in \mathbb{AS}$, represents the nodes that have the least utility level compared to what is specified by the data owner in the privacy requirement.*
- *The Utility Down (UD) denoted by $L_u \in \mathbb{AS}$, represents the nodes that have the least privacy level compared to what is specified by the data user in the utility requirement.*

  Respectively, any $L \in \mathbb{AS}$ will satisfy both requirements if $L_p \prec L$ (PU) and $L \prec L_u$ (UD) are both true.

Example 3.3.2 Figure 3.4 shows an example of anonymization space corresponding to the IP and ID attributes and the PU/UD rules for two data users, Alice and Charlie. The data owner requires hashing **(Ha)** for IPs and no anonymization **(NA)** for IDs. By the privacy-up rule, all levels inside the upper shaded area will also satisfy privacy requirements. Alice's and Charlie's utility requirements are as follows.

1. Charlie requires to preserve the one-to-one mapping for both IPs and IDs. Following the utility-down rule, the dark gray area highlights all the levels that satisfy Charlie's

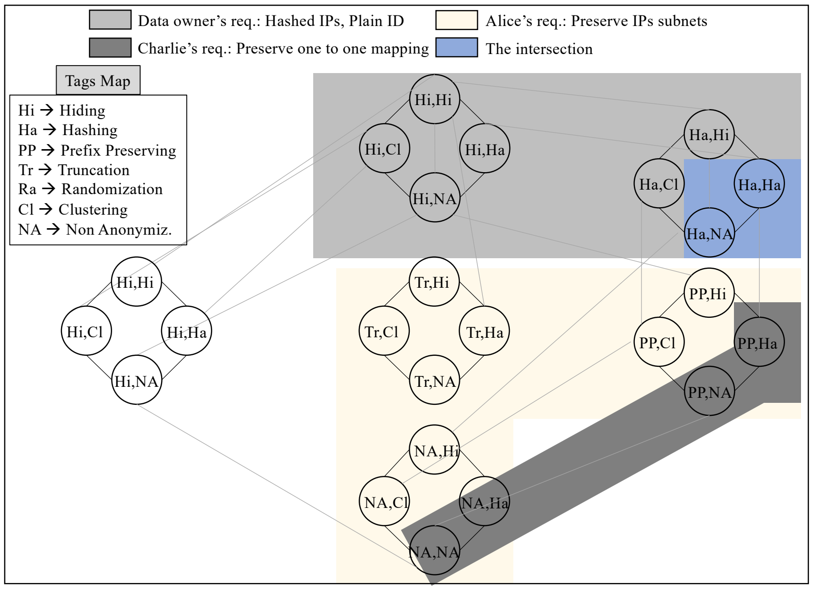


Figure 3.4: An example of *anonymization space* for attributes IP and ID, and the *PU/UD* rules for Alice and Charlie.

utility requirements. Also, the area with crossing lines includes all levels that satisfy both the privacy and utility requirements, i.e., $\langle$**Ha**, **Ha**$\rangle$ and $\langle$**Ha**, **Na**$\rangle$.

2. Alice requires to preserve the IP subnets. The light gray area highlights all the levels that satisfy Alice's utility requirement. Since there is no intersection between the upper shaded area and the light gray area, no level can satisfy both the privacy-up and utility-down rules, which means no anonymization primitive can satisfy both the privacy and utility requirements for Alice. However, the anonymization space makes it easy to choose an alternative level that will satisfy the privacy requirement while providing the best possible utility to Alice, e.g., $\langle$**Ha**,**Na**$\rangle$.

3.3.4 Machine Translation

To ensure a user-friendly interface, *iCAT* permits both the data owner and the data users to express their requirements in English. As a ramification, the primary challenge of translating any requirement is to understand that requirement linguistically. To overcome

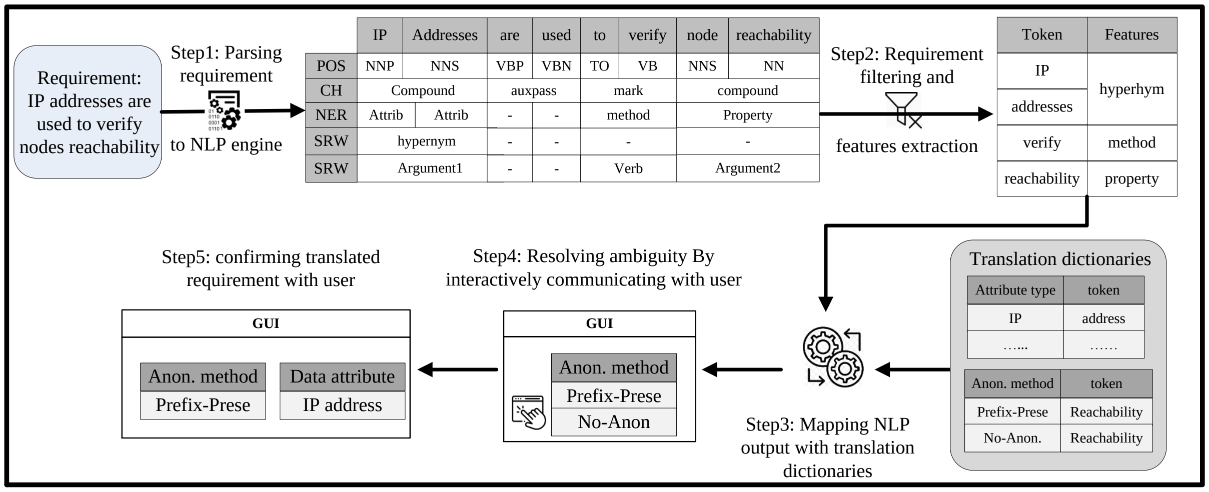

Requirement: IP addresses are used to verify nodes reachability
Step1: Parsing requirement to NLP engine
IP | Addresses | are | used | to | verify | node | reachability
POS NNP NNS VBP VBN TO VB NNS NN
CH Compound auxpass mark compound
NER Attrib Attrib - - method Property
SRW hypernym - - - -
SRW Argument1 - - Verb Argument2
Step2: Requirement filtering and features extraction
Token | Features
IP, addresses: hyperhym
verify: method
reachability: property
Translation dictionaries
Attribute type | token
IP | address
…... | ……
Anon. method | token
Prefix-Prese | Reachability
No-Anon. | Reachability
Step3: Mapping NLP output with translation dictionaries
Step4: Resolving ambiguity By interactively communicating with user
GUI
Anon. method
Prefix-Prese
No-Anon
Step5: confirming translated requirement with user
GUI
Anon. method | Data attribute
Prefix-Prese | IP address

Figure 3.5: An example to explain the requirement translation process.

this challenge, *iCAT* leverages a convolutional neural network (CNN) architecture [80] to form a natural language processing (NLP) tool. The input of this tool is an English sentence representing the requirement, and a set of language processing predictions are the expected outputs. However, this deep neural network architecture should be trained in an end-to-end fashion to process the input sentence by several layers of feature extraction. Unlike other NLP tools, the extracted features range from semantic to syntactic constituent. Table 3.2 shows the NLP standards which are used here.

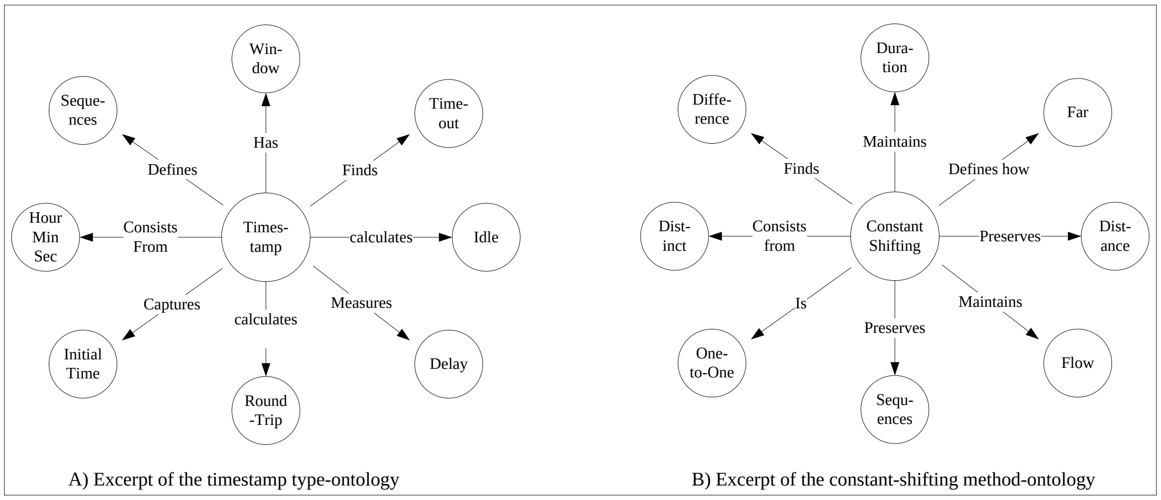


Figure 3.6: Ontologies of A) timestamp type-ontology and B) constant shifting method-ontology.

Table 3.2: A list of NLP features used in *iCAT* and their definitions.

NLP Feature	**Meaning**
Part-Of-Speech (POS)	Labeling each word with a unique tag that indicates its syntactic role (i.e., plural, noun, adverb)
Chunking (CH)	Labeling segments of a sentence with syntactic constituents (i.e., noun phrase (NP) or verb phrase (VP)).
Named Entity Recognition (NER)	Labeling atomic elements in the sentence into categories (i.e., attribute, method, property)
Semantic Role Labeling (SRL)	Giving a semantic role to a syntactic constituent of a sentence.
Semantically Related Words (SRW)	Predicting whether two words are semantically related (i.e., synonyms, holonyms, hypernym)

As an example, Figure 3.5 shows how a data owner's requirement (e.g., "IP addresses are used to verify nodes reachability") is processed to obtain the attribute data type *IP* and the associated anonymization primitive *Prefix-Preserving*. Since the aforementioned requirement may have multiple interpretations for the anonymization method, the user interacts with the tool through a GUI interface to solve the issue. This will be further discussed in Section 3.3.5.

3.3.5 Ambiguity Resolution

If a particular requirement has multiple numbers of translation candidates, this may lead to an ambiguous condition, i.e., which one from those translation candidates should be chosen. The reasons behind such ambiguous situations and the corresponding solutions are as follows.

- At the requirement parsing step, due to the typos or NLP failures, the sentences entered by the user might be mistakenly parsed. As a ramification, the requirement translation fails and the user has to re-enter the requirement.

- On the other hand, a particular requirement can be translated into different anonymization methods. As an example, we may consider a requirement, *Req-1: each IP address must be mapped to one IP address* and that can be satisfied with both the IP hashing and the prefix-preserving. In this case, the ambiguity solver of *iCAT* will display a small multi-choice menu to the user, such that this ambiguity can be resolved interactively, with the click of a button. This is worth mentioning that we evaluate

Table 3.3: Different datasets used in evaluating *iCAT*.

Datasets	**Category**	**Format**	**Records**	**Attributes**	**Requirements**
DS1: Google cluster	Real	CSV	2,000	9	56
DS2: OpenStack Neutron	Synthetic	log	2,000	18	62
DS3: OpenStack Nova	Synthetic	DB	2,000	22	44
DS4: BHPOBS ML	Real	text	1,027	22	43

Table 3.4: Distribution of participants over the user experience levels.

Category	**Research**		**Industry**	
Expertise Level	**M.Sc.**	**Ph.D.**	**Junior**	**Senior**
Participants percentage	30.4%	8.6%	43.4%	17.6%
Overall percentage	39%		71%	

the user selection and propose a solution to help him/her choose the right translation from the multi-choice menu.

- On the other hand, a requirement even can be expressed in many different ways. For example, one requirement states that the sequence of events in the logs should be maintained, while the other requires that the correlation between the logged records should be preserved. And in both requirements, the order of the data is mandatory for the analysis task and should be preserved to serve the use purpose.

- There is also a possibility that a data user's minimum requirement leads to the utility level being higher than what is allowed by the data owner according to his/her privacy requirements. In this case, *iCAT*, suggests alternative anonymization primitives that offer the closest utility level to what is specified by the data owner. A further discussion is presented in Section 3.3.8, while we describe the requirement mapping in detail.

3.3.6 Anonymization Space Creation and Requirement Mapping

In this section, we describe the procedures of building the anonymization space and mapping the requirements on that space in an appropriate way to ensure the privacy of the data owner and the utilities for data users.

3.3.7 Anonymization Space Creation

iCAT creates an anonymization space based on the data received from a data owner to provide more choices to users. Different steps of creating such a space are as follows. To identify the Anonymization Space modeling, and hence to build the anonymization space lattice, first, *iCAT* loads the available data from the data owner, deletes empty rows or columns and converts the dataset into data frames, and detects all the data attributes and their types based on pre-defined patterns. For example, the time and date format, the IP format, the IDs based on sequence numbering or predefined patterns from the data owner. Second, *iCAT* allows the user to perform several data filtering operations manually or automatically to remove records from data. For example, column deletion, row deletion, frequency deletion based on the number of occurrences of a text, searched deletion based on the existence of a keyword. Thus, the anonymization space lattice is generated by identifying the attribute-type lattices corresponds to each data attribute in the input data. Finally, those lattices are multiplied together to generate the privacy/utility access control model as explained in section 3.3.2.

3.3.8 Requirement Mapping

All the requirements are mapped with the anonymization primitives as follows.

Ontology Modeling. We utilize ontology modeling to define the relationship between requirements and data attributes/anonymization primitives. To explain the *ontology learning* process, first, we define the following concepts for data owners and users: i) *anony-methods*; ii) *method-functionality*; iii) *attribute-types*; and iv) *attribute-synon*. The instances of the *anony-methods* are the existing anonymization primitives and the *method-functionality* instances are manually created based on the functionality and unique properties that each anonymization primitive can achieve. Moreover, the instances of the *attribute-type* concept are the given attributes types and the *attribute-synon* instances are manually created based on the use/synonymous of each attribute type. Note that those instances can be updated accordingly based on the user interaction with *iCAT*, whenever a requirement is failed to translate by the NLP module as we mentioned earlier. After that, we find the relationships between those concepts' instances by defining relations between the *anony-methods* and the *method-functionality* concepts. Also, by defining relations between the *attribute-types* and the *attribute-synon* instances. As an example, Figure 3.6 shows that the type-ontologies related to the *timestamp attribute* type and the *method-ontology* related to the constant shifting anonymization primitive. After that, we store the resulted ontologies into two separate tables, namely, the *type-ontology* and the *method-ontology*.

Requirement Matching. For requirement matching, the learned ontologies are applied to the processed and the filtered requirements provided by the NLP to find the data attributes and the anonymization primitives. Every tokenized word in the processed requirement is matched with the attribute type and the anonymization method ontology tables as shown in Figure 3.5 and discussed as follows.

- For each tokenized word of each annotated requirement, first, the tokenized word is matched with the type ontology and then with the method ontology.
- If the tokenized words are mapped to only one record from the attribute type ontology table and one record from the attribute method ontology table, then the requirement is translated properly, and the mapper will pass to the second requirement.

- If none of the tokenized words matches any record in both the type and the method ontology tables, the word is removed from the sentence annotation table.
- If the user tokenized words fail to map to any record from the type and/or the method ontologies or if the tokenized words have multiple matching, then the mapper will return an error message to the user reporting this issue and forward this conflict to the ambiguity solving process.

3.3.9 Permission Granter and Anonymization

After translating all the requirements and generating the corresponding anonymization space lattice, the data users are allowed to access only that portion of the anonymization space which is approved by the data owner. Hence, *iCAT* associates the data user identity with the privacy level specified by the data owner, and that is required to determine the anonymization sub-space assigned to them based on the PU/UD access control rules as discussed in Section 3.3.3. Finally, based on the granted anonymization primitives, data users can choose among different anonymization combinations to anonymize the data and get the final anonymized output.

3.4 Evaluation

In this section, we measure *iCAT*'s effectiveness and performance through experiments using both synthetic and real datasets. Additionally, we evaluate *iCAT*'s usability through a user-based study with participants from both industry and academia working on data analysis.

3.4.1 Experimental Setup

Our experimental setup for the evaluation is as follows.

Dataset Specification. We consider four different datasets, i.e., DS1, DS2, DS3, and DS4 for the experimental evaluation of the *iCAT*. DS1 is the Google cluster dataset [76] (i.e., traces from requests processed by the Google cluster management system), while DS2 is cloud logs collected from OpenStack Neutron services (i.e., the networking service of Openstack). DS3 is a database dump of the OpenStack Nova service, and DS4 is the BHP-OBS machine learning dataset [81]. We select these datasets for the following reasons: (i) the privacy constraints and requirements are already known for datasets from the industrial collaborator; (ii) these public datasets are widely used in research labs [76]. In Table 3.3, we provide more details about the selected datasets (i.e., category, format, number of records, etc.).

User-based Study Specification. To evaluate the usability of *iCAT*, we prepared questionnaires and conducted a survey among people involved in university research and industry. In this process, we considered two types of participants, i.e., data owner participants and data user participants. To solicit participants, we placed a flyer on the university campus and also sent it to our industrial collaborators. The on-campus flyer requires that: i) participants should be able to pose clear requirements (e.g., how to use the data and what properties need to be preserved); ii) participants should be able to evaluate the usefulness and usability of the data after the experiments. On the other hand, the request sent to the industry indicates that: i) participants should be able to write their institutional privacy constraints and requirements that govern data sharing; ii) participants should be able to verify whether the final anonymized output of the data meets those requirements/constraints. At the end of this data acquisition procedure, we received feedback from nine researchers from different university labs and fourteen participants from four industrial organizations. Table 3.4 summarizes the participants' experience level for each category in percentage, where we categorize them based on their educational level and industrial experience (i.e., for Research: M.Sc. and Ph.D., and for Industry: junior and senior).

Table 3.5: An evaluation of the failed translation of requirements.

Datasets	Requirements	data owner			Data user	
		Utility loss	Manual validation	No-translation	Utility loss	No-translation
DS1	56	2	3	0	4	0
DS2	62	0	2	0	7	0
DS3	44	1	2	0	2	0
DS4	43	4	4	0	5	0

Procedures. We divided our study into four data anonymization operations based on the considered datasets and asked the participants to select one of those four; corresponding to their domain. After that, the participants had to input their requirements and interact with *iCAT* until the anonymization operation was completed. Finally, we asked the participants to fill a post-experiment questionnaire to report the correctness or satisfaction level of the usefulness of data and the privacy constraints. We also recorded the requirements entered by the participants to evaluate the effectiveness of *iCAT*.

3.4.2 Effectiveness of Requirements' Translation

Since this is a multi-class problem, we calculate the percentage of correctly translated requirements to measure the effectiveness of the system (in terms of translation capability) for four different datasets depicted in Table 3.3. Hence, we manually investigated the recorded user's requirements and categorized the failures as follows: i) the privacy leakage/utility loss caused by both data owners/users through wrongly chosen anonymized methods; ii) the failures caused due to misinterpretation of *iCAT* on either the data owners or the data user's requirements or both.

Figure 3.7.A depicts that the overall effectiveness of translating data owners' requirements is significantly high, while DS2 shows the lowest accuracy but even that is 97.1%. The primary reason behind such higher accuracy is our highly efficient CNN model for the NLP in language processing predictions and consequently assists in developing a rich ontology table. However, any failure of the NLP (e.g., typos of user's input can easily guide the NLP to make a wrong interpretation) may have a significant impact on overall

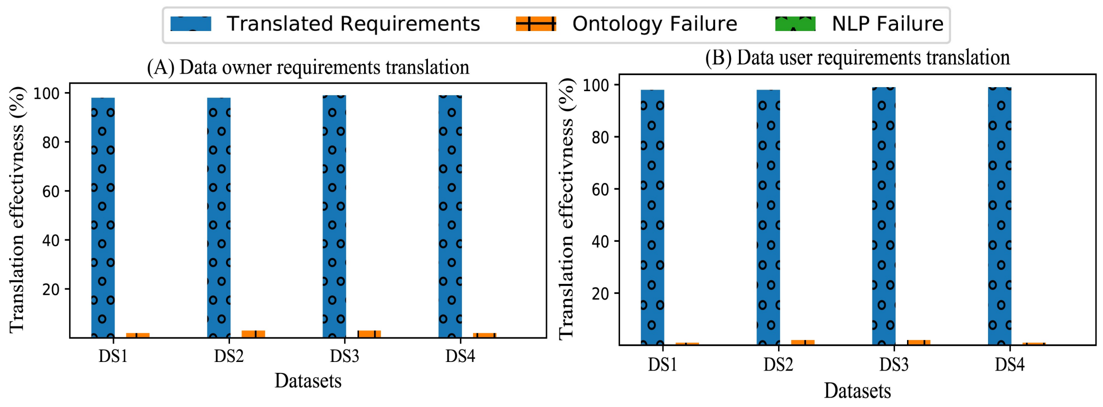


Figure 3.7: The effectiveness of requirement translation at A) data owner and B) data user sides.

performance. After finding such translation failure, *iCAT* immediately triggers the ambiguity solver for asking a manual interpretation. This solver reduces the error rate through interactive communication with the users, where they can directly intervene in the case of any uncertain requirement as we discussed earlier. Hence, there is no failure reported from the ontology modeling mapping as depicted in Figure 3.7.A. Not only for data owners, but the ambiguity solver also assists in attaining high accuracy in the translation of data users' requirements as depicted in Figure 3.7.B.

The translation accuracy is also influenced by the number of attributes. Table 3.5 depicts that a higher number of attributes may lead to a higher probability of translation failure. The reason behind such a negative influence is that while the number of attributes is higher, there is a higher probability that the same attribute type may appear multiple times in the dataset and hence, causes a translation failure. As an example, the dataset DS2, in Table 3.3, is a cloud log dataset, and the attribute ID appears five times (i.e., project ID, tenant ID, event ID, VM ID, and host ID). The user has to be precise in writing his/her requirement to differentiate between these attributes when s/he writes that requirement which is involved with the ID attribute type. As a ramification, if a user requirement is not precise enough to differentiate between the attributes of the same type, the translation operation will fail because the tool will not be able to select the relevant attribute to the entered requirement.

We also observe that data user participants are not aware of all existing anonymization methods. They often fail to understand the mapping between anonymization primitives suggested by *iCAT's* ambiguity solver and their utility requirements. For example, the data users are not able to differentiate between the privacy level of the prefix-preserving and constant substitution anonymization primitive. This observation has led us to add a pop-up message showing an example of each primitive to guide the user to avoid selecting the wrong suggestion. Finally, Table 3.5 shows a comparative analysis of the number of failed requirements presented in Figure 3.7. We can only observe privacy loss from data owners' side due to failures in the NLP modeling. Utility loss could be caused at data users' sides due to an incorrect translation of data owners' requirements and a misinterpretation of the anonymization methods by data users.

3.4.3 Usability

Since there are many existing anonymization tools in practice, along with the performance comparison of *iCAT* with these existing tools, we also intend to evaluate its acceptance probability on mass users. For this purpose, we conduct a survey based on two questionnaires as we mentioned earlier. The first questionnaire follows the standardized usability questionnaire [2] and consists of 19 questions. This questionnaire determines the users' satisfaction towards the services provided by the tool (e.g., whether this tool converges the views and bridges the gaps between data owners and users). On the other hand, the second one surveys the sensitivity of the attributes and the trust level in different actors used to propose privacy/utility access control mechanisms for different attributes anonymization.

The surveys are summarized in table 3.6, where we categorize the evaluation criteria and rate their respective average score out of seven, as instructed in the used questionnaire [2]. The results indicate that the data users are satisfied with being a part of the anonymization process through expressing their requirements. On the other hand, the data owner participants from the industry clearly show interest in this tool for being able in owning different anonymization levels of the same input data instead of the encrypt/hide policy which they usually use. Data users also report that the tool requires some privacy expertise, especially during the implementation of the ambiguity solver. As mentioned earlier, to handle such issues, we have revised our design by adding concrete examples for the anonymization primitives to make them more understandable.

Table 3.6: The results of usability based on a questionnaire designed following [2].

Category	Question	Score/7
Ease of use, interactivity, and user friendly	It was simple to use *iCAT*	6.3
	I can effectively complete my work using *iCAT*	5.2
	I am able to complete my work quickly using *iCAT*	4.8
	I am able to efficiently complete my work using *iCAT*	5.45
	I feel comfortable using *iCAT*	5.7
	It was easy to learn to use *iCAT*	4.2
	I believe I became productive quickly using this system	6.4
	The interface of this system is pleasant	6.5
	like using the interface of this system	6.6
Errors detecting, reporting and recovery	*iCAT* gives error messages to fix problems	5.7
	I recover easily/quickly when I make a mistake	5.8
iCAT does not need support/ background to use	It is easy to find the information I needed	4.4
	The information provided for *iCAT* is easy to understand	3.5
	The information is effective in completing the tasks	3.6
	The information organization on *iCAT* screens is clear	5.7
This system has all the functions and capabilities I expect it to have Comment		6.1
The information provided with this system is clear (e.g., online help and other documentation)		NA
The overall satisfaction	I am satisfied with how easy it is to use *iCAT*	5.3
	I am satisfied with this system	6.2

3.4.4 Evaluation of Resource Consumption

To evaluate the overhead from different modules of *iCAT*, we intend to estimate the required time, memory, and CPU consumption. All the experiments are performed on a

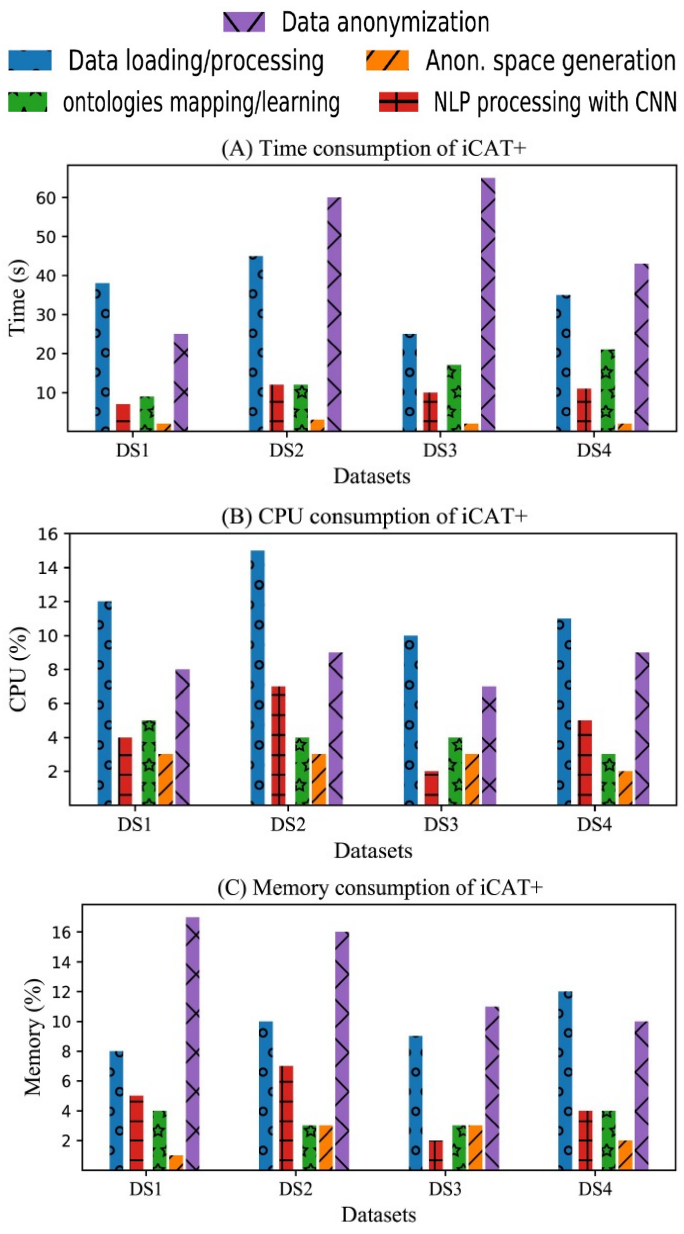


Figure 3.8: The resources consumption by *iCAT* for anonymizing the selected datasets: A) Time consumption; B) CPU consumption; C) Memory consumption.

machine running the MACOS 11.2 operating system equipped with Intel Quad-Core i5 CPU 3.8GHz and 16GB 2400 MHz DDR4 RAM.

Figure 3.8 depicts the required time, memory, and CPU consumption of the data anonymization process for four different datasets. We measure these resource consumptions for five distinct events: i) Data loading and pre-processing (i.e., data owner side only); ii) Anonymization space and access control matrix generation (i.e., data owner side only); iii) Ontology mapping and learning (i.e., on both data owner and data users sides); iv) NLP processing using the CNN results (i.e., on both data owner and data users sides); v) The resource consumption of data anonymization (i.e., data owner side only). Figure 3.8 illustrates that from the data owner side, after a one-time effort to load the data, other operations have negligible consumption. On the other hand, the overhead resulting from NLP processing and data anonymization is mainly related to the original implementation of these models and does not require too long to be operated [80].

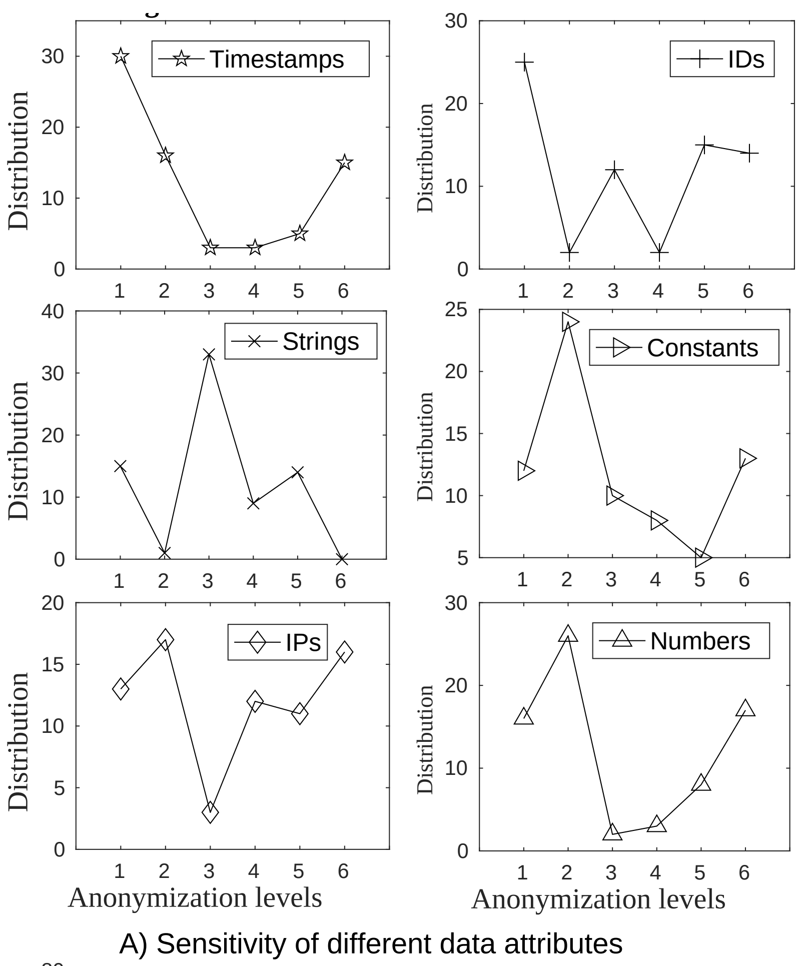
3.4.5 A Study on the Size of Anonymization Space

We study the impact of three publicly available data sets, which are widely used by the researchers [82], on the anonymization space or its size. The selected data sets vary from network traces to cloud logs and IoT data, etc. The main objectives of this study are: i) to measure the anonymization space size for different datasets; ii) to emphasize that the anonymization decision by data owners (i.e., represented by the selection from the multi-choice menu when ambiguity occurs) can vary the privacy/utility level of the final anonymized output.

Table 3.8 depicts the size of the anonymization space (i.e., the total number of anonymization combinations that can apply to the corresponding dataset). Based on the data owner's privacy requirements, a sub-space is selected for the data user and finalized the information to the final anonymized output. As per our best knowledge, *iCAT* is the first tool to allow

Table 3.7: Users' feedback on the multi-level anonymization and its analysis: Sensitivity of different data attributes;

Attribute	Actor	Level1	Level2	Level3	Level4	Level5	Level6
Time	I	95%				5%	
	E	45%	38%	6%	6%		5%
	R	25%	50%	10%	5%		10%
	C	5%			5%	20%	70%
ID	I	80%	5%	5%		10%	
	E	5%		70%		20%	5%
	R	50%	5%	5%	10%	20%	10%
	C	10%				25%	65%
String	I	55%	5%	40%			
	E			70%	15%	15%	
	R	25%		60%	5%	10%	
	C			20%	25%	55%	
IP	I	75%	20%		5%		
	E		35%	15%	20%	20%	5%
	R		40%		40%	10%	10%
	C					25%	75%
Constant	I	40%	40%	20%			
	E		55%	20%	10%	5%	10%
	R	45%	10%	30%	5%	10%	
	C	25%	5%			15%	55%
Number	I	60%	30%			5%	5%
	E	25%	50%	5%			20%
	R	5%	50%	5%	20%		20%
	C		5%			45%	55%



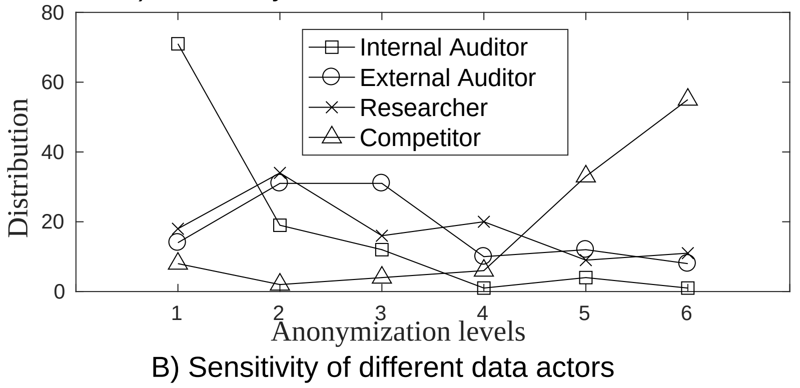


Figure 3.9: Users' feedback on the multi-level anonymization and its analysis: Sensitivity of different data actors.

data owners to determine the location of the final anonymized output from a utility and privacy point of view. Unlike other tools, *iCAT* is able to quantify the final anonymized output in terms of utility and privacy concerning all other anonymization possibilities.

Table 3.8: The size of the anonymization space for the selected datasets.

Datasets	Category	Source	Format	# of Attributes	Anony. space size
DS1: Google cluster	Real	UCI data repository	CSV	9	10.07 M
DS2: OpenStack Neutron	Synthetic	Generated in our lab	log	10	60.5M
DS3: OpenStack Nova	Synthetic	Generated in our lab	DB	8	1.7M
DS4: BHPOBS ML	Real	UCI data repository	text	8	1.7M
DS5: IoT data	Real	UCI data repository	CSV	11	362.8M
DS6: Network traces	Real	UCI data repository	pcap	7	280k

3.4.6 A Study on the Multi-level Anonymization

The objective of this study is to determine the need for multi-level anonymization by studying the sensitivity of the attributes and the trust level for different actors. To attain this purpose, we prepare an online questionnaire form that has been filled by participants from both academia and industry as we discussed in Section 3.4. This questionnaire asks participants to anonymize data given a set of anonymization primitives and different data receivers. The results of this questionnaire are listed in the Table 3.7. To demonstrate the trend, we also apply the marginal distribution and draw the trend of each attribute and actor of this survey as depicted in Figure 3.9. These two figures depict that the attributes and actors are associated with different sensitivity levels. The attributes e.g., *Time*, *ID*, *Constant*, and *Numbers* have similar data-sharing strategy; internal actors could have low privacy and high utility results, while competitors would be only provided with high privacy and low utility data. The main reason is that those attributes are not as sensitive as personally identifiable information, but still can leak information that can be used to stage security attacks. On the other hand, attributes *IP* and *Numbers* (e.g., salary in our survey) are considered to be sensitive attributes for all levels of actors who prefer to apply at least Level 2 anonymization on them. This can be due to sharing policies or cultural background which makes them less willing to share the information carried by those attributes. Figure 3.9 confirms the trust levels of the actors through the levels of anonymization methods they are mostly assigned. Internal auditors are mostly granted Level 1 anonymization only, while competitors could only get Level 6 anonymization results. On the other hand, external auditors and researchers (generally under a non-disclosure agreement) share similar trusted levels. This shows the participants share similar visions related to the internal auditor and competitors and consider the external auditors and researchers harmless.

3.4.7 A Study on the Satisfaction of Anonymized Data

We also investigate the understanding of different anonymization methods by data owners mainly for two reasons. First, to check whether the selected anonymization method meets the data owners' requirements and then to show the impact of the selected anonymization primitives on the privacy/utility level. For these purposes, we take a sample of OpenStack cloud data, provide it to the user study participants and ask them to anonymize it such that their privacy requirement would be satisfied. After that, we illustrate the plain and anonymized data and present it to the participant to check whether the anonymization process configured by data owners can meet their expectations.

The benefits of this selected cloud data to perform this experiment: i) the data syntax is simple and understandable (i.e., the data consists of IP addresses, IDs, and reachability rules), and ii) the data can be easily represented in a visualized form. Figure 3.10.A shows the plain data visualization and the remaining parts of the figure depict the visualization of the anonymized data using different anonymization primitives as mentioned at the bottom of each figure.

As we can see in Figure 3.10, the output of the anonymized data can vary from high utility output as shown in Figure 3.10.B where all the properties of the original data are preserved to high privacy output as shown in Figure 3.10.D where all the mentioned properties are hidden. 63% of the participant has selected the anonymization method presented in Figure 3.10.B, and only 39% of them were satisfied with the final anonymized output. On the other hand, 73% of the participants have selected anonymization methods presented

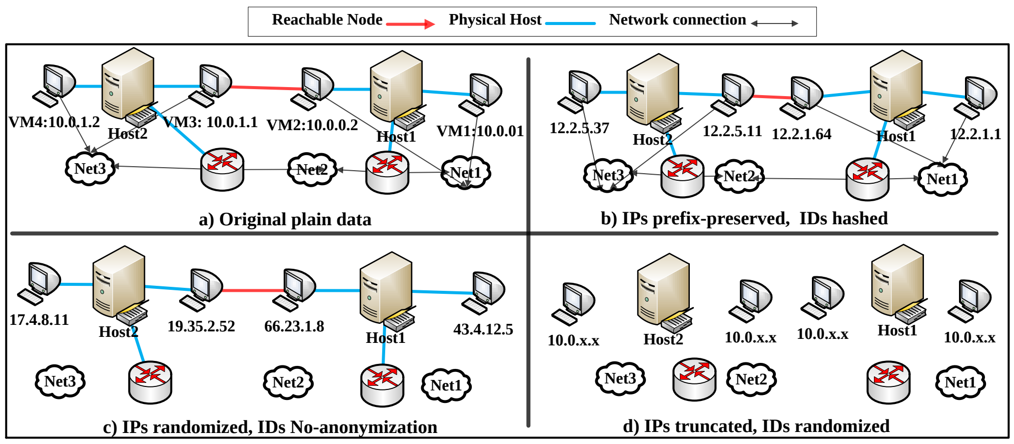


Figure 3.10: The visualization of plain and anonymized data using different anonymization methods.

in Figures 3.10.B and 3.10.C, while 84% of them are happy about the final anonymized output. Hence, it can be concluded that visualizing the data may assist the data owners in evaluating their selected anonymization primitives and their satisfaction level.

3.5 Summary

Due to a lack of understanding of the requirements as well as the non-customizability of the existing anonymization tools make this inflexible and hence inefficient to support various privacy and utility requirements of both data owners and data users. To address these issues, in this paper, we proposed an interactive and customizable data anonymization tool, namely, *iCAT*, which takes user requirements in English, automatically processes those requirements using an NLP technique, and addresses the flexibility limitations of most existing tools by creating a customizable anonymization space. *iCAT* can ensure the active participation of data users in making their own decisions. We leveraged a CNN-based NLP to make the requirements translation process automated. Since, due to typos, the designed NLP may fail to translate any requirement, *iCAT* can trigger on a feedback module to accept the manual interpretation. We made an extensive analysis based on both real and synthetic data to evaluate our proposed solution and formally achieved higher effectiveness (e.g., 98% of users' requirements were correctly translated), while the decision-making time was significantly small (e.g., 64 seconds). In addition, we conducted several user surveys and obtained quite positive feedback from the tool users who participated from both industry and academia.

Chapter 4

Auditing the Integrity of Virtual Network Functions Chain

4.1 Introduction

4.1.1 Motivation

Network Functions Virtualization (NFV) enables on-demand and cost-effective deployment of network services as chains of VNFs on top of an existing cloud infrastructure [4]. A growing trend in NFV is to host the VNFs on third-party clouds for more cost-effective deployment [83, 84, 85], e.g., DISH Network is reportedly deploying its cloud-native 5G network in AWS Cloud [84], and VMware is enabling communications service providers to accelerate the deployment of their VNFs on its VMware Telco Cloud platform [85]. In such scenarios, the provider manages all the virtual and physical resources needed for deploying the specified network services by tenants as a chain of VNFs. Despite its obvious benefits to NFV tenants, the multi-actor nature of such deployment may lead to novel security threats. In particular, potential integrity breaches may silently arise due to unintentional misconfigurations [86] or malicious intents [7] to cause harmful inconsistencies between tenant specifications and their provider deployment.

Most existing security auditing[1] approaches for NFV would face difficulties against such integrity breaches mainly due to the following challenges. First, relying on the cloud provider (e.g., [33, 87]) may be impractical (as many providers may be reluctant to take the burden of conducting security auditing on behalf of their tenants) or insufficient (as providers typically do not understand tenant-level requirements, e.g., modifying compromised switch forwarding rules may seem wrong to the tenant, but legitimate to the provider). Second, relying on the tenant alone (e.g., [88]) may not be feasible, either, as the tenant typically has limited access to provider-level data, and shipping all such data to the tenant could incur prohibitive overhead and confidentiality concern. Based on comprehensive studies and existing literature [89, 90], it becomes evident that a novel solution is required to tackle those challenges.

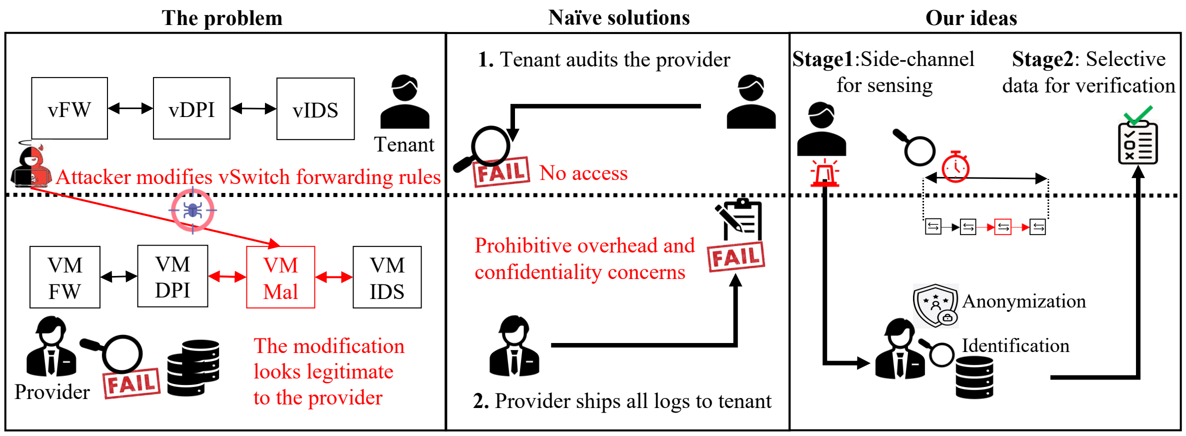

The problem
Naïve solutions
Our ideas
vFW
vDPI
vIDS
Tenant
Attacker modifies vSwitch forwarding rules
VM FW
VM DPI
VM Mal
VM IDS
Provider
FAIL
The modification looks legitimate to the provider
1. Tenant audits the provider
FAIL
No access
Prohibitive overhead and confidentiality concerns
FAIL
2. Provider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

Figure 4.1: Motivating example

High-level motivating example, naive solutions, and our proposed ideas. Figure 4.1 shows a motivating example to highlight the problem (left), limitations of naive solutions (middle), and our main ideas (right).

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

4.1.2 The problem and our main ideas

The left side of Figure 4.1 illustrates an example of an integrity breach and the limitation of provider-based auditing. Specifically, the top shows a chain of three VNFs (*vFW*, *vDPI*, and *vIDS*) specified by the tenant, and the bottom depicts that the cloud provider deploys those VNFs on three VMs. Suppose, due to an unintentional misconfiguration [91] or through the exploitation of compromised resources, an attacker (e.g., a co-located tenant) [86] can modify the switch forwarding rules to redirect traffic flowing through the chain to the malicious VM (i.e., *VM Mal*) into the chain. Since the chain now deviates from its specification, it allows the attacker to steal information, and hence, such a modification represents an integrity breach to the tenant. However, the modification may seem legitimate to the provider, as it is coming from a (seemingly) legitimate tenant, and therefore any provider-based auditing mechanism will likely miss such breaches. Therefore, relying on the provider alone may be insufficient due to its lack of understanding of tenant requirements, which motivates a tenant-based solution.

Tenant-based Naïve Solutions: The middle of Figure 4.1 illustrates the limitations of two naive solutions. In the first solution, the tenant typically has limited access to provider-level data (e.g., detailed logs or databases), and therefore it cannot directly audit the provider. In the second solution, even if the provider is willing to share, such a data transfer process may lead to prohibitive overhead and confidentiality concerns due to the multi-tenancy nature of NFV.

Our Ideas: The right side of Figure 4.1 illustrates our main ideas. Intuitively, we keep the auditing workload mostly on the tenant and only involve the provider to share selected anonymized data upon "sensing" (hence NFVSense) suspects of integrity breach. First, as illustrated by the clock and packet icons in the figure, the tenant identifies suspected integrity breaches based on tenant-level side-channel information. Second, it performs formal verification on selected provider-level data (which would be automatically identified and anonymized) to confirm (or reject) such suspects.

Specifically, we propose NFVSense, a tenant-based, two-stage approach to audit the integrity of VNF chains hosted on third-party clouds. In the first stage, NFVSense combines tenant-level side-channel information with machine learning (ML) techniques to identify potential suspects of integrity breaches. As such an approach will likely introduce false positives, the second stage automatically crafts selective data requests to the provider for each suspect from the first stage, anonymizes the data before returning it to the tenant, and finally performs rigorous verification based on such data to confirm (or reject) the suspects. We implement and integrate NFVSense into OpenStack/Tacker [92], a popular choice for NFV deployment. We evaluate the accuracy and efficiency of NFVSense through extensive experiments. In summary, our main contributions are as follows.

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

4.2 Preliminaries

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

4.2.1 Background on NFV

NFV enables the virtualization of network services and consists of two major abstraction levels as follows (according to the ETSI NFV reference architecture [4]):

1. Tenant Level: This level is specified and managed by an NFV tenant who specifies its network services as a chain of several VNFs, e.g., virtual firewalls and IDS.

2. Provider Level: This level is managed by a cloud infrastructure provider who instantiates the tenant's specifications of VNFs using both virtual resources, e.g., VMs, and physical resources, e.g., CPU and memory.

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

In the following, we explain our rationale for choosing performance-related side-channel information and then outline the challenges that come with this side-channel.

Our Rationale. We choose NFV performance-related side-channel information since, as Table 4.1 shows, there exist performance bottlenecks at different NFV abstraction levels, which can provide useful input for identifying any integrity breaches [93, 94, 95, 38], e.g., the impact from the hardware level can be more severe than the virtualization level [38]. The last column of Table 4.1 shows the parameters we choose to extract side-channel information at the corresponding level (the parameters selection are discussed in Chapter 4.3 and evaluated in the experiments Chapter 4.4).

Challenges. Using NFV performance side channels for identifying integrity breaches faces the following challenges, which will be addressed in Chapter 4.3:

- Limited Data Access: Since tenants cannot access the provider-level configuration, system-wide profiling tools to monitor hardware-based performance counters (e.g., cache-references or cross-system events, such as LinuxPerf [96] and OProfile [97]) cannot be utilized for our purpose.

- Performance Sensitivity and Probing Overhead: Network performance measurements in NFV can be highly sensitive to different factors (e.g., VNF's functionalities [34] and network workload [93]). Therefore, performance measurements must be repeated under different combinations of workloads and other parameters to ensure sufficient coverage.

- Fallacy of Profiling Standalone VNFs: The existing studies (e.g., [88, 98]) show that performance measurements of VNFs can change depending on their relative order in a chain. Also, adding or removing VNFs from a chain may affect the overall performance [98]. Consequently, establishing the performance profile of a chain can only be done by continuously measuring and analysing the performance of the chain as a whole after deployment [34].

- Other Performance Factors: There exist other factors that impact the network performance, such as the status of the VNF (e.g., active/passive) which may affect I/O waiting and processing time. The network I/O overheads in the virtual switches can affect the performance [34].

4.2.3 Threat Model

The basis of our work is the "trust but verify" principle behind most security auditing techniques. More specifically, we assume the cloud provider and its infrastructure are both trusted by the tenant, but the tenant may still be concerned about unintentional user mistakes (made by cloud operators working for the provider) or stealthy attacks (which

Table 4.1: Excerpt of existing NFV performance bottlenecks.

Level	**Bottleneck**	**Impact**	**Parameter**
Physical	Memory size, number of CPUs, disk operation and hypervisor type	Variation in packet processing time [38]	Unique gap in RTT
Virtual	I/O interrupts, number of ports, virtual switching	VNF performance degrades [95]	Probing window/ workload
Virtual resources	Stateful and stateless connections, VNF functionality, VNF image	Longer packets delivery time [93]	Connection type and VNF type

evade detection by the provider). Therefore, our solution is not meant to replace existing security mechanisms (e.g., security auditing and attack detection) of the provider, but rather to give the tenant additional control through performing independent tenant-based auditing.

Under such assumptions, we focus on a specific class of *in-scope threats*, i.e., integrity breaches in the form of invisible (to the tenant) modifications to the provider-level deployment of VNF chains, e.g., injection of malicious resources [50, 31], traffic redirection to bypass VNFs such as firewalls [6], reduction in virtual and physical resources [11], etc. We assume such threats are not thwarted by the provider because they come from external attackers who exploit zero-day attacks, or from malicious insiders, or unintentional user mistakes or misconfigurations caused by cloud operators themselves.

The out-of-scope threats include any attacks that can breach the integrity of network services without affecting network performance at any NFV layer. Also, we do not consider attacks that are directly visible to a tenant (e.g., removing a VM and its corresponding VNF) and thus do not require our solution. Moreover, similar to most side channel-based solutions (e.g., [99, 91]), we do not consider truly malicious providers or powerful attackers who have full control over the VNFs and thus can tamper with the captured performance results or the VNFs running NFVSense, or who can launch adversarial attacks against our tool (e.g., compromise the VNF chain at deployment time, or tamper with the training process using malicious samples). Finally, our work focuses on identifying integrity breaches (the consequences) rather than detecting or preventing attacks (the causes).

4.3 NFVSense

This Chapter presents the methodology of our solution.

4.3.1 Overview

As shown in Figure 4.2, NFVSense is comprised of the following two stages: Stage 1: *Identifying Suspect Breaches*, and Stage 2: *Selected Data Verification*.

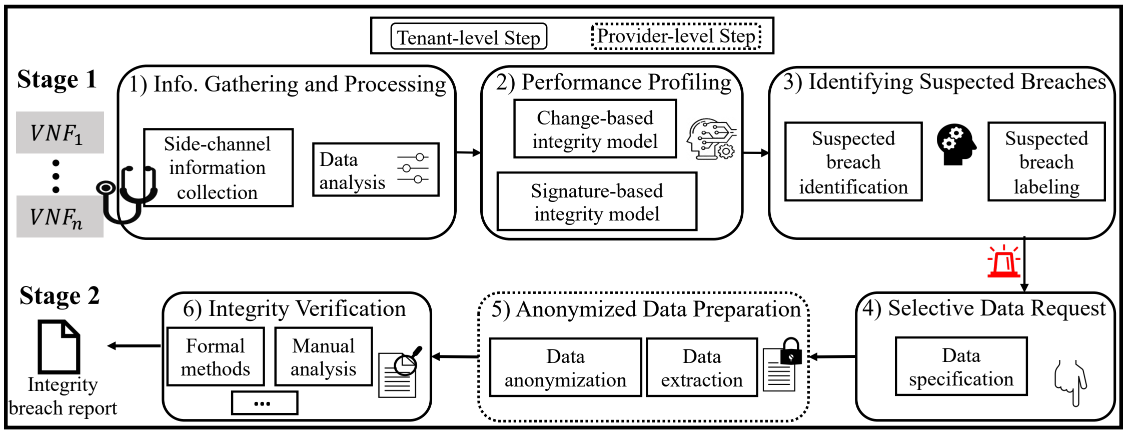

Tenant-level Step
Provider-level Step
Stage 1
VNF_1
VNF_n
1) Info. Gathering and Processing
Side-channel information collection
Data analysis
2) Performance Profiling
Change-based integrity model
Signature-based integrity model
3) Identifying Suspected Breaches
Suspected breach identification
Suspected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

Figure 4.2: An overview of NFVSense.

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

1. *Information Gathering and Processing:* When the chain is up and running, NFVSense gathers tenant-level performance measurements of the deployed topology as side-channel information, such as Round Trip Time (RTT). Using network performance tomography and active probing, it first characterizes the RTT between all pairs of VNFs in a tenant chain. Then it processes the collected data (i.e., outlier detection and filtering) to prepare for profiling in the next step.

2. *Performance Profiling:* Based on the impacts on the performance measures of VNFs, NFVSense identifies breaches by profiling the processed performance information as follows. It first learns an *Identification Model* (i.e., a binary classification ML model) to identify normal behavior and abnormal behavior (resulted from integrity breaches) and then learns a *Classification Model* (i.e., a signature-based multi-class classification ML model) to classify different types of breaches based on their impact on performance.

3. *Identifying Suspected Breaches:* By implementing two ML models (from Step 2), NFVSense identifies suspected breaches in VNF chains and classifies them. Specifically, it first checks the current network performance of all VNF pairs in the chain with the *identification model* to identify any suspected breach, then determines its type by using the *classification model*. Finally, based on the type of the suspected breach, a tenant queries the provider for selected data (e.g., port forwarding rules for a set of VMs) for further verification in *Stage 2*. We will further elaborate *Stage 1* in Chapter 4.3.2.

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

1. *Selected Data Request:* To minimize the overhead for the provider, NFVSense only asks for selected provider-level data relevant to a suspected breach based on its source and type. For instance, if NFVSense suspects an integrity breach from a malicious VM injection (as in Figure 4.1), it only requests for the path forwarding logs of the VMs in the VNF chain corresponding to the suspected breach.

2. *Anonymized Data Preparation:* The main goal of this step is to provide property-preserved output that is suitable for auditing tasks. NFVSense leverages iCAT [100] to anonymize the selected provider-level data to ensure both the privacy requirements

of the provider and the utility requirements of the tenant. Finally, the anonymized data is forwarded to the following steps.

3. *Data Verification:* To confirm an actual integrity breach from the suspects, NFVSense leverages existing verification solutions (using formal methods [101]) or performs a manual inspection on the anonymized data sent from the previous step. If the suspected breach is confirmed (e.g., VM port forwarding rules are modified), the tenant asks the provider for further action (e.g., mitigate the breaches). We will elaborate on *Stage 2* in Section 4.3.3.

4.3.2 Stage 1: Identifying Suspected Breaches

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

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

To that end, NFVSense collects data from all VNF pairs after establishing an active probing connection between all VNF pairs. For instance, if n VNFs are in the chain, $(n * (n-1))/2$ active probing connections need to be established to cover all pairs of VNFs. For each connection, different parameters (i.e., probing rate and probing window, connection type) are also varied to profile the network behavior for different settings. After that, NFVSense collects RTT values in the probing responses accordingly and characterizes them by correlating the RTT values with the corresponding VNF chains. For this purpose, we adopt Network Performance Tomography (NPT) [102] of NFV which only requires VNF's usual packet forwarding behavior. Specifically, we integrate a performance measurement tool (e.g., IPerf [43]) into the VNF images.

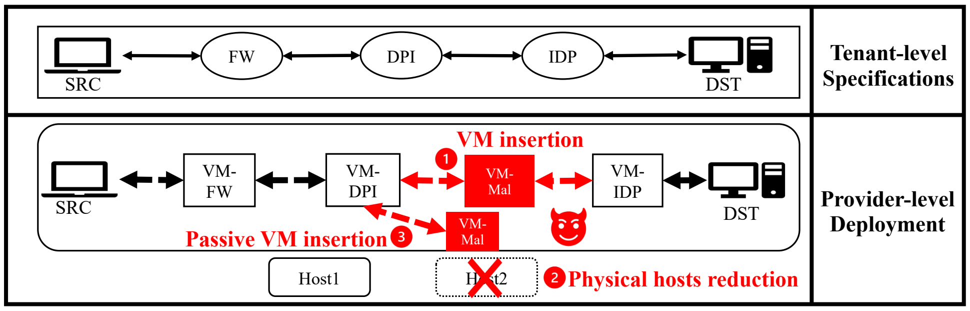


Figure 4.3: Example of different primitive scenarios for integrity breaches: i) $Scenario_1$: VM injection; ii) $Scenario_2$: Physical hosts reduction; and iii) $Scenario_3$: Passive VM injection.

Example 4.3.1 *For the sake of illustration, we assume four NFV setups corresponding to four different integrity breaches scenarios as follows: i)* $Scenario_0$*: Initial integrity breach-free setup at* $time_0$ *before the attacker performs any modification; ii)* $Scenario_1$*: Malicious VM injection (marked as ① in Figure 4.3); iii)* $Scenario_2$*: Reduction in physical hosts (marked as ②) and iv)* $Scenario_3$*: Traffic redirection to a malicious VM (marked as ③). For each scenario, the NPT is collected between all pairs of VNFs in the chain while different parameters like probing workload, probing window, connection type, etc. are also varied. Note that the methodology of NFVSense can be applied to other types of breaches as well, and also note that* $Scenario_1$ *to* $Scenario_3$ *are integrity breaches (i.e., the consequences) rather than attacks (hence NFVSense can identify such breaches regardless of the attacks causing the breaches).*

To process the collected data for profiling, NFVSense: i) identifies and filters out the outliers (e.g., extremely high RTT for the first few probing packets compared to the rest [34], or lost probing packets) in the network performance measurements using the Interquartile Range [103]; ii) merges all the features in the collected data, namely, the number of hops between source and destination of probes, VNF functionality type, probing rate and probing window; and iii) adds two ground-truth data labels to each probing record: *integrity breach* for indicating whether there is an integrity breach in a node pair and *breach type* (e.g., $Scenario_1$-$Scenario_3$ as mentioned in Example (1) from which those measurements are collected.

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

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

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

- NFVSense identifies the suspected breaches in a VNF chain by using the *identification model*. Specifically, this model compares the similarity of the measured performance features from the probing packets with the learned performance profiles (i.e., the normal behavior), and any deviation from the normal behavior indicates a suspected breach. Note that to identify suspected breaches, instead of using a single probing packet, the model tests a stream of packets to compare them against the normal profile and hence attains a higher accuracy.

- After identifying a suspected breach, NFVSense classifies that breach by using the *classification model*. Specifically, using this model, NFVSense maps the deviated behaviors in the extracted integrity breach signatures into two levels: *physical resource level* and *virtual resource level*, where the integrity breaches related to changes in the physical resource level have distinct performance gaps compared to the breaches related to the virtual resource level, as discussed in Table 4.1.

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

Example 4.3.2 *For $Scenario_2$ (refer to Figure 4.3), NFVSense identifies the suspected breaches as follows. As the packets delivery gaps between VM_{DPI} and VM_{IDP} are expected to be lower compared to the normal behavior since the virtual switch level consumes less time to deliver the traffic between these two VMs running on the same physical hosts. Consequently, this leads the* identification model *to raise an alarm of a suspected breach. Then using the* classification model*, NFVSense maps the suspected breach with the pre-defined breaches (i.e., $Scenario_1$ to $Scenario_3$ as described in Example 1). More specifically, the RTT values would be lower after the* reduction in physical host *($Scenario_2$)*

Table 4.2: An example of event logs for different cloud services in OpenStack [3]

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

compared to the malicious VM injection *($Scenario_1$), where the virtual switch level forwards the traffic to an extra VM (i.e., a malicious VM) in a VNF chain. On the other hand, $Scenario_2$ has higher RTT values compared to the* traffic redirection *breach ($Scenario_3$), where the virtual switch duplicates the traffic to the injected passive VM, VM_{Mal}. Hence, by this mapping, NFVSense determines the suspected breach as a potential* reduction in physical host *attack.*

4.3.3 Stage 2: Selected Data Verification

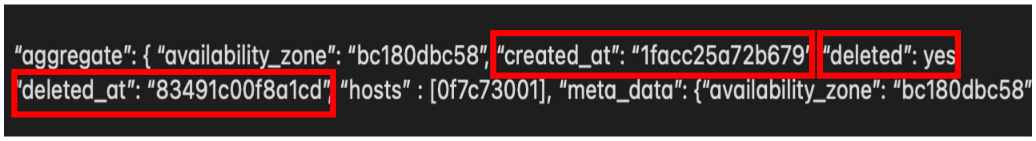
In the following, we elaborate on the steps of Stage 2.

Selected Data Request. This step is to request the provider for specific data related to the suspected breach. More specifically, NFVSense considers the decision made by Stage 1 (i.e., the identified and classified suspected breaches) to determine which provider-level data is required to verify those decisions, and requests the provider to send those related logs accordingly. If the location of the suspected breach is identified and traced to a specific pair of VNFs based on the built performance profiles and measured performance metrics, then NFVSense requests only the provider-level logs which correspond to those VNF deployments at the tenant level. On the other hand, if the class of the suspected breach is not identified (i.e., a breach is suspected but cannot be classified), NFVSense queries the log of all cloud services for a specific time range (i.e., $T_{Abnormal} - T_{Normal}$), where $T_{Abnormal}$ is the time when the suspected breach is identified, and T_{Normal} is the last time when there was no breach.

Example 4.3.3 *Table 4.2 shows an example (from OpenStack [3], a popular cloud platform) of event logs specific to different services (e.g., computing, networking, etc.) at the provider level; part of which can be requested during this step for any suspected breach. Due to a suspected breach related to $Scenario_2$, between the VNF pairs VNF_x and VNF_y, the tenant will query the provider for the ports forwarding logs of the corresponding VMs VM_x and VM_y from the* Neutron *service in OpenStack. Similarly, other selected provider-level data can be requested for other types of breaches such as OpenVswitch and Nova logs for $Scenario_1$ and $Scenario_3$.*

Anonymized Data Preparation. Though the previous step can minimize the overhead of sending all logs to the tenant, that can not ensure the privacy and confidentiality of both the provider and other tenants. For example, important network configuration information (i.e., potential bottlenecks and topology of the network) may be inferred from the logs and subsequently exploited by adversaries [100]. To address this concern, in this step, NFVSense leverages iCAT [100] to anonymize the logs, while this tool meets both the provider's privacy requirements along with the tenant's utility requirements. To that end, iCAT leverages natural language processing techniques to translate those requirements and find the most suitable anonymization primitives to meet the requirements. Hence the confidentiality concern of the provider in sharing data is addressed, while auditing at the tenant level is enabled.

Example 4.3.4 *To verify the suspected breach in $Scenario_2$, the tenant's utility requirement is the Sequence of the events must be preserved (i.e., the events must be persevered chronologically in the anonymized output). On the other hand, the provider's privacy requirement is all tenants' network topologies should be unidentifiable. Considering those requirements, the leveraged anonymization tool first determines the suitable anonymization primitives (e.g., timestamp shifting, IPs truncation, etc.) and then produces an anonymized output (i.e., ports forwarding logs).*



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

Figure 4.4: Anonymized Nova logs related to integrity breach.

Integrity Verification. NFVSense can leverage existing auditing tools or perform a manual inspection on the received anonymized data to validate the suspected breaches identified in Stage 1. To that end, we utilize a formal security verification tool, NFVGuard [101] to formally verify the alerts generated by NFVSense. NFVGuard verifies chain integrity using the formal method in two steps. First, VNF Forwarding Graph (i.e., VNFFG, a feature used to orchestrate and manage traffic through VNFs) configuration consistency properties are applied to verify consistency between the VNFFGs specification uploaded by the tenants (such as the size of VNFFG and the sequences of VNFs) and their corresponding implementation. Second, The VNFFG configuration consistency properties are applied to verify the consistency between the created SFCs and hardware implementation. If either property does not hold in the deployed NFV system, it means that there is a breach in the integrity of the underlying deployment. In practice, the tenant admins can further manually inspect the logs to verify the integrity of the SFC.

Example 4.3.5 *In the case of $Scenario_2$, the tenant will look for system event logs in the computing service,* Nova *in OpenStack, resulting from deleting a physical server. Figure 4.4 shows a snippet of the Nova anonymized logs for the* deleted host *event. From this log entry (highlighted in red), NFVSense confirms the breach from the host creation and deletion time (i.e., created-at and deleted-at) and the success of the deletion operation (i.e., deleted:Yes).*

4.4 Implementation and Experiments

4.4.1 Implementation

We implement and integrate NFVSense into OpenStack/Tacker [92], which is a popular NFV management platform. Specifically, NFVSense is implemented in Python, by leveraging the NumPy and Pandas [104] libraries for gathering NFV performance measurements (RTT for this case), and scikit-learn [105] for implementing ML models. Finally, we use our designed anonymization tool iCAT [100] to generate the anonymized proofs for verification. All the VNFs are built based on the official Ubuntu 18.04 cloud image customized by Canonical to run on public clouds [106]. We follow the model stated in [34] to configure our VNFs and use Open vSwitch [107] to manage the connectivity between them.

4.4.2 Experimental Environment

This section describes our experiments datasets and environment.

Dataset Description. We consider both real and synthetic data to evaluate NFVSense effectiveness. We collect and generate four datasets, $DS0$-$DS3$, for four scenarios (as shown in Figure 4.3), respectively. In summary, a dataset of size 4.5 GB corresponding to three different integrity breach scenarios are used, while over 136K probing requests are generated on four scenarios, where four VNF images and two physical hosts are used. During probing, we vary the traffic to four different workloads, the probing period to three different windows with two connection types (i.e., stateful and stateless). In the following, we briefly explain how data is collected.

- *Real Data:* We implement an NFV testbed to collect real data from the NFV stack. We use Python scripts to automatically generate TOSCA templates in order to deploy NFV entities, such as VNFs and VNFFGs. In our implementation, we deploy three different VNF images for widely-used network services: i) Tcpdump [108], a network data packet analyzer with the default configuration, ii) Snort [109], a network intrusion, detection, and prevention system configured with the default rules, and iii) IPtables [110], a firewall program for Linux configured with 50 rules (in a way that only the last rule matches with our probing traffic and the VNF checks all rules). On each VNF, we implement NPT for characterizing NFV performance measurements using IPerf3 [43], a tool that can produce standardized performance measurements for any network, and Bash scripting to perform active probing between all pairs of VNFs in a chain while varying different probing parameters (i.e., probing rate, probing window, connection type). For each probing packet, we log the source and destination nodes, the number of hops between them, and the resulting RTT.

- *Synthetic Data:* We provide an option in NFVSense that is only enabled when a tenant cannot gather a sufficient amount of training data for ML models due to the higher cost of continuously obtaining labeled data (e.g., via manual efforts) [111], or the overhead of substantial probing traffic using NPT [88]. This alternative option generates synthetic training data using generative adversarial network (GAN) models [112]. In contrast to most existing works that apply GAN to deceive systems (e.g., IDS [113], steganalyser [114]), we leverage GAN, similarly as in [115, 116], to generate synthetic network traffic data that would be similar to the real flow data obtained through active probing. More specifically, by generating realistic data, GAN solves the challenge of acquiring "labeled data" for training our ML model [116] along with ensuring a negligible overhead compared to the imposed overhead by the NPT step. To evaluate the effectiveness of GAN in generating synthetic data, we compare the performance of NFVSense with and without data synthesis in the next section. Note that the synthetic data generated by GAN is proposed as an auxiliary source of data, only when real data are scarce.

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

Environment Setup. All the experiments are conducted on SuperServer 6029P-WTR running the Ubuntu 18.04 operating system equipped with Intel(R) Xeon(R) Bronze 3104CPU 1.70GHz and 128GB of RAM without GPUs. Moreover, the NFV stack implementation to collect the real datasets follows the four scenarios (shown in Figure 4.3).To evaluate the accuracy (AUC) of NFVSense, we implement, train, and deploy different ML models: Decision Tree (DT), Random Forest (RF), k-nearest neighbors (kNN), and Support Vector Regression (SVR).

4.4.3 Experimental Results

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

Evaluation of Stage 1 in Identifying Suspected Breaches. The first set of experiments (Figure 4.5) is to evaluate the accuracy of Stage 1 in identifying suspected breaches. We use the dataset without a breach, $DS0$, to train the *identification model* to learn the normal behavior, and the three datasets having breaches ($DS1$-$DS3$) for validating and testing the model (10% of data are for validating and 90% of them are for testing). To avoid overfitting,

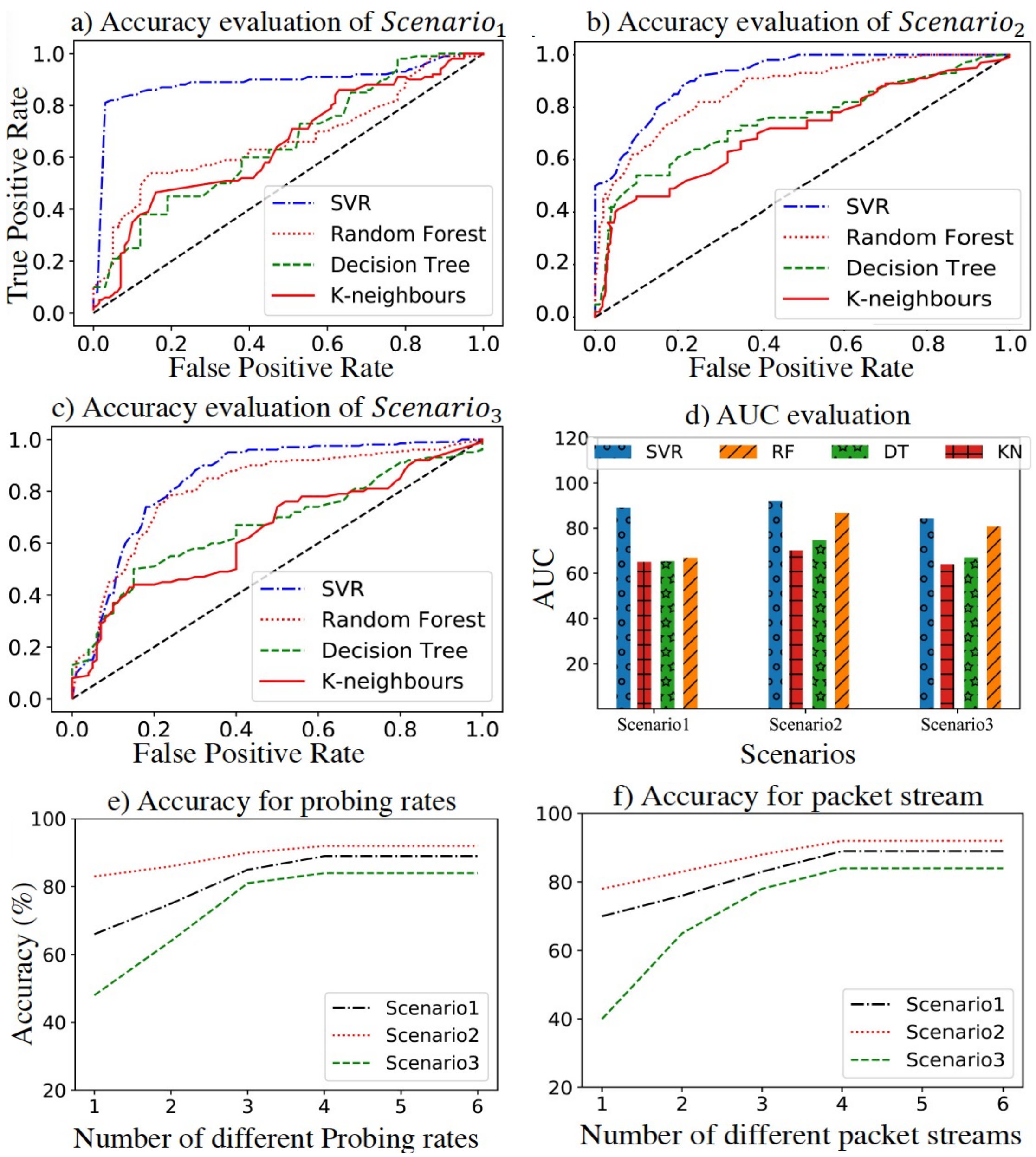


Figure 4.5: NFVSense Stage 1 evaluation.

we run our experiments up to 100 epochs. We also use an early stopping mechanism, while the trigger parameter is set to *early-stop-threshold* $= 5$, which means that during the training period, the accuracy is calculated after each epoch, and if there is no improvement in the accuracy of the validation set compared to the training set for consecutive five epochs, the training process stops. Finally, we set the length of packet trains and the number of different probing rates to four (a study on the selection of these parameters is shown in Figures 4.5.e and 4.5.f).

Figures 4.5.(a-c) show the identification accuracy of Stage 1 for different integrity breach scenarios and different ML algorithms. Among all four ML algorithms, the SVR achieves the best accuracy in all cases, as it tries to fit the best line within a threshold value that separates the normal behavior from the deviated one. The accuracy is also dependent on the integrity breach scenario as depicted in Figure 4.5.d. For $Scenario_1$ (i.e., malicious VM injection), the AUC value is about 89%, while for $Scenario_2$ (i.e., one physical host is deployed instead of two), the AUC value increases up to 91.92%. For $Scenario_2$, SVR achieves the best accuracy due to having the highest impact on the measured performance [38]. Finally, for $Scenario_3$ (an adversary inserts passive and malicious VM to the chain), Stage 1 achieves the AUC of 84.3% as depicted in Figure 4.5.c. The accuracy of the results related to this scenario is the lowest due to the performance profiles at low probing rates being very close to the scenario without a breach (i.e., normal behavior), and the effect of this scenario only appears at higher probing rates. This is because the observed delay on the end-to-end service results from the delay caused by the OvS duplicating the traffic to the passive malicious VNF, and it only creates a distinct difference when the amount of traffic to be duplicated is higher.

Figures 4.5.e and 4.5.f show the accuracy of identifying the suspected breaches by the SVR model while varying the performance profiling parameters, i.e., the length of a packet stream and the probing rates.

During measurements for each scenario, we fix one parameter and vary the other. For varying the number of probing rates, we progressively increase the number of used probing rates (in KB/s), i.e., 64, (64,128), (64, 128, 256), and so on. We observe that the accuracy increases for all scenarios when we increase the number of probing rates and the length of the packet stream. When both the length of a packet stream and the number of probing rates

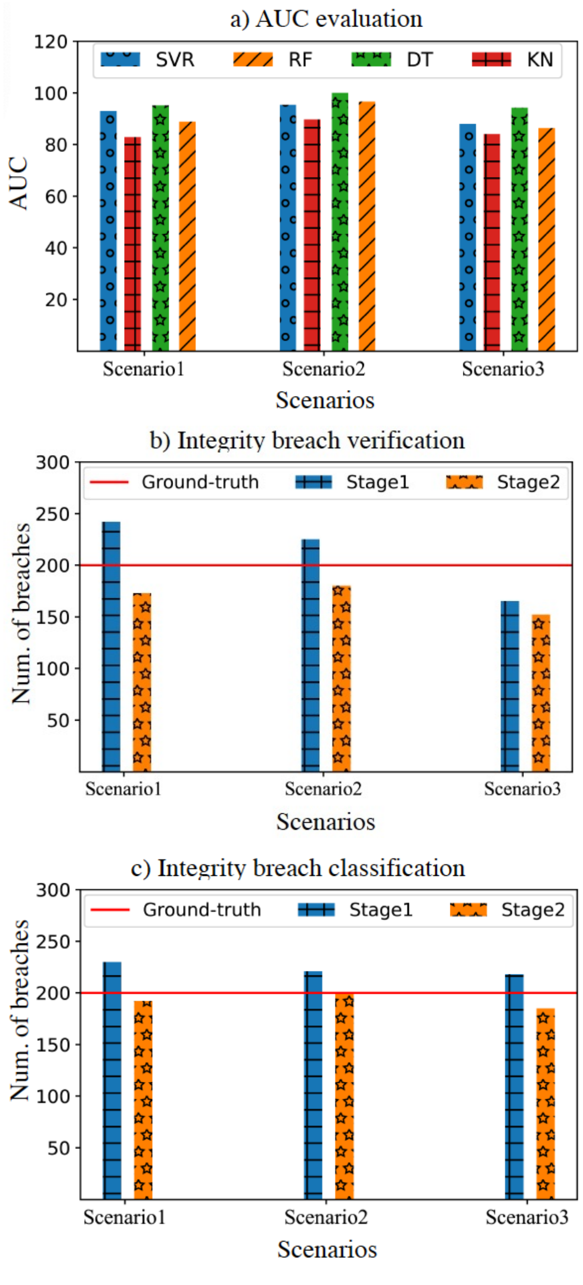


Figure 4.6: a) Stage 1 evaluation for integrity breach classification, b) evaluation of Stage 2 performance in correctly identifying integrity breaches from Stage 1, and c) evaluation of Stage 2 performance in classifying the integrity breaches from Stage 1.

are set to four, the accuracy becomes almost saturated (i.e., around 95% for $Scenario_2$). As a result, the model decision is made to identify suspected breaches based on the results of four probing samples that cover four different probing rates. Also, even though the performance profile of $Scenario_3$ is the closest to the scenario without a breach (as discussed above), our model successfully identifies it and the accuracy increases significantly when increasing both the number of probing rates and the length of the packet stream.

Evaluation of Stage 1 in Classifying Suspected Breaches. The second set of experiments (Figure 4.6.a) is to evaluate the accuracy of Stage 1 in classifying the suspected breaches. To that end, we use 20% of the data to train the signature-based *classification model*, while 5% for validation, and 75% for testing. Similar to the previous set of experiments, we implement four ML algorithms and run our experiments up to 100 epochs with an early stopping mechanism (i.e., early-stop-threshold $= 5$). Figure 4.6.a shows that the Decision Tree (DT) model achieves the best AUC compared to other ML models for all three scenarios as it reduces the variance by creating its predictions based on the training data. The DT model achieves the AUC value of 95.1% for $Scenario_1$, 100% for $Scenario_2$, and 94.2% for $Scenario_3$. Note that Stage 1 attains the highest accuracy for $Scenario_2$ as there is a distinct gap between the data points of this scenario compared to other scenarios.

Evaluation of Stage 2 in Verifying Selective Data. Figures 4.6.b and 4.6.c show the outcomes of Stage 2 in verifying the suspected breaches. The logs are anonymized by using iCAT [100] (an interactive and customizable anonymization tool as discussed in Section 4.3.3), while Nova logs are used to evaluate $Scenario_2$ (i.e., physical hosts reduction), and Neutron and OpenvSwitch logs are used to evaluate $Scenario_1$ and $Scenario_3$ (i.e., active and passive VM insertion). Figure 4.6.b shows the number of breaches identified at two stages (separately) compared to the ground truth. We can see that Stage 1 identifies a slightly larger number of breaches compared to the ground truth (i.e., 242 and 225 for $Scenario_1$ and $Scenario_2$, respectively), which denotes false positives. However, due

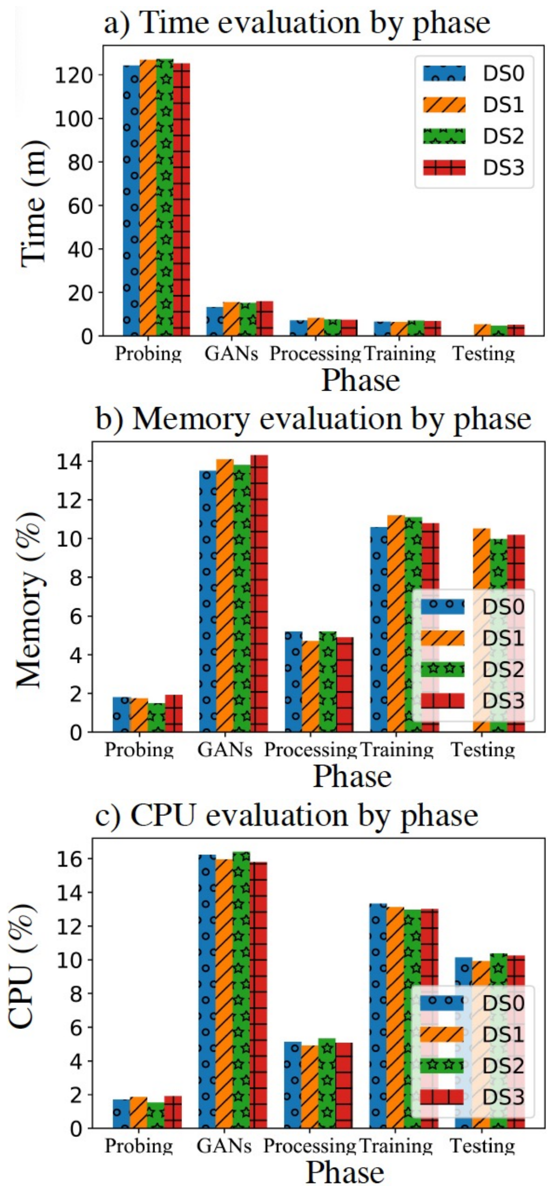


Figure 4.7: NFVSense resource consumption evaluation.

to Stage 2, we can successfully filter out those false positives from Stage 1. Thus, in $Scenario_1$ and $Scenario_2$, 173 and 183 (respectively) of the suspects are identified by Stage 2 to correspond to real integrity breaches. In $Scenario_3$, Stage 1 identifies a slightly lower number of breaches compared to the ground truth (i.e., 165 for $Scenario_3$), which denotes few missed integrity breaches. Stage 2 filters out the false positives and identifies 152 of those suspects corresponding to real ones. Figure 4.6.c shows the number of classified suspected breaches at two stages (separately) compared to the ground truth. In this context, even though some false positives are generated by Stage 1, Stage 2 filters the false positives and shows that there are only a few misses by Stage 1, which joins the accuracy obtained in the previous evaluation experiment. In summary, Stage 2 can successfully filter out false positives generated by Stage 1. Furthermore, the total number of integrity breaches reported by Stage 2 indicates that the identification and classification of suspected breaches by Stage 1 are fairly accurate concerning the ground truth (e.g., for $Scenario_2$, 183 out of 200 breaches are identified by Stage 1). Note that, the total number of false negative decisions is also significantly low (e.g., for $Scenario_2$, the number of false negatives is 17 out of 200 breaches), but NFVSense cannot verify this false negative decision using its Stage 2.

Table 4.3: Datasets partitioning for synthetic data validation.

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

Evaluation of Resource Consumption. This set of experiments (Figure 4.7) is to evaluate the time, CPU, and memory consumption by Stage 1. Figure 4.7.a shows the time consumption of Stage 1 by phase. Note that the scenario without a breach dataset ($DS0$) has no testing time as it is only used for training the suspected breach identification model, and this evaluation is based on the ML models that achieve the best accuracy (i.e., SVR for *identification model* and DT for *classification model*). This figure depicts that the most costly

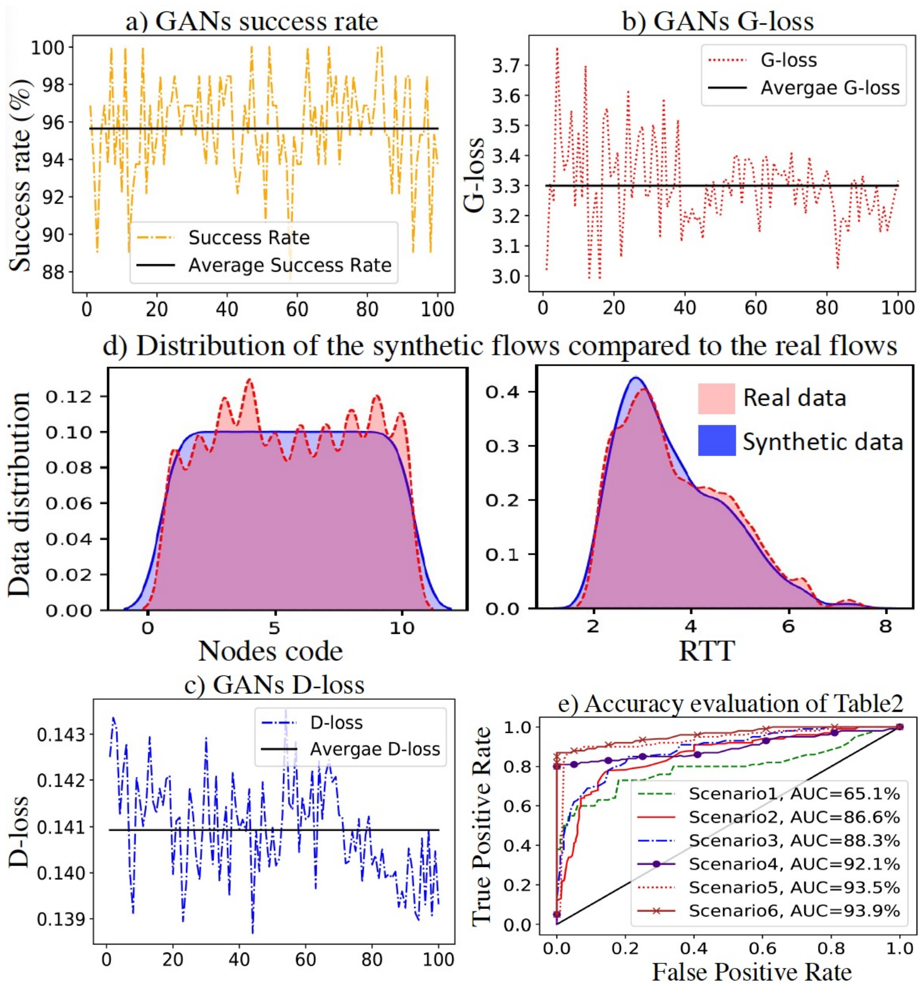


Figure 4.8: Synthetic data effectiveness evaluation.

operation by Stage 1 is to characterize network performance data using NPT, where we collect performance statistics about the NFV deployment to train the *identification model*. This observation indicates that using GAN-generated synthetic data may reduce the data collection time. On the other hand, Figures 4.7.b and 4.7.c show each dataset's memory and CPU consumption, respectively. Though the probing step consumes a higher time, it consumes the lowest CPU and memory resources. Also, synthesizing by GANs is the most expensive step in terms of CPU and memory consumption. However, the CPU consumption (17%) and the memory consumption (15%) for GANs are still not very high and might be affordable.

Synthetic Data Effectiveness. The final set of experiments (Figure 4.8) is to measure the effectiveness of our synthetically generated data (using GAN). To that end, we evaluate the success rate of synthetic data that is indistinguishable from the real one, and the losses of the discriminator and the generator models, and conduct a comparison between the real and synthetic data distributions. Figure 4.8.a depicts a summary of the effectiveness evaluation of GAN for 100 epochs, while on average, the success rate of the synthetic data is around 95.7%. On the other hand, the average generator and discriminator losses are only 3.3% and 0.14%, as shown in Figures 4.8.b and 4.8.c, respectively. Such high accuracy indicates a successful generation of indistinguishable synthetic data, while Figure 4.8.d also illustrates the similarity of the distribution of generated data (red-shaded areas) with the real one (blue-shaded areas). Additionally, in Figure 4.8.e, we evaluate the SVR model only on one integrity breach dataset, $DS2$ (i.e., physical host reduction scenario), to evaluate the performance of the synthetic data in the identification accuracy of the NFVSense using various partitioning setups between synthetic and real data (as shown in Table 4.3). The figure demonstrates that the synthetic data provides almost the same accuracy as the real one. As an example, despite having different ratios of the real and synthetic data, the two equal-sized datasets (i.e., $Setup_2$ and $Setup_3$ in Table 4.3) show almost the same identification accuracy. The identification accuracy also increases with the increased amount of training data and hence, augmenting the real data (i.e., 4K) with 20K of synthetic data ($Setup_4$ in Table 4.3) increases the accuracy significantly (92.1%), while this is 93.5% for $Setup_5$. Thus, these experimental results indicate that utilizing synthetic data where collecting large-scale real data is infeasible might be an alternative for the NFVSense users.

4.5 Discussions

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

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

Various Options for Stage 2 Verification. The design of NFVSense, particularly its Stage 2, is as such that a wide range of existing tools (e.g., VS [117]) can be leveraged for the verification step. In this paper, we mainly leveraged formal methods (e.g., NFVGuard [101]) and manual inspection as examples. Nonetheless, other verification tools can be leveraged.

Compatibility with Other NFV Platforms. NFVSense is designed based on the generic NFV architecture and deployment model [4], and all its modules are mostly platform-agnostic (except the learned models). Therefore, NFVSense is a generic solution, which can consequently be adapted to other NFV platforms (e.g., OSM [118] and OPNFV [119]), by learning platform-specific models.

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

4.6 Summary

Network Functions Virtualization (NFV) has emerged as an innovative networking architecture that aims to overcome the limitations of traditional networks. NFV exploits sophisticated virtualization technologies to implement VNFs as hardware-based middleboxes as software appliances. Many enterprises still consider the cloud as an untrustworthy domain to operate mission-critical applications (e.g., firewalls, IDS, etc.). They anticipate dishonest behaviors by cloud providers, including violating strategic services clauses such as data privacy and Service Level Agreement while obscuring these violations. Also, co-resident tenants, external attackers, or malicious insiders may compromise enterprises' network services to steal sensitive data or sabotage business operations. These risks are exacerbated in cloud-based NFV environments. In this work, we have proposed a two-stage mechanism to overcome the above challenges. In the first stage, we proposed a cloud provider-based data anonymization tool to overcome the challenges of limited access to audit data. In the second stage, we proposed a set of tenant-based solutions that utilize the side channels available as indirect effects of the attacks to detect NFV-based anomalies. In future work, we plan to investigate other side channels to audit the correctness of the deployed VNFs and the forwarding behavior of the SFCs. We will also merge our solutions with existing direct auditing-based methods to fill any gap between our BlackBox methods and whitebox existing solutions. Finally, we intend to compare our proposed solution's performance and effectiveness against existing direct auditing-based techniques.

Chapter 5

Continuous Verification of Virtual Network Functions Services

5.1 Introduction

By decoupling network functions from proprietary physical boxes, Network Functions Virtualization (NFV) [4] allows tenants to host their network services on top of existing clouds managed by third-party cloud providers [83, 84, 85]. For instance, Amazon AWS Cloud is reportedly used to deploy an entire cloud-native 5G network [83], and VMware Telco Cloud platform is also designed for similar purposes [85]. However, outsourcing network services to third-party clouds may also bring novel security challenges. Since tenants typically have limited visibility into the underlying cloud infrastructure, they cannot directly inspect the cloud-level deployment of Service Function Chains (SFCs) to ensure their deployment matches the specification. Therefore, cloud-level integrity breaches may silently arise and stay invisible to tenants [7]. For instance, an attacker can exploit a vulnerability or a misconfiguration in cloud-level resources (e.g., virtual switches) either to attack the SFC forwarding path (e.g., skipping a firewall inside the SFC [1]), or to attack the traffic (e.g., packet/flow injection, dropping, and reordering [11]).

Many existing solutions for verifying forwarding paths (e.g., ICING [48], OPT [47], and EPIC [120]) are not directly applicable to SFCs as they are incompatible with the inherent characteristics of SFCs (i.e., paths dynamicity and packets mutability) [5, 1]. Later works [1, 11, 51]) address this through adding cryptographic trailers to packets, which can guarantee the integrity and detect various attacks. However, such capabilities come at a cost, i.e., the added communication overhead is usually proportional to the flow size (e.g., three times increase of the packet size [47] and 1.69 times increase of the network traffic size [120]). Such an overhead may be prohibitive for applications with large flow sizes, e.g., music streaming, video conferencing, and virtual reality, which have become increasingly popular today.

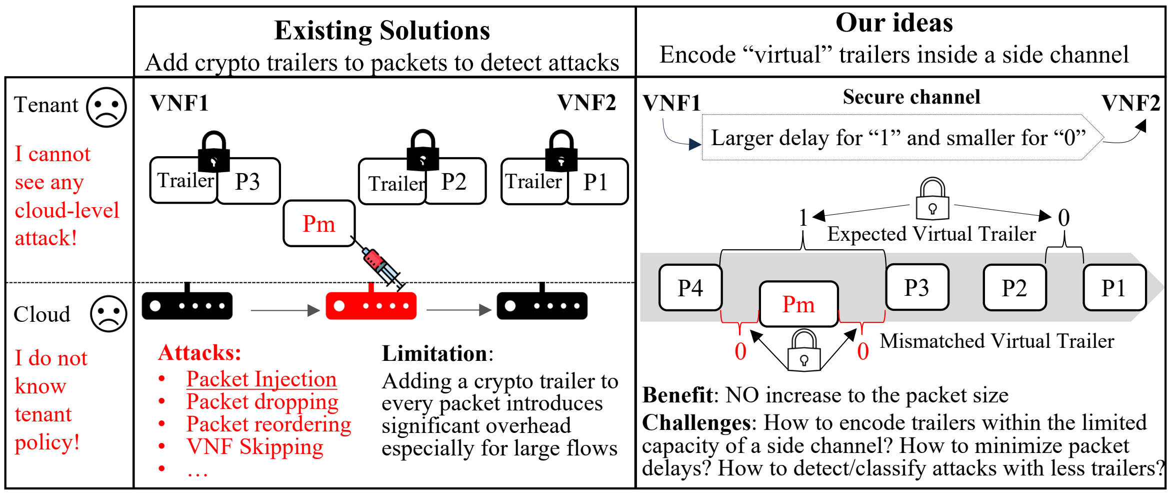

Existing Solutions
Add crypto trailers to packets to detect attacks
Our ideas
Encode "virtual" trailers inside a side channel
Tenant
I cannot see any cloud-level attack!
Cloud
I do not know tenant policy!
VNF1
VNF2
Trailer
P3
Trailer
P2
Trailer
P1
Pm
Attacks:
Packet Injection
Packet dropping
Packet reordering
VNF Skipping
…
Limitation:
Adding a crypto trailer to every packet introduces significant overhead especially for large flows
Secure channel
Larger delay for "1" and smaller for "0"
1
0
Expected Virtual Trailer
P4
Pm
P3
P2
P1
0
0
Mismatched Virtual Trailer
Benefit: NO increase to the packet size
Challenges: How to encode trailers within the limited capacity of a side channel? How to minimize packet delays? How to detect/classify attacks with less trailers?

Figure 5.1: Motivating example

Motivating Example. Figure 5.1 shows a motivating example to further illustrate the limitation of existing solutions (left), and our ideas (right). For simplicity, we consider a toy example of SFC consisting of two Virtual Network Functions (VNFs) connected through three cloud-level virtual switches, among which the middle one is compromised.

The Research Problem: The left side of Figure 5.1 illustrates a packet injection attack against the SFC. By exploiting either a vulnerability [121, 86] or misconfiguration [91] in the middle virtual switch, an attacker can inject a crafted packet (*Pm*) into the flow of normal packets (*P1*, *P2*, and *P3*) between *VNF1* and *VNF2*. Since the tenant has no direct access to cloud-level resources, it cannot easily uncover this attack. The protection provided by the cloud provider may also be limited since it is not necessarily aware of all tenants' SFCs and forwarding policies. Therefore, such attacks may fall into the gap and go undetected.

Existing solutions: The left side of Figure 5.1 also shows how existing cryptographic solutions (e.g., [1, 11, 51]) can detect the aforementioned attacks. Specifically, such solutions would modify *VNF1* to append a verifiable cryptographic trailer (including a Message Authentication Code (MAC) value computed over the packet and other trailer fields) to every packet before it leaves *VNF1*. *VNF2* is also modified to verify the trailer when the packet arrives. The malicious packet *Pm* can be reliably detected by *VNF2* since adversaries cannot forge such a trailer. However, as those solutions are adding a trailer to every single packet, they imply an overhead that is proportional to the flow size.

Our ideas: As shown on the right side of Figure 5.1, we "virtualize" the physical trailers as side channel watermarks, e.g., encoding a "virtual" trailer of '10' by slightly delaying packets *P1* and *P4* before they leave *VNF1*, such that the inter-packet delay between *P1* and *P2* becomes slightly smaller than usual, representing a '0' bit, while the delay between *P3* and *P4* becomes slightly larger than usual, representing a '1' bit. The injection of a malicious packet *Pm* can be detected as it will partially destroy our virtual trailer (there will now be two '0' bits after the injection, as shown in the figure). The exact way the trailer is destroyed would also provide us additional information to not only detect the attack, but also to classify its type (detailed in Section 5.4).

Specifically, this paper presents ChainPatrol, a solution for lightweight and verifiable detection and classification of various attacks against SFC forwarding paths and traffic. At the source VNF, ChainPatrol encodes *virtual trailers* using side-channel information, namely, inter-packet delays (IPDs). At the destination VNF, ChainPatrol extracts the virtual trailers from observed IPDs to detect and classify various attacks. This novel approach provides the best of both worlds. First, ChainPatrol offers a much lighter-weight solution than existing works based on physical trailers [1, 11, 51], since no extra bit needs to be added to packets (virtual trailers are encoded as side channel watermarks). Second, since virtual trailers contain similar information as in their physical counterparts, ChainPatrol can still guarantee the integrity of SFCs when the trailers are intact, and provide useful information for detecting and classifying attacks when the trailers are compromised (note these cannot be achieved by directly applying existing watermarking techniques [53, 54, 56], because the watermarks lack the semantics of a cryptographic trailer, e.g., sequence numbers of packets or flows and MAC values). Finally, unlike many existing works (which either require direct accesses to the underlying cloud infrastructure [31] or require modifications to the VNFs [30]), ChainPatrol provides a tenant-level solution that can be transparently integrated with existing SFCs, since IPDs are directly observable and mutable at tenant level and outside the VNFs. In summary, our main contributions are as follows.

- We propose the novel concept of *virtual trailer*, which inherits the advantages of both cryptographic trailer-based solutions (i.e., verifiable attack detection) and side-channel watermarking (i.e., lightweight). To realize this, we address several key challenges such as encoding virtual trailers within the limited capacity of a side channel, minimizing packet delay while computing virtual trailers, and detecting/classifying attacks with less trailers. We believe this concept may potentially find other applications in a broader context.
- We apply the concept of virtual trailer to design a tenant-based solution, ChainPatrol, for lightweight attack detection and classification in SFCs hosted on third-party clouds. First, ChainPatrol performs fast attack detection by identifying destroyed virtual trailers. Second, it performs in-depth attack classification through partial reconstruction of destroyed virtual trailers to match the expected ones. Finally, it verifies

classification results through sharing a limited amount of information between the source and destination.

- We implement and deploy ChainPatrol based on Amazon EC2 and perform extensive experiments using both public datasets and an in-house 5G testbed. Our experimental results demonstrate the effectiveness and efficiency of ChainPatrol (e.g., close to zero communication overhead and 0.68 ms end-to-end delay). Our comparison of ChainPatrol to an existing physical trailer-based approach under real-world applications shows a significant reduction of communication overhead (up to 45%). ChainPatrol is demonstrated through its integration into Amazon ECS.

5.2 Background

5.2.1 SFC Forwarding Path Verification

Existing cryptographic solutions for forwarding path verification [1, 11] adapt traditional forwarding path verification protocols [48, 47, 120] to the NFV setting in order to meet its unique characteristics: i) path dynamicity and unpredictability, ii) packet modification by VNFs, and iii) SFC migration and autoscaling. Specifically, those solutions intercept each egress packet at a source VNF to append a custom trailer to the packet, which consists of additional data fields along with a MAC computed over the entire packet content (including the trailer) using a secret key. A valid MAC extracted at the destination VNF (under the same key) would attest to the integrity of the packet content. This allows the traffic between VNFs to be verified for correctness.

Example 5.2.1 *As shown in Figure 5.2, a packet tagger at the source VNF appends a packet trailer to each passing packet (top), and a packet verifier at the destination VNF verifies the packet and its trailer for integrity (bottom). First,* `SeqNum` *is a sequential number representing the ordering of packets between each pair of VNFs for a given flow.*

Together with `FlowID`*, it is used to detect packet reordering and replay attacks within a given flow. Second,* `FlowID` *is uniquely mapped to the classic 5-tuple ⟨source IP address, destination IP address, source port, destination port, protocol⟩, and is used to detect flow-based attacks (e.g., flow injection and flow dropping). Third,* `SrcID` *and* `DstID` *refer to the source and destination VNFs, respectively, and are used to ensure compliance with forwarding policy between the two VNFs. Finally,* `MAC` *is computed over both the packet content and above trailer fields, and is used to authenticate the integrity of the packet and its trailer by the destination VNF.*

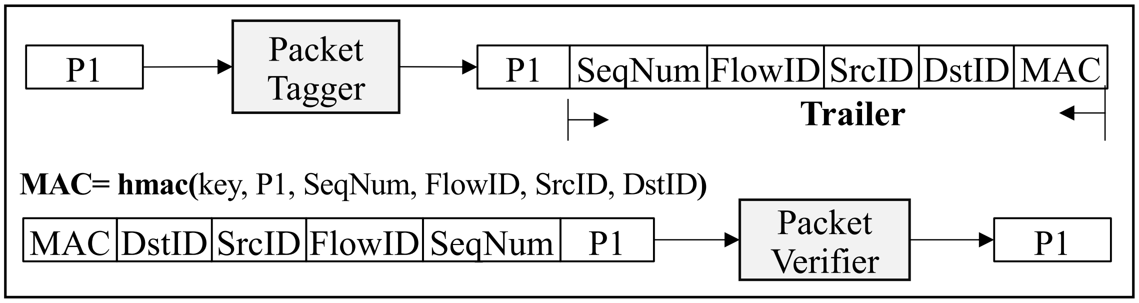


Figure 5.2: Example of a packet with cryptographic trailer [1]

5.2.2 Blind Watermarking

In this work, we adopt a *blind* watermarking approach (i.e., the packet verifier does not require knowledge about the original IPDs or watermarks). This choice was made for the following two reasons: i) As we leverage watermarks to transmit virtual trailers without actually sending any bit, a non-blind approach, which must send information about the original IPDs and watermarks between the two VNFs, would defy our purpose; ii) Although a non-blind watermarking approach is usually more resistant to network jitters (since the packet verifier knows the original IPDs), a blind approach is sufficient for our purpose, because the network connection between two VNFs in a data center is typically more stable (in contrast to the Internet for which most existing watermarking approaches are designed) [122].

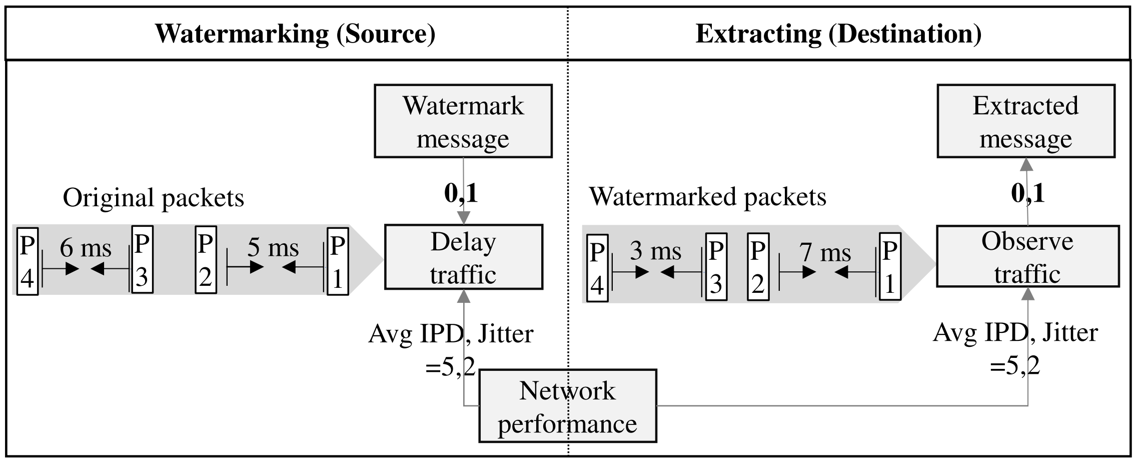


Figure 5.3: Example of watermarking and extracting two bits

Example 5.2.2 *Figure 5.3 shows how a two-bit watermark message* $\langle 1, 0 \rangle$ *is encoded into the IPDs between four packets at the source (left), and decoded at the destination (right). First, the average IPD (*5 *ms) and network jitter (*2 *ms) are continuously measured and used for reference on both sides. Second, the source encodes a bit* 1 *by increasing the original IPD between packets* P_1 *and* P_2 *corresponding to the summation of the average IPD and jitter (i.e., from* 5 *ms to* 7 *ms). Similarly, it encodes a bit* 0 *by decreasing the IPD between* P_3 *and* P_4 *corresponding to the difference between the average IPD and jitter (i.e., from* 6 *ms to* 3 *ms). Third, by comparing the observed IPD with the average IPD and jitter, the destination decodes a bit* 1 *between* P_1 *and* P_2 *as* $7 \geq (5 + 2)$*, and a bit* 0 *between* P_3 *and* P_4 *as* $3 \leq (5 - 2)$*. Note a jitter no greater than* 2 *would not change this result.*

5.2.3 Threat Model

We consider a similar threat model as in recent works on crypto-based forwarding path verification [1, 11]. We also make several assumptions related to our watermarking scheme.

In-Scope threats. Our in-scope threats include integrity breaches of the SFC forwarding paths or traffic caused by either: i) a malicious attacker compromising an underlying cloud-level forwarding device [86]; ii) misconfigurations (intentionally or mistakenly) introduced by a cloud provider [91].

Specifically, we assume the network links between VNFs and the virtual switches used to steer traffic between such VNFs are both subject to the following attacks. First, a compromised cloud-level device may be used to skip VNFs or append malicious VNFs to the forwarding path, or to cause other unexpected forwarding decisions such as redirecting traffic intended for one VNF to another [11]. Second, a compromised cloud-level device may also be used to disrupt SFC traffic, such as injecting fake packets and dropping, modifying, reordering, or replaying packets. We assume the adversary may attempt to evade the detection through the so-called coward attack [47] (i.e., attacking selected flows not subject to detection) or attacks on the watermarking scheme used to encode virtual trailers (by deliberately altering the IPDs). A more detailed list of attacks covered in our work is given later in Table 5.2.

Out-of-Scope threats. By taking a tenant-based viewpoint, we assume no direct access to cloud-level resources or data, and therefore we must trust all the components to which the tenant has direct access. This includes the VNFs and the gateway to access the SFC, and we also consider the cloud provider to be cheap-and-lazy but not malicious [91]. Attacks that can compromise those components or our solution itself (including the secret key used to compute the MAC) are out of scope for our work (which can be addressed through hardware-based solutions [51, 1]).We focus on verifying the integrity of SFCs, and thus denial of service attacks (e.g., dropping all packets) and attacks on confidentiality or privacy of the traffic are out of scope. Moreover, since we leverage a fragile watermarking scheme, which relies on modified watermarks to detect attacks [123], watermark invisibility attacks [53]) are out of scope for our work. Finally, as demonstrated in Section 5.6, our solution is more beneficial for flows of relatively large sizes (for small flows, the overhead of physical trailers may be acceptable).

5.3 Virtual Trailer

This section defines the *virtual trailer* concept and details how to encode and decode virtual trailers based on IPDs.

5.3.1 Definitions

Our goal is to design virtual trailers to contain similar information as in their physical counterparts, such that they can support attack detection and classification (detailed in Section 5.4). However, there are several challenges. First, unlike a physical trailer which can be defined for (and added to) each packet, a virtual trailer can only be encoded in the IPDs of multiple packets. Therefore, we define a virtual trailer for each equal-sized group of consecutive packets within the same network flow, namely, a *block*. Second, due to the limited capacity of this side channel, we need to simplify the design of virtual trailers to only retain a minimal number of trailer fields, i.e., the identifiers of blocks and flows, while implicitly representing the source and destination VNFs inside the flow identifiers. Specifically, each flow identifier is now uniquely associated with a 7-tuple ⟨`source IP`, `destination IP`, `source port`, `destination port`, `protocol`, `source VNF`, `destination VNF` ⟩. Third, since a MAC value typically contains a much larger number of bits than what can be encoded inside a single block, we redefine the MAC to be computed over an equal-sized group of consecutive blocks inside the same flow, namely, a *SuperBlock*. Finally, to enable efficient attack classification and verification, the MAC is computed based on a Merkle hash tree [124] defined over the SuperBlock. The following first formally defines a virtual trailer for each block.

Definition 5.3.1 (Virtual Trailer) *Given block B_i ($1 \leq i \leq S_F$) inside a SuperBlock SB (with totally S_F blocks) in a flow F, the virtual trailer of B_i is a 3-tuple $VT_i = \langle BlockNum_i, FlowID, MAC_i \rangle$. First, $BlockNum_i \in \mathbf{N}$ is an integer value uniquely and*

Table 5.1: Examples of Virtual Trailers (VT)

Flow	SB	B	VT		
			BlockNum	FlowID	MAC
¡IP1,IP2,8,80,6,VNF1,VNF2¿	1	1	0001	0001	0100
		2	0010	0001	1100
	2	3	0011	0001	1100
		4	0100	0001	1100
¡IP1,IP2,4,80,6,VNF1,VNF2¿	1	1	0001	0010	1010
		2	0010	0010	0101

sequentially assigned to each block in the given flow F. Second, FlowID $\in \mathbf{N}$ is an integer value uniquely and sequentially assigned to each flow. Third, MAC$_i$ $\in \mathbf{N}$ is the $(\frac{i}{S_F})^{th}$ fraction of the Merkle tree [124] HMAC value computed over all the blocks in SB concatenated with their corresponding BlockNum and FlowID fields, i.e., B_i||BlockNum$_i$||FlowID $(1 \leq i \leq S_F)$.

Example 5.3.1 *Table 5.1 shows an example with two flows, their SuperBlocks and blocks, and the corresponding virtual trailers (last three columns). The* `BlockNum` *is sequentially assigned to consecutive blocks inside the same flow, starting from a random number. Similarly, the* `FlowID` *is sequentially assigned to consecutive flows. For the first flow, the Merkle tree HMAC value of* 01001100 *is divided into two equal-sized bit strings (*0100 *and* 1100*) each of which forms the last field of the virtual trailer (similarly for the second flow). Figure 5.4 also depicts a virtual trailer (bottom right) and how the MAC value is computed (left) (the rest of the figure will be explained later in Example 5.3.2).*

5.3.2 Virtual Trailer Encoding and Decoding

We detail how we encode and decode virtual trailers using the IPDs between the source and destination VNFs.

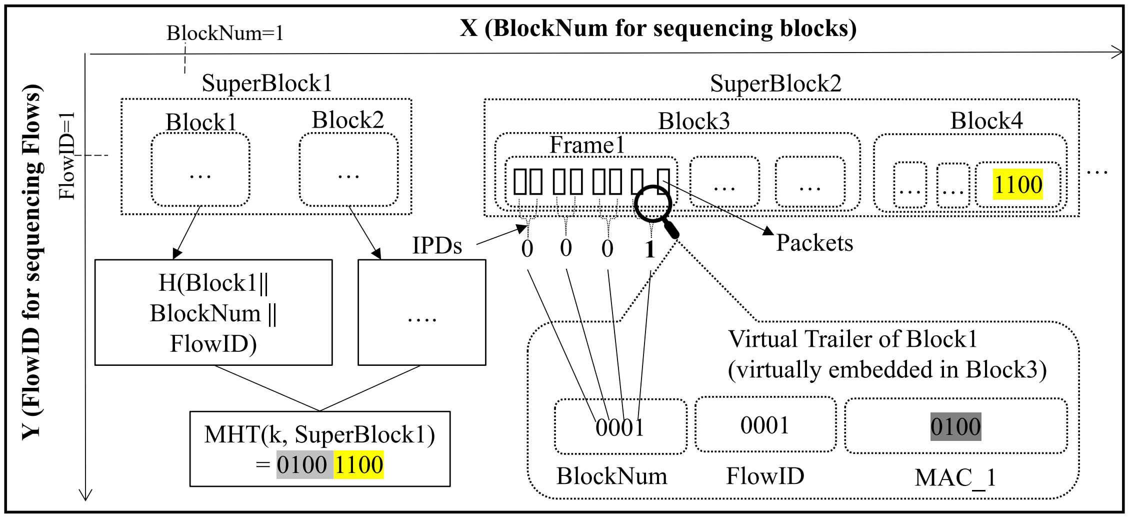

X (BlockNum for sequencing blocks)
BlockNum=1
Y (FlowID for sequencing Flows)
FlowID=1
SuperBlock1
Block1
Block2
SuperBlock2
Block3
Block4
Frame1
1100
Packets
IPDs
0 0 0 1
H(Block1‖ BlockNum ‖ FlowID)
….
MHT(k, SuperBlock1) = 0100 1100
Virtual Trailer of Block1 (virtually embedded in Block3)
0001
0001
0100
BlockNum
FlowID
MAC_1

Figure 5.4: Example of a virtual trailer and its encoding

Virtual Trailer Encoding

ChainPatrol intercepts packets at the egress of source VNF and determines their corresponding flows, SuperBlocks, and blocks (see Section 5.3.1). It then generates a virtual trailer for each block, and encodes the virtual trailer by modifying the IPDs (i.e., delaying one packet between every pair of packets, as explained in Section 5.2.2). Specifically, as each virtual trailer contains three fields (i.e., `BlockNum`, `FlowID`, and `MAC`), each block is divided into a series of three *frames*, and then the IPDs in each frame are modified to encode a corresponding trailer field.

However, a key challenge lies in the encoding of the MAC trailer field. Specifically, since the MAC field of a virtual trailer is defined over a SuperBlock (as illustrated on the left side of Figure 5.4), it cannot be computed before the entire SuperBlock is received. Therefore, any received blocks of this SuperBlock must be delayed, since we do not yet know how to modify their IPDs. Such delays must be maintained until the last block arrives, after which we can then compute the virtual trailer, encode it by modifying the IPDs of all the buffered blocks, and finally forward all the blocks to the destination VNF. However, doing so would certainly cause a prohibitive delay (proportional to the size of the

SuperBlock). To address this, our key idea is to shift the encoding of each virtual trailer to the next SuperBlock. This would allow us to compute the virtual trailer (which tells us how to modify the IPDs) ahead of the arrival of any block of the next SuperBlock, such that we will know how to modify the IPD as soon as each such block arrives, and can forward the block with minimum delay (as detailed in our experimental results in Section 5.6). This is further illustrated in the following example.

Example 5.3.2 *Figure 5.4 shows how the virtual trailer of a block (*`Block1`*) in the first SuperBlock (shown on the left) is encoded using a block (*`Block3`*) in the next SuperBlock (shown on the right). First,* `Block3` *is divided into three frames, each of which corresponds to a virtual trailer field. Second, the IPD between each pair of packets inside a frame are then modified (by delaying one of those packets) to encode each bit of the field. Similarly, the virtual trailer of* `Block2` *(i.e, the second block of* `SuperBlock1`*) is encoded in* `Block4`*. Note that, by the time* `Block3` *arrives, this virtual trailer of* `Block1` *would have already been computed. Therefore, we know how to modify the IPDs in* `Block3` *as soon as each pair of packets arrives, which can then be immediately forwarded.*

Another challenge is to ensure the correct decoding of virtual trailers despite potential network jitters. For this purpose, ChainPatrol leverages an existing watermarking scheme [53] which is known to enable reliable extraction of watermarks at the destination side despite network jitters. The main idea is to introduce an additional delay that is proportional to the expected level of jitters to cancel their impact on the encoded watermarks. Specifically, let $m_1 \cdots m_w$ be the w-bit representation of a virtual trailer field to be encoded. Denote the IPD between two packets arriving at time t_i and t_{i+1} as *IPD*$_i = (t_{i+1} - t_i)$. To encode bit m_i ($1 \leq i \leq w$) using *IPD*$_i$, the new IPD, denoted by *nIPD*$_i$, is computed as: *nIPD*$_i =$ *IPD*$_{AVG} + a \times M_i^e$, where IPD_{AVG} is the average IPD, a is the *watermarking amplitude* (computed using the Signal-to-Noise formula [53]), and $M_i^e = 1$ if $m_i = 1$, or $M_i^e = -1$ if $m_i = 0$ (the effectiveness of this method will be further evaluated in Section 5.6).

Virtual Trailer Decoding

ChainPatrol passively monitors packets at the destination VNF and determines their corresponding flows, SuperBlocks, blocks, and frames. It then decodes the virtual trailers following a reversed process based on the observed IPDs. More specifically, let $rIPD_i$ $(1 \leq i \leq w)$ be the observed IPD between the i^{th} and $(i+1)^{th}$ packets, where w is the size of a virtual trailer field, and let IPD_{AVG} and a be the average IPD and the watermarking amplitude, respectively. Denote $M_i^d = (rIPD_i - IPD_{AVG})/a$. The w-bit binary representation of the virtual trailer field can be computed as: $m_i = 1$ if $M_i^d = 1$, or $m_i = 0$ if $M_i^d = -1$.

Attacks such as dropping, injection, or reordering of packets may shift the frames and blocks among the observed packets. Section 5.4 will detail how we address this issue and leverage it to classify attacks. Another challenge is related to the last SuperBlock. First, the virtual trailers of this last SuperBlock itself cannot be encoded using the IPDs of next SuperBlock (which does not exist). Second, when we divide a given flow into equal-sized blocks and SuperBlocks, the last SuperBlock may be incomplete, i.e., there are not enough packets remaining to compose a complete SuperBlock, which would prevent encoding the virtual trailers of the previous SuperBlock using IPDs. ChainPatrol addresses the first challenge by directly appending a physical version of the virtual trailers to the flow. Since ChainPatrol is designed for large flows (as stated in Section 5.2.3) with a significant number of SuperBlocks, the overhead of one trailer will be negligible in contrast to the flow size. To address the second challenge, ChainPatrol performs one of the following two options that introduce less overhead, i.e., i) adding dummy packets to the last SuperBlock such that it can have enough IPDs for encoding the virtual trailers of the previous SuperBlock, or ii) appending the physical trailers.

5.4 Attack Detection and Classification

5.4.1 Attack Detection

Similar to existing works using physical trailers [1, 11], ChainPatrol detects SFC attacks by matching decoded virtual trailers with corresponding blocks. However, the unique design of virtual trailers (detailed in Section 5.3) leads to two differences as follows. First, since the virtual trailers of one SuperBlocks are always encoded in the IPDs of the next SuperBlock, ChainPatrol iteratively performs attack detection inside a moving window that slides over the next two consecutive SuperBlocks of the current flow in each iteration. Second, recall that the HMAC of a SuperBlock is computed based on a Merkel hash tree defined over all the blocks, and each virtual trailer only includes part of this HMAC value (as illustrated in Figure 5.4). Therefore, ChainPatrol first decodes the virtual trailers encoded in the second SuperBlock inside the moving window and recomputes the Merkle tree HMAC based on the decoded `BlockNum` and `FlowID` values and all packets of the first SuperBlock. It then compares this re-computed HMAC with the decoded HMAC value (obtained by concatenating the `MAC` fields of all the virtual trailers decoded from the second SuperBlock).

One challenge is that, since an attack may cause the frames, blocks, or SuperBlocks to shift, ChainPatrol needs to identify the beginning of the next SuperBlock once an attack is detected. Specifically, if all the virtual trailer fields match the first SuperBlock in the window, ChainPatrol marks the first SuperBlock as "recovered", and the window slides forward by one complete SuperBlock. When an attack is detected, ChainPatrol marks the current pair of SuperBlocks as "attacked" (which will be further inspected for attack classification), and the window slides forward by only a pair of packets to identify the beginning of the next SuperBlock. Once a new SuperBlock is identified, ChainPatrol resumes the normal attack detection as described above.

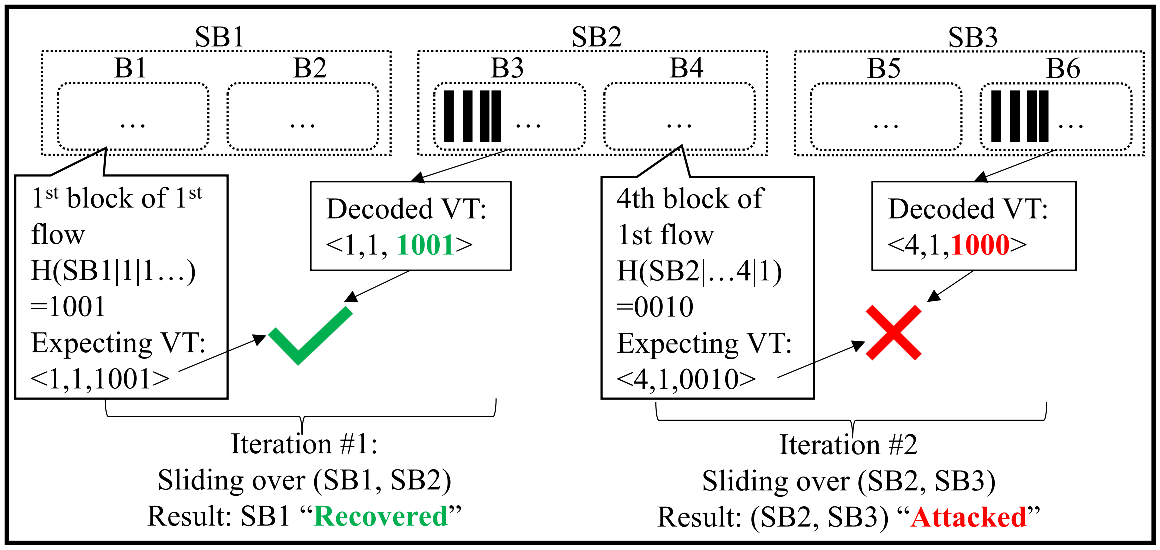

SB1
B1
B2
SB2
B3
B4
SB3
B5
B6
1st block of 1st flow H(SB1|1|1...) =1001 Expecting VT: <1,1,1001>
Decoded VT: <1,1, 1001>
4th block of 1st flow H(SB2|...4|1) =0010 Expecting VT: <4,1,0010>
Decoded VT: <4,1,1000>
Iteration #1: Sliding over (SB1, SB2) Result: SB1 "Recovered"
Iteration #2 Sliding over (SB2, SB3) Result: (SB2, SB3) "Attacked"

Figure 5.5: Example of attack detection

Example 5.4.1 *Figure 5.5 (top) shows the first three SuperBlocks of the first flow, and (bottom) an example of attack detection.*

- *In the first iteration, the window slides over the first two SuperBlocks,* `SB1` *and* `SB2`*. For the first block of* `SB1`*, the expected* `BlockNum` $= 1$*) and* `FlowID` $= 1$*), as this is assumed to be the first block of the first flow. As the figure shows, the first virtual trailer decoded from* `SB2` *also contains the same* `BlockNum` *and* `FlowID` *values.*
- *The expected* `MAC` *needs to be computed over the content of* $SB1$ *concatenated with the decoded* `BlockNum` *and* `FlowID` *fields of both blocks inside* `SB1`*. As the figure shows, the first half of the MAC value decoded from* `SB2` *matches the expected value* (1001)*. Assuming the second half also matches,* `SB1` *can be marked as "recovered" as no attack is detected.*
- *In second iteration, the window slides to* `SB2` *and* `SB3`*. Similarly, the expected* `BlockNum` *and* `FlowID` *match the ones decoded from* `SB3`*. However, assuming the first half of the expected MAC value* (0010) *does not match the decoded one* (1000)*,*

an attack is thus detected. At this point, it is unclear whether the attack happened to `SB2` *or the virtual trailers (i.e., IPDs) in* `SB3`*, so both SuperBlocks are marked as "attacked" for classification.*

- *Since the attack may have caused packets to be injected, dropped, or reordered, we cannot take the beginning of the next SuperBlock for granted. Instead, the window can only slide forward by one pair of packets at a time, until we identify the next intact SuperBlock.*

5.4.2 Attack Classification

Upon the detection of an attack, ChainPatrol classifies it with one of the known attack types listed in Table 5.2. In addition to the well-known packet-level attacks reported in the literature, ChainPatrol also considers block-level attacks that involve the manipulation of a whole packet block, flow-level attacks involving all packet blocks in a flow, and SFC-level attacks targeting the forwarding paths (instead of packets). To classify a SuperBlock marked as attacked during the detection stage (Section 5.4.1), ChainPatrol attempts to reconstruct the expected virtual trailers by applying each of the classification rules listed in the last column of Table 5.2, and it classifies with an attack type if the reconstruction is successful under the corresponding rule.

One challenge is that, depending on the level of the classification rule (first column of Table 5.2), this reconstruction may involve two SuperBlocks, the remainder of the flow, or even other flows (e.g., flow or SFC-level attacks). Therefore, attack classification naturally requires more effort than attack detection. This explains why ChainPatrol adopts a layered approach to separate the (faster) detection from (slower) classification such that it can provide faster detection (as shown through experiments in Section 5.6).

Another challenge is that the missing information (e.g., in case of dropping or modification attacks) and lower granularity of virtual trailers (i.e., per block instead of per packet)

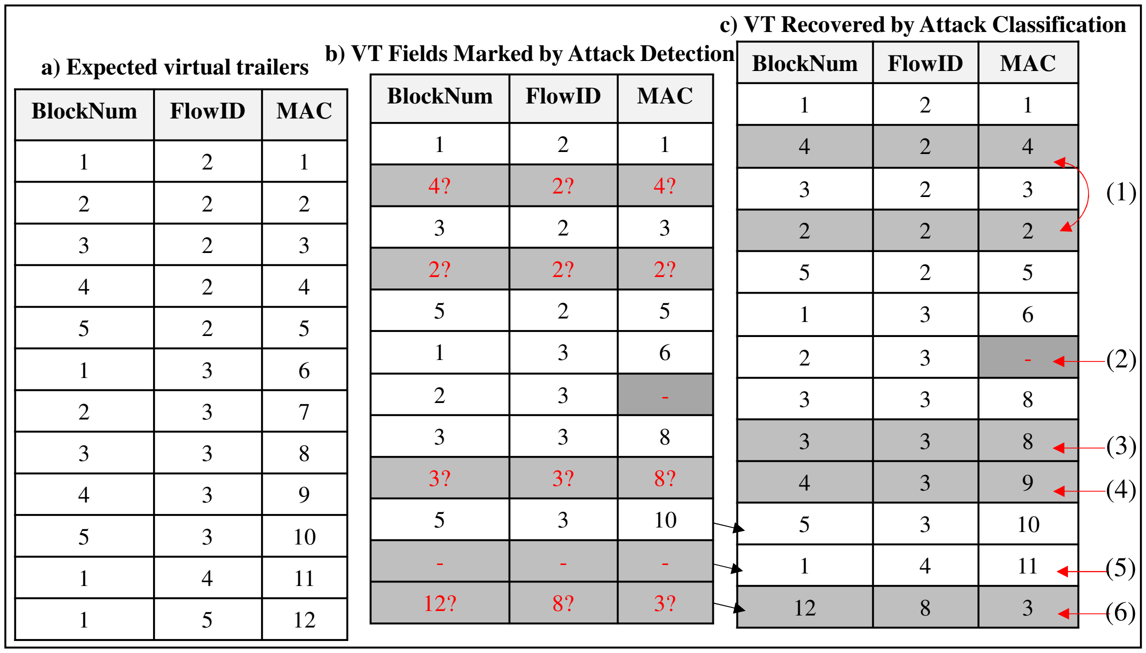
Table 5.2: Attack types covered by ChainPatrol

Level	**Attack type**	**Classification rules based on virtual trailers**
Packet	Injection	Unrecovered VT & larger block size
	Reordering	Partially recovered VT & expected block size
	Replay	Unrecovered VT & larger block size
	Modification	Unrecovered VT with mismatched MAC
	Dropping	Partially recovered VT & smaller block size
Block	Injection	Unrecovered VT & expected block size
	Reordering	Out of order `BlockNum`
	Replay	Recovered VT with repeated `BlockNum`
	Dropping	Missing VT with specific `BlockNum`
Flow	Injection	Unrecovered `FlowID`
	reordering	Out of order `FlowID`
	replay	Repeated VT for all blocks
	dropping	Missing `FlowID`
SFC	VNF Inj.	Unrecovered `FlowID`
	Reordering	Out-of-order `FlowID`
	VNF-Loop	Repeated `FlowID`
	Skipping	Missing `FlowID`

together may prevent a full reconstruction of virtual trailers, and hence the attack classification result can no longer be guaranteed like with detection. To address this, we leverage our design choice of computing the MAC based on a Merkle hash tree [124] defined over a superblock (detailed in Section 5.3) to enable efficient source-assisted verification of the classification result in such cases. Specifically, the well-known property of a Merkle hash tree [124] allows us to verify the MAC (and hence the correctness of classification) by requesting selected tree nodes from the source, e.g., a single node (common ancestor) is sufficient for $log(N)$ consecutive blocks (in contrast, N nodes would be requested if the MAC were computed directly over the SuperBlock).

Example 5.4.2 *Figure 5.6 shows six examples of classification.*

- *Attack (#1) is classified as reordering of the two shaded blocks since their* `BlockNum`



a) Expected virtual trailers

BlockNum	FlowID	MAC
1	2	1
2	2	2
3	2	3
4	2	4
5	2	5
1	3	6
2	3	7
3	3	8
4	3	9
5	3	10
1	4	11
1	5	12

b) VT Fields Marked by Attack Detection

BlockNum	FlowID	MAC
1	2	1
4?	2?	4?
3	2	3
2?	2?	2?
5	2	5
1	3	6
2	3	-
3	3	8
3?	3?	8?
5	3	10
-	-	-
12?	8?	3?

c) VT Recovered by Attack Classification

BlockNum	FlowID	MAC
1	2	1
4	2	4
3	2	3
2	2	2
5	2	5
1	3	6
2	3	-
3	3	8
3	3	8
4	3	9
5	3	10
1	4	11
12	8	3

Figure 5.6: Examples of attack classification (a question mark (?) means mismatching with expected value; a dash (-) means no received packet; grey rows are involved in detection or classification; numbers (1-6) indicate attacks)

fields are out-of-order (illustrated in Figure 5.6.c). Specifically, attack classification attempts to reconstruct the expected virtual trailers (i.e., first four rows in Figure 5.6.a) by applying each classification rule (last column of Table 5.2) to the attacked virtual trailers in Figure 5.6.b. The rule for block reordering leads to a successful reconstruction, and hence the attack is classified as such.

- *Attack (#2) is classified as packet-dropping, since attack classification can partially reconstruct the virtual trailer for the seventh row (pointed to by #2, where the* `BlockNum` *and* `FlowID` *fields are intact) and beyond, while the block size is smaller than expected (the* `MAC` *field is missing), which matches the classification rule for packet dropping in Table 5.2. This verification result can be further verified by requesting the missing information (Merkel tree node).*
- *Similarly, Attacks (#3) and (#4) are classified as packet block replay and drop, respectively.*
- *Attack (#5) is a flow drop attack, as the expected* `FlowID = 4` *(end of Figure 5.6.a)*

is missing in Figure 5.6.c, which matches the classification rule for flow dropping.

- *Attack (#6) is a VNF skipping attack since a packet block received at the destination VNF (the last row in Figure 5.6.b) has a missing FlowID (which does not appear in Figure 5.6.a), which matches the classification rule for VNF skipping.*

5.5 Implementation

5.5.1 Overview

ChainPatrol is composed of two *agents* per pair of communicating VNFs, plus a central *orchestrator*. Figure 5.7 illustrates the architecture and main components of ChainPatrol. The agents perform watermark encoding/decoding, attack detection, and local attack classification. The orchestrator is in charge of managing the agents and performing global attack classification. ChainPatrol is implemented in C++ programming language, with approximately 1300 lines of code at the agent, and 200 lines at the orchestrator. The following details the implementation.

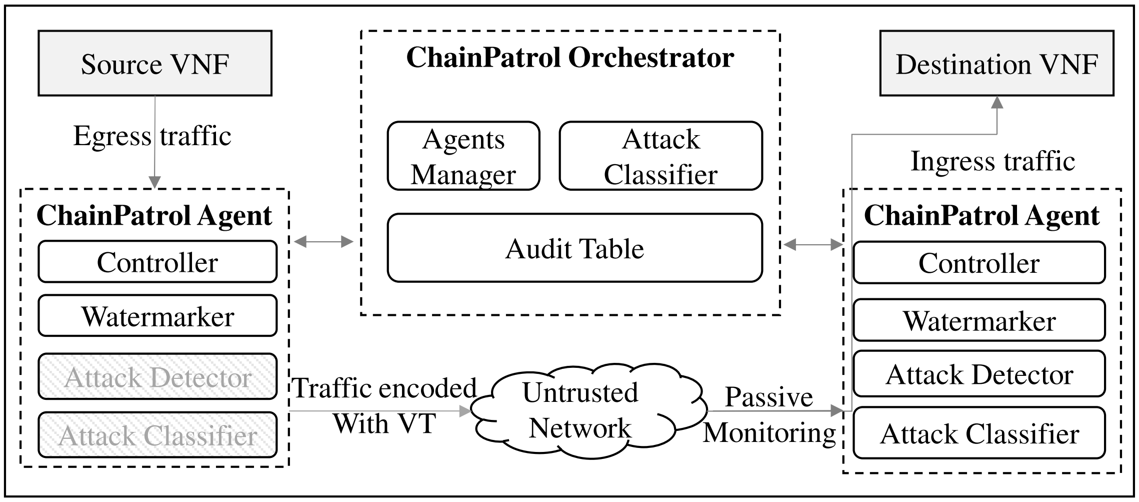


Figure 5.7: ChainPatrol Architecture

ChainPatrol agent. Each agent includes four components: i) The *controller* performs the following functionalities. First, it receives ChainPatrol parameters from the orchestrator at initialization, and it then continuously monitors the network performance (e.g., IPD and network jitter) to compute the watermark amplitude and X-shift values. Second, at the source VNF, it intercepts the egress traffic, generates virtual trailers (the Merkel hash trees are stored inside shared storage in case the destination-side agent may need selected nodes, as discussed in Section 5.4.2), triggers the watermarker, and forwards the traffic. Third, at the destination VNF, it passively monitors the ingress traffic to measure the IPDs, triggers the watermarker, attack detector, and attack classifier, and finally reports the detection and classification results to the orchestrator; ii) The source-side *watermarker* encodes the virtual trailers generated by the controller, and the destination-side watermarker decodes the virtual trailers from the ingress traffic. iii) Once triggered by the controller, the attack detector marks SuperBlocks as either recovered or attacked. iv) Finally, once triggered, the *attack classifier* classifies the detected attacks using classification rules that only involve local traffic.

ChainPatrol orchestrator. The orchestrator includes three components: i) The *agents manager* is responsible for instantiating the agents and communicating with them; ii) The *attack classifier* is responsible for global attack classification that cannot be performed locally at each agent (e.g., flow or SFC-level attacks); iii) The *Audit Table* is used to store the detection and classification results both from the agents and from the orchestrator. At the initialization of ChainPatrol, the orchestrator shares with the agents the following ChainPatrol parameters: i) SuperBlock size and block size per flow type (as different numbers of packets can be exchanged per flow type, for example, HTTP vs. SSH); ii)Initial seed values per pair of agents (used with a pseudorandom generator for virtual trailer generation); iii) A cryptographic key per pair of agents (for computing the MAC).

5.5.2 Challenges

We discuss the challenges faced during implementation and deployment in Amazon EC2.

Avoiding VNF instrumentation. To ease the deployment of ChainPatrol, it is designed to intercept the egress traffic of a source VNF (to encode virtual trailers before forwarding the traffic) and to monitor the ingress packet of a destination VNF (to decode virtual trailers) without the need for instrumenting those VNFs. A commonly used stream socket cannot achieve this as it only carries the payload but not the header information. Therefore, we resolve to use raw sockets (i.e., netinet, sys and arpa libraries) to attach each instantiated ChainPatrol agent to the interface of a source/destination VNF as a proxy, such that the agent can listen to (and forward) the traffic. Moreover, at the source side, we use iptables to drop the egress packets of the source VNF such that only the watermarked traffic forwarded by the ChainPatrol agent will reach the destination VNF.

Minimizing communications. As ChainPatrol is designed to avoid the communication overhead added by physical trailers, we implement ChainPatrol in such a way as to minimize the overhead of control messages exchanged between its agents and orchestrators. First, a pair of ChainPatrol agents (attached to the source and destination VNFs) do not talk to each other, except at the initiation, when they obtain common seed values from the orchestrator to independently generate the same sequence of virtual trailer fields (i.e., `BlockNum` and `FlowID`) using pseudorandom generators without communications. Second, the ChainPatrol agents and orchestrators communicate through shared storage to reduce communication overhead. We leverage the Amazon ElastiCache for Memcached, which is a highly efficient in-memory key-value store service with a sub-millisecond response time [125], to store and share data between agents and orchestrator (e.g., for performing global attack classification, and for sharing selected nodes of Merkel hash trees for verifying classification results, as mentioned in Section 5.4.2). Finally, we employ

AWS Lambda service [126] to facilitate access to Memcached (otherwise not accessible by tenant-level applications).

Handling unexpected IPD variation. Our virtual trailer encoding scheme already takes into account potential variations in IPDs by computing the watermarking amplitude based on average network jitter (detailed in Section 5.3.2). However, an unexpected amount of IPD variation may still happen, which can render the encoding challenging. Specifically, if an unexpected jitter causes the second packet to arrive so late, that it has not arrived when the agent needs to send it (to match the desired new IPD), then encoding becomes impossible since we cannot send a packet that has not yet arrived. For instance, Figure 5.8 shows two watermark bits $1, 0$ to be encoded with new IPDs $52, 48$ (assuming an average jitter of 2 ms) between two pairs of packets ($P1$ to $P4$). The top timeline depicts their actual arrival time, which shows an unexpected delay of 3 ms between $P3$ and $P4$. As the middle timeline shows, a standard encoding scheme will fail to encode the second bit, since $P4$ is supposed to depart at t=150 (to have an IPD of 48) but by that time it has not yet arrived. One naive solution here is for both agents to use two pre-defined high IPD values (which will never occur naturally) to encode 0 and 1, respectively, such that $P4$ can now be further delayed to encode 0. However, this could lead to significant delay considering that the watermarks in our application are generally large in size so this scenario may occur very often. To address this, we propose a pragmatic X-shifting approach as follows. As the lower axis in Figure 5.8 shows, the first packet in each pair will be proactively delayed by a small amount ($X = 44$ ms in this case) to minimize the likelihood of requiring the costly solution of using pre-defined large IPDs. The source-side agent periodically updates X based on observed IPD variation (note that, unlike watermarking amplitude, the destination agent does not need to know X). Finally, the impact of delaying the first packet of each pair will not add up and hence remains negligible for the entire flow (see experiments in Section 5.6).

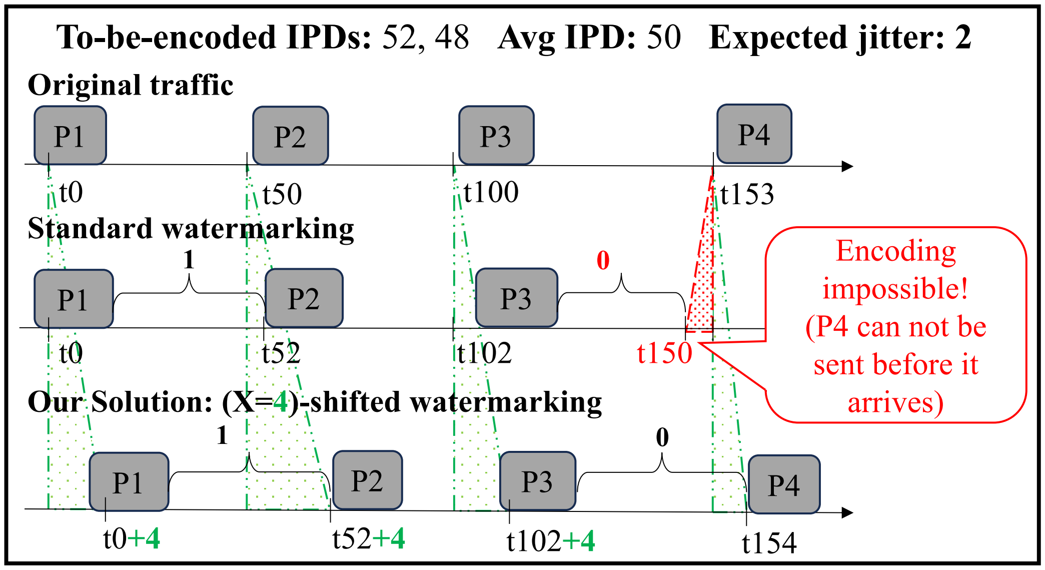

To-be-encoded IPDs: 52, 48 Avg IPD: 50 Expected jitter: 2
Original traffic
P1
P2
P3
P4
t0
t50
t100
t153
Standard watermarking
1
0
P1
P2
P3
t0
t52
t102
t150
Encoding impossible! (P4 can not be sent before it arrives)
Our Solution: (X=4)-shifted watermarking
1
0
P1
P2
P3
P4
t0+4
t52+4
t102+4
t154

Figure 5.8: X-shift for encoding w/ unexpected IPD variation

5.6 Performance Evaluation

5.6.1 Evaluation Environment

Testbed setup. We deploy ChainPatrol in Amazon EC2 using EC2 instances of the `t3a.small` type (i.e., 2 vCPUs of 2.50 GHz AMD EPYC 757, 2 GB RAM, up to 5Gbps of network bandwidth). Each instance runs Amazon Linux 2 (based on Ubuntu 22.04). We leverage the *Memcached database cluster*[1] (a high-performance, distributed memory object caching system) running on a `t3.micro` instance (2 vCPUs of 3.10 GHz Intel Xeon Scalable processor, Skylake 8175M or Cascade Lake 8259CL, 1 GB RAM, up to 5Gbps of network bandwidth) as shared storage between the ChainPatrol orchestrator and agents. We additionally use *Lambda Functions*[2], written in Python 3, as the interface between ChainPatrol and the Memcached cluster. Our evaluation focuses on data plane TCP flows between VNFs inside an SFC, although ChainPatrol can work for other types of traffic.

Datasets. Our evaluation leverages three public datasets [127, 128, 129], an in-house

[1]Amazon ElastiCache: `https://aws.amazon.com/elasticache/memcached/`
[2]AWS Lambda: `https://aws.amazon.com/lambda/`

Table 5.3: Datasets description

Dataset	Avg IPD	IPD range	Source	Application	Packet rate
Dataset1	80ms	70-90ms	Public [127]	Cyber defense	600K/s
Dataset2	7ms	5-10ms	In-house	5G core	500K/s

Free5GC/Kubernetes-based 5G core testbed, and a state-of-the-art solution [1] (for comparison). First, since IPD has the most significant impact on performance, we use both a public cyber defense dataset [127] and the live traffic of our in-house 5G core testbed to collect realistic IPD values. As shown in Table 5.3, those two applications demonstrate distinct ranges of IPDs (80 ms vs. 7 ms) and jitters, which are representative of traditional networks (based on physical infrastructures) and virtual networks (based on containers and virtual machines), respectively. We follow the same distribution as those collected IPDs to create two large live-streaming datasets with up to 600-750 packets per second. In the following, we evaluate ChainPatrol using those two datasets. Additionally, we also compare it to a state-of-the-art solution [1] using the traffic of several real-world applications in two public datasets [128, 129].

5.6.2 Experimental Results

We first evaluate the impact of various ChainPatrol parameters on its effectiveness and overhead. Then, we measure the accuracy and efficiency of its attack detection and classification. Finally, we compare ChainPatrol with a state-of-the-art physical trailer-based solution, namely, AuditBox [1].

Parameters Evaluation

We study how watermarking parameters may affect its effectiveness, and how ChainPatrol parameters may impact the service delay and its overhead.

Watermarking effectiveness. Figure 5.9.A and Figure 5.9.B show the impact of the two

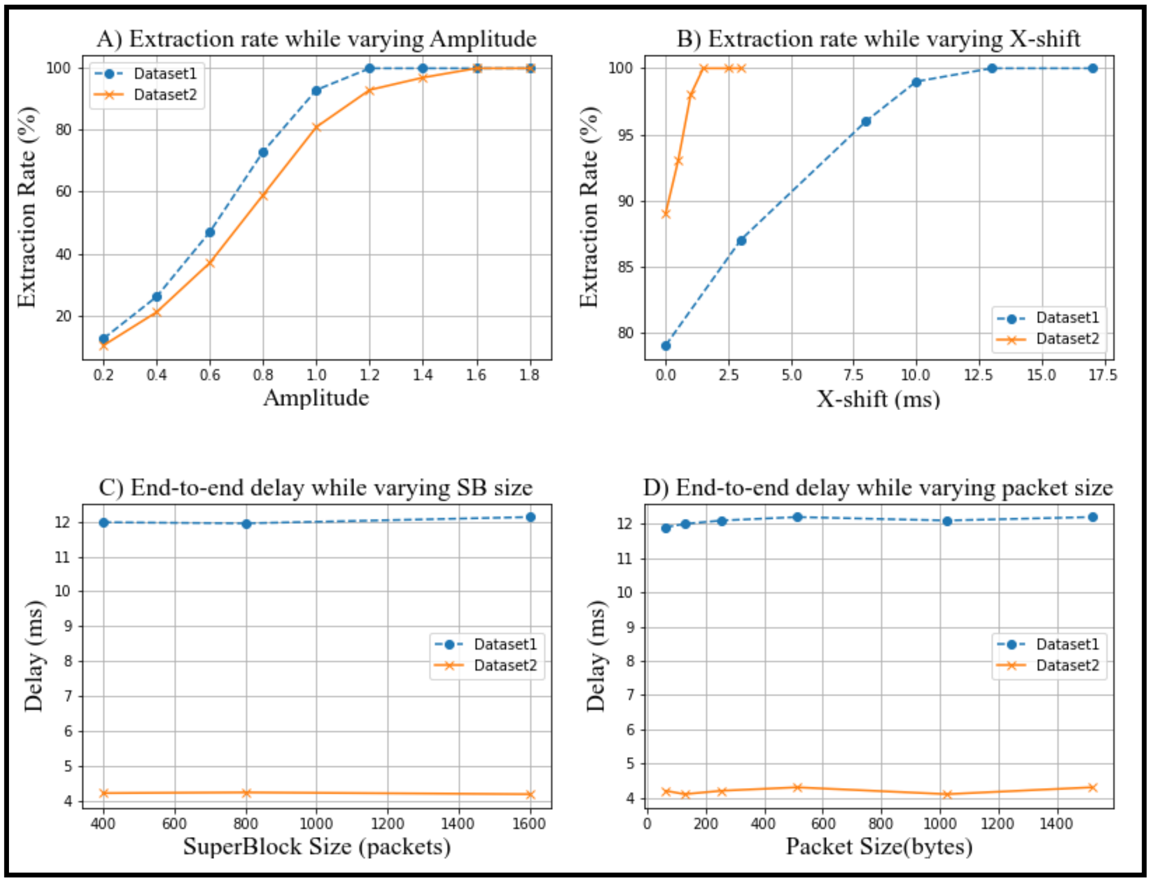


Figure 5.9: Parameters evaluation results

watermarking parameters, i.e., the watermarking amplitude (Section 5.3.2) and the X-shift value (Section 5.5), on the watermark extraction rate (the standard metric for watermarking effectiveness [59]). First, Figure 5.9.A shows that a watermark amplitude value of 1.2 (for Dataset1) and 1.6 (for Dataset2) can already achieve 100% extraction rate. The results also show that Dataset2 generally requires a slightly larger amplitude value than Dataset1 (this can be explained by the relatively higher percentage of jitter in Dataset2, as shown in Table 5.3). Second, Figure 5.9.B shows that a larger X-shift value is required for Dataset1 to achieve 100% extraction rate (which implies this parameter is more closely related to the average IPD). Specifically, Datatset1, whose average IPD is 80 ms, requires $X \approx 3$; Dataset2, whose average IPD is 8 ms, requires $X \approx 0.5$. The results also show that, without our X-shift solution (i.e., $X = 0$), the extraction rate would be significantly lower (less than 80% for Dataset1 and 90% for Dataset2). Note in both experiments, we fix the other parameter (at a value achieving 100% extraction rate).

Service delay. In this set of experiments, we study how the end-to-end service delay may be affected by different ChainPatrol parameters. Figure 5.9.C and Figure 5.9.D show the end-to-end service delay for different SuperBlock sizes (a parameter of ChainPatrol) and packet sizes (a characteristic of the traffic), respectively. Both results show that ChainPatrol introduces negligible end-to-end service delay on both datasets (around 2.1 ms for Dataset1 and 0.68 ms for Dataset2), and varying the SuperBlock and packet size has almost no impact on the end-to-end delay. This is mainly due to the fact that ChainPatrol agents mainly examine the headers (hence packet sizes have little impact), and the virtual trailers are always encoded in the next SuperBlock so the time for generating virtual trailers (which depends on the SuperBlock size) does not affect packet processing (as the trailers are already ready when the packets arrive). Figure 5.10.E shows how the X-shift value affects the end-to-end delay. As we can see, a larger X value generally leads to a lower delay before it reaches the value that produces a 100% extraction rate (mentioned above).

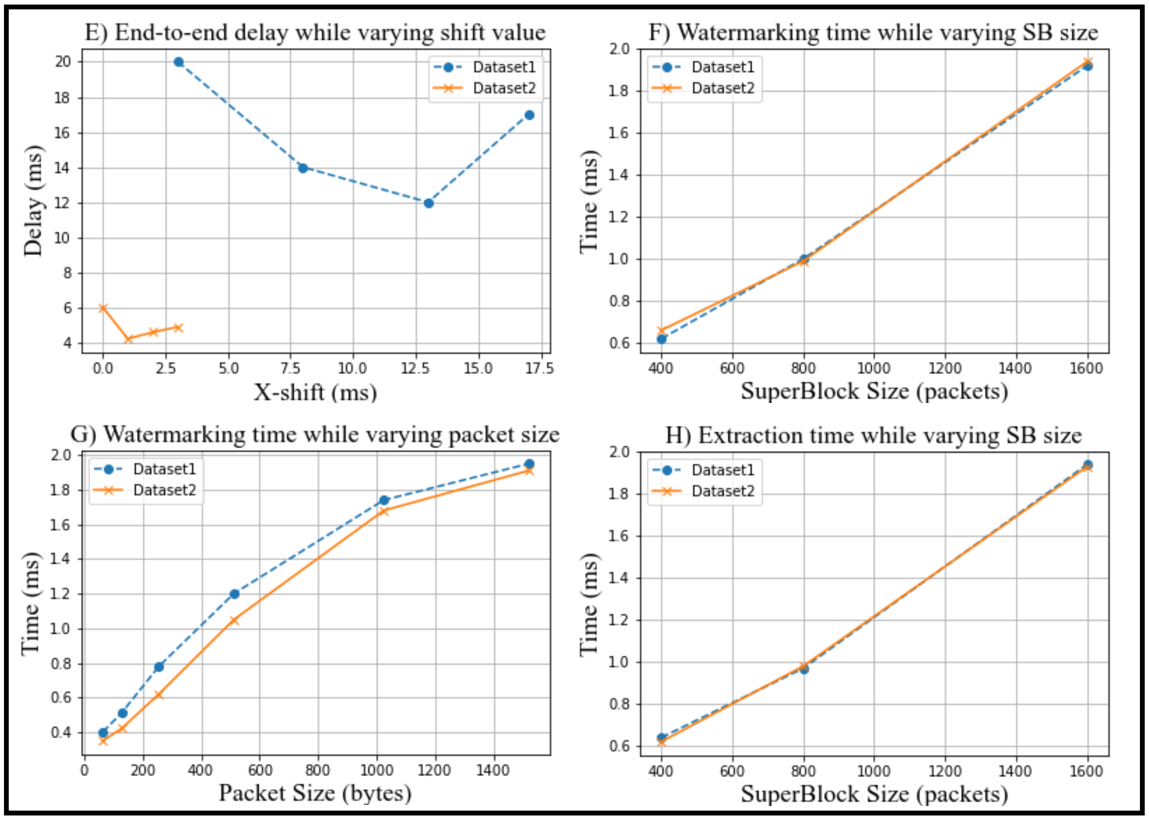


Figure 5.10: Parameters evaluation results

Overhead. In this set of experiments, we evaluate the overhead of ChainPatrol concerning time, memory, and CPU. First, Figure 5.10.F and G show the average watermarking time (i.e., the time taken for a source-side agent to generate and encode virtual trailers) for different SuperBlock and packet sizes. The results show that the watermarking time increases almost linearly in SuperBlock size (as more packets need to be processed) and increases more slowly in packet size (as only the MAC field is affected), while the maximum watermarking time is less than $2ms$ for both datasets. Second, Figure 5.10.H shows the extraction time (i.e., the time taken for a destination-side agent to decode and verify virtual trailers) for different SuperBlock sizes. The results are very similar to Figure 5.10.F since the two agents perform similar but reversed operations (the results on extraction time vs. packet size are also very similar to Figure 5.10.G and hence omitted). Finally, the second to fourth columns of Table 5.4 (the last two columns will be discussed later) show the CPU and memory consumption of the source-side agent, destination-side agent, and orchestrator of ChainPatrol. The resource consumption is very low and does not depend on the block size (which can be explained by the fact that those components mostly perform very simple operations such as concatenation and hash).

Table 5.4: CPU (C) and Memory (M) consumption for ChainPatrol source/destination agents and orchestrator

	Without attacks			With attacks	
Block Size	Source	Dest.	Orch.	Dest.	Orch.
24	C:0.01% M:0.01%	C:0.08% M:0.01%	C:0.01% M:0.01%	C:0.08% M:0.5%	C:0.07% M:0.5%
60	C:0.01% M:0.01%	C:0.08% M:0.01%	C:0.01% M:0.01%	C:0.08% M:0.5%	C:0.07% M:0.5%
96	C:0.01% M:0.01%	C:0.08% M:0.01%	C:0.01% M:0.01%	C:0.08% M:0.5%	C:0.07% M:0.5%

Summary. It is relatively easy (i.e., with small amplitude and X values) for ChainPatrol to ensure the correctness of its decoding of virtual trailers due to the more stable nature of traffic between VNFs (as explained in Section 5.5, ChainPatrol also continuously adjusts those parameters based observed traffic). Moreover, our results show that ChainPatrol causes negligible end-to-end delay to services on both datasets, e.g., the entire flow will be delayed by only about 2.1 ms and 0.68 ms (2.6% and 9.8% of the normal delay between two packets), respectively. Finally, the overhead of ChainPatrol in terms of both time and resources is negligible.

Attacks detection and classification

We evaluate the accuracy and efficiency of attack detection and classification.

Accuracy Evaluation. First, Figure 5.11.A shows that ChainPatrol achieves 100% detection accuracy with up to 50 attacks of different types (as listed in Table 5.2) performed on both datasets. This is expected as the experiment is performed with the ChainPatrol parameters that can ensure 100% watermark extraction rate (detailed in Section 5.6.2), and the extracted virtual trailers possess similar cryptographic properties as physical trailers to guarantee accurate detection (detailed in Section 5.4.1). Second, Figure 5.11.B shows the attack classification accuracy. To evaluate the true capability of virtual trailers for attack classification, this experiment deliberately skips the source-assisted verification step (described in Section 5.4.2), such that all inaccurate classification results will be counted. The results show that the attack classification accuracy stays almost fixed at around 70% under different amounts of attacks for both datasets. In contrast to detection, classification shows a lower accuracy. This is expected due to missing information caused by attacks and potential collision between attack types (e.g., a packet reordering attack may impact virtual trailers same as a combination of packet dropping and injection attacks).

Efficiency Evaluation. First, we evaluate the attack detection time while varying the block size and the number of compromised blocks, respectively. In Figure 5.11.C, the total number of compromised blocks is fixed at 30, and we measure the time required to detect those compromised blocks. As the results show, the detection time increases almost linearly in the block size, with less than 2.4 seconds required for all block sizes for both datasets (no significant difference between the datasets). Note the detection time has no impact on the service delay (due to our multithreading implementation, as detailed in Section 5.5), and the second-level detection time is reasonable since the detection results are meant to be inspected by human experts. Second, we evaluate the time required to detect different amounts of block-level attacks while fixing the total number of SuperBlocks at 50. In Figure 5.11.D, the detection time increases almost linearly in the number of attacks, with up to 5.5 seconds required for detecting all 50 attacks.

Next, we evaluate the attack classification time by both the agents and the orchestrator, for block-level attacks and packet-level attacks, respectively, while varying the number of attacks. First, in Figure 5.11.E, the total number of SuperBlocks is fixed at 50, and we measure the time required to classify different amounts of block-level attacks locally by the destination-side agent. As the results show, the local classification time increases almost linearly in the number of block-level attacks, with around 7.5 seconds required for classifying all the 50 block-level attacks, which is slightly higher than the time for detection (i.e., 5.5 seconds). Second, Figure 5.11.F shows the global classification time taken by the orchestrator for classifying different amounts of packet-level attacks. The results show a similar linear trend, with around 12.5 seconds required for classifying all the 50 attacks. The orchestrator is taking more time since the global classification it performs involves more input information, and the packet-level attack classification rules may require the orchestrator to slide forward more slowly (by two packets, instead of a block, as detailed in Section 5.4.2).

Finally, the last two columns of Table 5.4 show the CPU and memory consumption of the destination-side agent (note source-side agent is not affected) and the orchestrator, respectively, during attack detection and classification. The results show the same CPU consumption of the destination agent as in the no-attack case (third column), whereas the

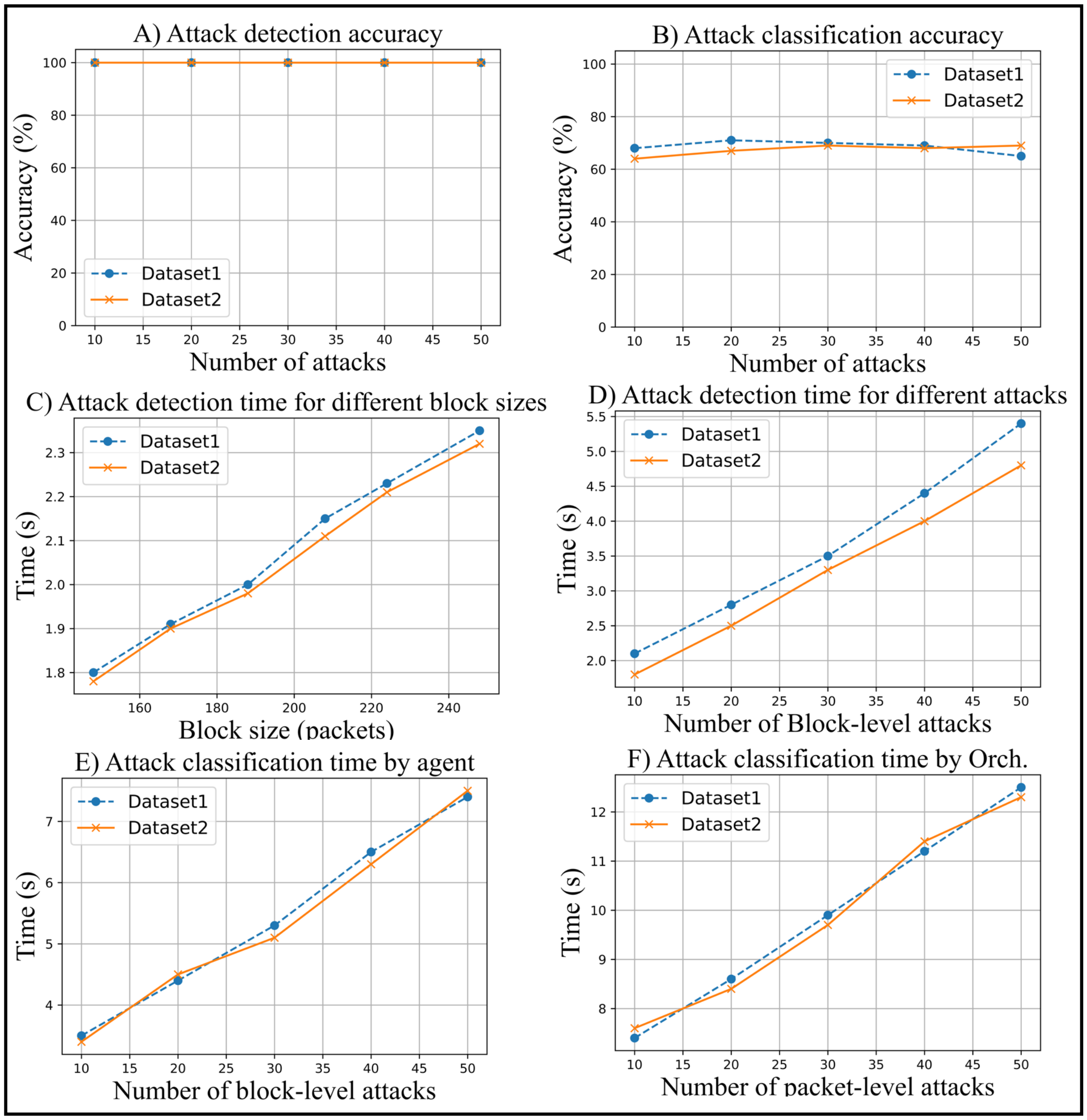


Figure 5.11: Attacks detection and classification results

memory consumption slightly increases. This shows local detection and classification incur negligible computational overhead as the agent only performs simple operations such as concatenations and hashes. In contrast, both the CPU and memory consumption of the orchestrator slightly increases compared to the no-attack case. This is expected as global classification is generally more complex. **Summary.** Virtual trailers can guarantee 100% detection accuracy as their physical counterparts do. Virtual trailers provide a lower (70%) but still acceptable classification accuracy since the result will be further verified in the source-assisted verification step described in Section 5.4.2 (in many cases, the attacked traffic will likely be re-transmitted regardless of attack types). Our results also show attack detection and classification to be efficient (both take a few seconds for 50 attacks), scalable (both show a linear trend), and lightweight (both consume negligible resources).

Comparison with existing work

We compare ChainPatrol to a recent state-of-the-art solution using physical trailers, namely, AuditBox [1]. To ensure the two solutions can be compared under the same environments and settings, we re-implement the approach as specified in [1], and compare it to ChainPatrol using Dataset2 (whose traffic is more representative of SFC applications).

First, we compare the communication overhead added to traffic by ChainPatrol and AuditBox [1] while varying the packet size (with flow size fixed). As Figure 5.6.2.A shows, that ChainPatrol adds almost no extra traffic (ChainPatrol only needs to add a negligible amount of bits at the end of a flow, as discussed in Section 5.3.2), regardless of the packet sizes. On the other hand, the overhead of physical trailers ranges from around 10% (for large packet sizes up to 512 bytes) to 200% (for small packet size of 20 bytes) of the original traffic. Second, we study the level of reduction in communication overhead that can be achieved by ChainPatrol (over AuditBox) for four categories of real-world applications based on two public datasets [128, 129]. As Figure 5.6.2.B shows, ChainPatrol can achieve

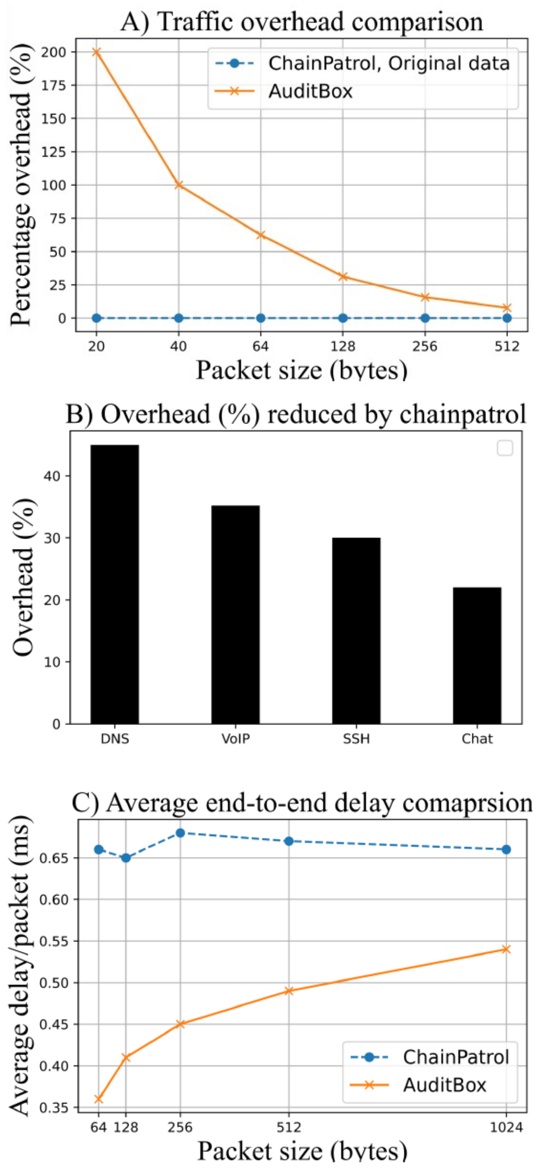


Figure 5.12: Comparative evaluation results

between 45% (for DNS traffic) and 22% (for instant messaging traffic) reduction in communication overhead (the difference in those results can be explained by the distinct packet sizes of those applications). Finally, we compare the end-to-end delay of both solutions while varying the packet size (with fixed flow size). Figure 5.6.2.c shows that the average end-to-end delay of AuditBox ranges between 0.35 and 0.55 ms, while the delay of ChainPatrol is between 0.65 to 0.68 ms. Both of those delays are negligible (less than 10% of the delay of a single packet, as the average IPD is 7 ms). AuditBox incurs only slightly less delay than ChainPatrol, despite the fact that each physical trailer is computed over a single packet (instead of a block of packets, as with ChainPatrol) and directly appended to the packet (hence no need for delaying any packet).

Summary. The comparison between AuditBox [1] and ChainPatrol shows that ChainPatrol introduces almost no extra communication overhead, regardless of the packet or flow sizes, On the other hand, the overhead of physical trailers can be significant, especially for applications whose traffic includes large flows and smaller packets. ChainPatrol is shown to provide a significant reduction in communication overhead (up to 45% of the original traffic) for common applications. Although the more complex virtual trailer design of ChainPatrol inevitably incurs slightly more end-to-end delay, at 0.68 ms the delay is completely negligible for most applications (e.g., 20-30 ms is shown to be noticeable for interactive music, and 100 ms for games [130], and the fastest fiber-based ISPs in US have a 7-13 ms idle latency [131]). Therefore, we conclude that virtual trailers enable ChainPatrol to provide a lightweight solution with both negligible communication overhead and negligible service delay, whereas physical trailers are better for applications with small flows with large packet sizes.

5.7 Summary

Deploying network functions on top of existing cloud infrastructures brings significant benefits but at the same time makes it challenging for tenants to detect cloud-level attacks on their SFCs. Existing solutions based on cryptographic trailers can incur significant overhead for applications with large flows and small packets. In this paper, we proposed a novel concept, *virtual trailer*, which leveraged the inter-packet delay-based side-channel to encode cryptographic trailers without adding extra bits to packets. We developed ChainPatrol, a solution for encoding/decoding virtual trailers and detecting and classifying various SFC attacks based on virtual trailers. We implemented and deployed ChainPatrol based on Amazon EC2, and our experimental results confirmed its effectiveness and efficiency.

Chapter 6

Conclusion

The swift expansion of Network Function Virtualization has captured considerable attention from both industry and academia due to its potential benefits. Nonetheless, the virtualization opens doors to many security issues that must be carefully considered before adopting this technology. To this end, most existing works fail to provide satisfactory solutions for virtualized networks under the NFV constraints. First, relying on the cloud provider may not be sufficient, since modifications made by a stealthy attacker may seem legitimate to the provider. Second, the tenant cannot directly perform the auditing due to limited access to the provider-level data. Finally, shipping all such data to the tenant would incur prohibitive confidentiality concerns. In this thesis, we proposed solutions to overcome the above-mentioned challenges by using data anonymization and side-channel information as the indirect effects of the attacks at the tenant level. To this end, we first proposed an interactive and customizable data anonymization tool to enable the cloud provider to selectively share data in a privacy-preserving manner. Second, we propose an approach to verify the forwarding integrity of virtualized network function chains; which covers a wide range of integrity verification scenarios (i.e., VM injection, passive VM injection, and physical hosts reduction). Third, we proposed a solution for continuous verification of the forwarding integrity of the service chains (i.e., entire service chain bypassing, packet and flows dropping, injection, reordering, modification, reordering, and reply). The following discusses the limitations and our future research focus:

- First, although our interactive and customizable data anonymization tool already leverages deep learning algorithms such as CNN, further enhancing it with the latest advances in generative artificial intelligence to improve the level of user-friendliness would be an interesting future direction.

- Second, our tenant-based auditing solution is based on a side channel. Therefore, the accuracy of our auditing is not as precise as direct observation solutions that have access to the cloud's underlying information. To address this concern, we plan to improve the precision of our techniques by combining complementary information from multiple side channels.

- Third, our continuous verification approach generates a negligible end-to-end delay, which is accumulative over multiple pairs of network functions along the chain. A future direction is to develop more intelligent approaches that can perform verification only between selected pairs of network functions in order to achieve an optimal tradeoff between security and overhead (delay).

- Finally, although we have separately integrated each of our solutions into various cloud platforms and applications (e.g., OpenStack, Amazon EC2, Open5GS, and free5GC), a future direction is to deploy and integrate those solutions as complementary and cohesive modules of the same environment.

Bibliography

[1] G. Liu, H. Sadok, A. Kohlbrenner, B. Parno, V. Sekar, and J. Sherry, "Don't yank my chain: Auditable NF service chaining," in *NSDI.USENIX.*, 2021.

[2] A. Assila, H. Ezzedine *et al.*, "Standardized usability questionnaires: Features and quality focus," *Electronic Journal of Computer Science and Information Technology: eJCIST*, vol. 6, no. 1, 2016.

[3] OpenStack, "OpenStack," 2021, available at: https://www.openstack.org/.

[4] ETSI, "Network functions virtualisation (NFV) release management and orchestration; architecture enhancement for security management specification," 2018.

[5] M. Zoure, T. Ahmed, and L. Réveillére, "Network services anomalies in nfv: Survey, taxonomy, and verification methods," *IEEE Transactions on Network and Service Management*, 2022.

[6] N. C. Thang and M. Park, "Detecting compromised switches and middlebox-bypass attacks in service function chaining," in *2019 29th International Telecommunication Networks and Applications Conference (ITNAC).* IEEE, 2019, pp. 1–6.

[7] 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. IEEE.*, 2019.

[8] Y. Yue and B. Cheng, "Easyorchestrator: An end-user oriented network service creation platform with verification mechanism," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2019, pp. 1–6.

[9] 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 *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2016, pp. 134–140.

[10] N. Bouten, R. Mijumbi, J. Serrat, J. Famaey, S. Latré, and F. De Turck, "Semantically enhanced mapping algorithm for affinity-constrained service function chain requests," *IEEE Transactions on Network and Service Management*, vol. 14, no. 2, pp. 317–331, 2017.

[11] X. Zhang, Q. Li, Z. Zhang, J. Wu, and J. Yang, "Vsfc: Generic and agile verification of service function chains in the cloud," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 78–91, 2020.

[12] M. K. Shin, Y. Choi, H. H. Kwak, S. Pack, M. Kang, and J. Y. Choi, "Verification for NFV-enabled network services," in *IEEE ICTC*, 2015.

[13] *Amazon ECS*. [Online]. Available: https://aws.amazon.com/ecs/

[14] G. Szarvas, R. Farkas, and R. Busa-Fekete, "State-of-the-art anonymization of medical records using an iterative machine learning framework," *Journal of the American Medical Informatics Association*, vol. 14, no. 5, pp. 574–580, 2007.

[15] R. J. Abbott, "Program design by informal english descriptions," *Communications of the ACM*, vol. 26, no. 11, pp. 882–894, 1983.

[16] A. I. Antón and J. B. Earp, "A requirements taxonomy for reducing web site privacy vulnerabilities," *Requirements engineering*, vol. 9, no. 3, pp. 169–185, 2004.

[17] T. D. Breaux and F. Schaub, "Scaling requirements extraction to the crowd: Experiments with privacy policies," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 2014, pp. 163–172.

[18] J. Bhatia, T. D. Breaux, and F. Schaub, "Mining privacy goals from privacy policies using hybridized task recomposition," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 25, no. 3, pp. 1–24, 2016.

[19] M. Foukarakis, D. Antoniades, S. Antonatos, and E. P. Markatos, "Flexible and high-performance anonymization of netflow records using anontool," in *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*. IEEE, 2007, pp. 33–38.

[20] Y. Li, A. Slagell, K. Luo, and W. Yurcik, "Canine: A combined conversion and anonymization tool for processing netflows for security," in *International conference on telecommunication systems modeling and analysis*, vol. 21, 2005.

[21] D. Moore, K. Keys, R. Koga, E. Lagache, and K. C. Claffy, "The coralreef software suite as a tool for system and network administrators," in *Proceedings of the 15th USENIX conference on System administration*. USENIX Association, 2001, pp. 133–144.

[22] A. J. Slagell, K. Lakkaraju, and K. Luo, "Flaim: A multi-level anonymization framework for computer and network logs." in *LISA*, vol. 6, 2006, pp. 3–8.

[23] Eddie Kohler, "ipsumdump tool," 2015, available at: https://read.seas.harvard.edu/~kohler/ipsumdump/.

[24] P. Haag, "Nfdump," *Available from World Wide Web: http://nfdump. sourceforge. net*, 2010.

[25] W. Yurcik, C. Woolam, G. Hellings, L. Khan, and B. Thuraisingham, "Scrub-tcpdump: A multi-level packet anonymizer demonstrating privacy/analysis tradeoffs," in *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on.* IEEE, 2007, pp. 49–56.

[26] Franceesco Gringoli, "Tcpanon tool," 2019, available at: http://netweb.ing.unibs.it/~ntw/tools/tcpanon/.

[27] Greg Minshall of Ipsilon Networks, "Tcpdpriv," 2005, available at: http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html.

[28] R. Pang, M. Allman, V. Paxson, and J. Lee, "The devil and packet trace anonymization," *ACM SIGCOMM*, 2006.

[29] Ethan Blanton, "Tcpurify tool," 2019, available at: https://web.archive.org/web/20140203210616/irg.cs.ohiou.edu/~eblanton/tcpurify/.

[30] M. Flittner, J. M. Scheuermann, and R. Bauer, "ChainGuard: Controller-independent verification of service function chaining in cloud computing," in *IEEE NFV-SDN*, 2017, pp. 1–7.

[31] K. Bu, Y. Yang, Z. Guo, Y. Yang, X. Li, and S. Zhang, "Flowcloak: Defeating middlebox-bypass attacks in software-defined networking," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications.* IEEE, 2018, pp. 396–404.

[32] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing {Network-Wide} policies in the presence of dynamic middlebox actions using {FlowTags}," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 543–546.

[33] G. Marchetto, R. Sisto, J. Yusupov, and A. Ksentini, “Virtual network embedding with formal reachability assurance,” in *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018, pp. 368–372.

[34] S. Van Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester, “VNF Performance Modelling: From stand-alone to chained topologies,” *CN*, vol. 181, 2020.

[35] S. K. Fayazbakhsh, M. K. Reiter, and V. Sekar, “Verifiable network function outsourcing: requirements, challenges, and roadmap,” in *MNFV*, 2013.

[36] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, “An analysis of performance interference effects in virtual environments,” in *ISPASS*, 2007.

[37] Y. Mei, L. Liu, X. Pu, S. Sivathanu, and X. Dong, “Performance analysis of network i/o workloads in virtualized data centers,” *IEEE TSC*, vol. 6, no. 1, pp. 48–63, 2011.

[38] D. Battré, N. Frejnik, S. Goel, O. Kao, and D. Warneke, “Inferring network topologies in infrastructure as a service cloud,” in *CCGRID*. IEEE, 2011.

[39] M. Peuster and H. Karl, “Understand your chains: Towards performance profile-based network service management,” in *EWSDN*, 2016.

[40] P. Naik, D. K. Shaw, and M. Vutukuru, “NFVPerf: Online performance monitoring and bottleneck detection for NFV,” in *NFV-SDN*, 2016.

[41] R. V. Rosa, C. E. Rothenberg, and R. Szabo, “VBaaS: VNF benchmark-as-a-service,” in *EWSDN*, 2015.

[42] Linux, “Traceroute,” 2021, available at: t.ly/tq0k.

[43] Jon Dugan et al, “active measurements of the maximum achievable bandwidth on IP networks.” 2021, available at: https://iperf.fr/iperf-doc.php.

[44] A. Chen, J. Cao, and T. Bu, “Network tomography: Identifiability and fourier domain estimation,” *IEEE TSP*, 2010.

[45] D. Arifler, G. de Veciana, and B. L. Evans, “Network tomography based on flow level measurements,” in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2. IEEE, 2004, pp. ii–437.

[46] F. Valenza, S. Spinoso, and R. Sisto, “Formally specifying and checking policies and anomalies in service function chaining,” *Journal of Network and Computer Applications*, vol. 146, p. 102419, 2019.

[47] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, “Lightweight source authentication and path validation,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 271–282.

[48] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller, and A. Seehra, “Verifying and enforcing network paths with ICING,” in *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies*, 2011, pp. 1–12.

[49] IETF, “Proof of transit,” 2020, available at: https://datatracker.ietf.org/doc/draft-ietf-sfc-proof-of-transit/.

[50] X. Zhang, Q. Li, J. Wu, and J. Yang, “Generic and agile service function chain verification on cloud,” in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE, 2017, pp. 1–10.

[51] S. Yao, M. Xu, Q. Li, J. Cao, and Q. Song, “cSFC: Building credible service function chain on the cloud,” in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[52] X. Wang and D. S. Reeves, “Robust correlation of encrypted attack traffic through

stepping stones by manipulation of interpacket delays," in *Proceedings of the 10th ACM conference on Computer and communications security*, 2003, pp. 20–29.

[53] A. Houmansadr and N. Borisov, "The need for flow fingerprints to link correlated network flows," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2013, pp. 205–224.

[54] A. Houmansadr, N. Kiyavash, and N. Borisov, "Non-blind watermarking of network flows," *IEEE/ACM Transactions on Networking*, vol. 22, no. 4, pp. 1232–1244, 2013.

[55] P. Kadian, S. M. Arora, and N. Arora, "Robust digital watermarking techniques for copyright protection of digital data: A survey," *Wireless Personal Communications*, vol. 118, pp. 3225–3249, 2021.

[56] N. Kiyavash, A. Houmansadr, and N. Borisov, "Multi-flow attacks against network flow watermarking schemes." in *USENIX security symposium*. Berkeley, CA, 2008, pp. 307–320.

[57] Z. Lin and N. Hopper, "New attacks on timing-based network flow watermarks," in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 381–396.

[58] C. Qin, P. Ji, X. Zhang, J. Dong, and J. Wang, "Fragile image watermarking with pixel-wise recovery based on overlapping embedding strategy," *Signal processing*, vol. 138, pp. 280–293, 2017.

[59] A. Iacovazzi and Y. Elovici, "Network flow watermarking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 512–530, 2016.

[60] Y. H. Park and D. S. Reeves, "Adaptive timing-based active watermarking for attack attribution through stepping stones," North Carolina State University. Dept. of Computer Science, Tech. Rep., 2007.

[61] Z. Pan, H. Peng, X. Long, C. Zhang, and Y. Wu, “A watermarking-based host correlation detection scheme,” in *2009 International Conference on Management of e-Commerce and e-Government*. IEEE, 2009, pp. 493–497.

[62] P. Peng, P. Ning, D. S. Reeves, and X. Wang, “Active timing-based correlation of perturbed traffic flows with chaff packets,” in *25th IEEE International Conference on Distributed Computing Systems Workshops*. IEEE, 2005, pp. 107–113.

[63] F. Rezaei and A. Houmansadr, “TagIt: Tagging network flows using blind fingerprints.” *Proc. Priv. Enhancing Technol.*, vol. 2017, no. 4, pp. 290–307, 2017.

[64] H. Fang, Y. Qiu, K. Chen, J. Zhang, W. Zhang, and E.-C. Chang, “Flow-based robust watermarking with invertible noise layer for black-box distortions,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, 2023, pp. 5054–5061.

[65] J. Xu, S. Koffas, O. Ersoy, and S. Picek, “Watermarking graph neural networks based on backdoor attacks,” in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, 2023, pp. 1179–1197.

[66] C. Tenopir, S. Allard, K. Douglass, A. U. Aydinoglu, L. Wu, E. Read, M. Manoff, and M. Frame, “Data sharing by scientists: practices and perceptions,” *PLoS one*, vol. 6, no. 6, p. e21101, 2011.

[67] Marty Swant, “People are becoming more reluctant to share personal data, survey reveals,” 2021, available at: t.ly/tydm.

[68] Josh D’Addario, “New survey finds british businesses are reluctant to proactively share data,” 2020, available at: https://theodi.org/article/new-survey-finds-just-27-of-british-businesses-are-sharing-data/.

[69] EU General Data Protection Regulation, “Fines and Penalties,” 2018, available at: https://www.gdpreu.org/compliance/fines-and-penalties/.

[70] T. Brekne, A. Årnes, and A. Øslebø, “Anonymization of ip traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies,” in *PET*. Springer, 2005, pp. 179–196.

[71] A. Majeed and S. Lee, “Anonymization techniques for privacy preserving data publishing: A comprehensive survey,” *IEEE Access*, 2020.

[72] G. Cormode and D. Srivastava, “Anonymized data: generation, models, usage,” in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 1015–1018.

[73] W. Zhang, Y. Lin, S. Xiao, J. Wu, and S. Zhou, “Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing,” *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1566–1577, 2015.

[74] Sys4 consults, “a generic log anonymizer,” 2018, available at: https://github.com/sys4/loganon.

[75] IMPREVA, “Camouflage data masking,” 2018, available at: https://www.imperva.com/products/data-security/data-masking/.

[76] Google, “Traces from Requests Processed by Google Cluster Management System,” 2019, available at: https://github.com/google/cluster-data.

[77] R. S. Sandhu, “Lattice-based access control models,” *Computer*, no. 11, pp. 9–19, 1993.

[78] B. A. Davey and H. A. Priestley, *Introduction to lattices and order*. Cambridge university press, 2002.

[79] E. D. Bell and J. L. La Padula, “Secure computer system: Unified exposition and multics interpretation,” Bedford, MA, 1976. [Online]. Available: http://csrc.nist.gov/publications/history/bell76.pdf

[80] R. Collobert and J. Weston, “A unified architecture for natural language processing: Deep neural networks with multitask learning,” in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160–167.

[81] UCIMLR, “Burst Header Packet Flooding Attack on Optical Burst Switching Network Data Set,” 2019, available at: https://archive.ics.uci.edu/ml/datasets/.

[82] UCI, “Machine Learning Repository,” 2020, available at: https://archive.ics.uci.ed u/ml/datasets.php.

[83] P. Jiang, Q. Wang, M. Huang, C. Wang, Q. Li, C. Shen, and K. Ren, “Building in-the-cloud network functions: Security and privacy challenges,” *Proceedings of the IEEE*, vol. 109, no. 12, pp. 1888–1919, 2021.

[84] Ammar Latif, Ash Khamas, Sundeep Goswami, Vara Prasad Talari, and Dr Young Jung., “Telco meets aws cloud: Deploying DISH’s 5G network in AWS cloud,” 2022, available at: https://aws.amazon.com/blogs/industries/telco-meets-aws-cloud -deploying-dishs-5g-network-in-aws-cloud/.

[85] VMware, “VMware expands its VMware ready for telco cloud program to accelerate the deployment of 5G services,” 2020, available at:t.ly/BIIW.

[86] V. Moorthy, R. Venkataraman, and T. R. Rao, “Security and privacy attacks during data communication in software defined mobile clouds,” *Computer Communications*, 2020.

[87] Y. Zhang, W. Wu, S. Banerjee, J.-M. Kang, and M. A. Sanchez, “Sla-verifier: Stateful and quantitative verification for service chaining,” in *INFOCOM*, 2017.

[88] M. Peuster and H. Karl, "Profile your chains, not functions: Automated network service profiling in devops environments," in *NFV-SDN*, 2017.

[89] S. Lakshmanan, M. Zhang, S. Majumdar, Y. Jarraya, M. Pourzandi, and L. Wang, "Caught-in-translation (cit): Detecting cross-level inconsistency attacks in network functions virtualization (nfv)," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[90] J. Aikat, A. Akella, J. S. Chase, A. Juels, M. K. Reiter, T. Ristenpart, V. Sekar, and M. Swift, "Rethinking security in the era of cloud computing," *IEEE S&P*, 2017.

[91] H. Wang, H. Sayadi, A. Sasan, S. Rafatirad, and H. Homayoun, "Hybrid-shield: Accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks," in *CCD*, 2020, pp. 1–9.

[92] *Tacker*. [Online]. Available: https://wiki.openstack.org/wiki/Tacker

[93] M. Bunyakitanon, A. P. da Silva, X. Vasilakos, R. Nejabati, and D. Simeonidou, "Auto-3P: An autonomous VNF performance prediction & placement framework based on machine learning," *CN*, 2020.

[94] M. Anisetti, C. A. Ardagna, F. Gaudenzi, E. Damiani, N. Diomede, and P. Tufarolo, "Moon cloud: a cloud platform for ict security governance," in *2018 IEEE (GLOBECOM)*. IEEE, 2018.

[95] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," *IEEE TNSM*, 2015.

[96] LinuxPerf, "Profiling with performance counters," 2021, available at: t.ly/miMd.

[97] OProfile, "Linux system profiler," 2022, available at: t.ly/rqN0.

[98] S. Van Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester, "Profile-based resource allocation for virtualized network functions," *IEEE TNSM*, vol. 16, no. 4, pp. 1374–1388, 2019.

[99] S. Yu, G. Xiaolin, L. Jiancai, Z. Xuejun, and W. Junfei, "Detecting vms co-residency in cloud: Using cache-based side channel attacks," *Elektronika*, 2013.

[100] M. Oqaily, Y. Jarraya, M. Zhang, L. Wang, M. Pourzandi, and M. Debbabi, "icat: An interactive customizable anonymization tool," in *European Symposium on Research in Computer Security*. Springer, 2019, pp. 658–680.

[101] A. Oqaily, L. Sudershan, Y. Jarraya, S. Majumdar, M. Zhang, M. Pourzandi, L. Wang, and M. Debbabi, "NFVGuard: Verifying the security of multilevel (NFV) stack," in *CloudCom*, 2020.

[102] N. Duffield, "Network tomography of binary network performance characteristics," *IEEE TIT*, vol. 52, no. 12, pp. 5373–5388, 2006.

[103] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester, *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer Science & Business Media, 2005.

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

[105] Scikit-learn, "Machine learning in python," 2021, https: https://numpy.org/.

[106] Ubuntu, "Cloud images," 2021, available at: https://cloud-images.ubuntu.com/.

[107] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *NSDI. USENIX*, 2015.

[108] *Tcpdump*. [Online]. Available: https://www.tcpdump.org/

[109] Snort Org, “snort,” 2021, available at: https://www.snort.org/.

[110] Netfilter Org, “IPTables,” 2021, available at: https://www.netfilter.org/.

[111] K. Tian, S. T. Jan, H. Hu, D. Yao, and G. Wang, “Needle in a haystack: Tracking down elite phishing domains in the wild,” in *IMC*, 2018, pp. 429–442.

[112] Soumith Chintala, Emily Denton, Martin Arjovsky, Michael Mathieu, “How to train a GAN? tips and tricks to make GANs work,” 2021, available at: https://github.com/soumith/ganhacks.

[113] Z. Lin, Y. Shi, and Z. Xue, “IDSGAN: Generative adversarial networks for attack generation against intrusion detection,” *arXiv preprint arXiv:1809.02077*, 2018.

[114] D. Volkhonskiy, I. Nazarov, and E. Burnaev, “Steganographic generative adversarial networks,” in *Twelfth international conference on machine vision (ICMV 2019)*, vol. 11433. SPIE, 2020, pp. 991–1005.

[115] M. R. Shahid, G. Blanc, H. Jmila, Z. Zhang, and H. Debar, “Generative deep learning for internet of things network traffic generation,” in *PRDC*, 2020.

[116] S. T. Jan, Q. Hao, T. Hu, J. Pu, S. Oswal, G. Wang, and B. Viswanath, “Throwing darts in the dark? detecting bots with limited data using neural data augmentation,” in *IEEE S&P*, 2020.

[117] F. T. Jaigirdar, C. Rudolph, and C. Bain, “Risk and compliance in IoT-health data propagation: A security-aware provenance based approach,” in *ICDH*, 2021.

[118] OSM Group, “Open source mano,” 2021, available at: https://osm.etsi.org/.

[119] OPNFV Group, 2021, available at: https://www.opnfv.org/.

[120] M. Legner, T. Klenze, M. Wyss, C. Sprenger, and A. Perrig, "EPIC: Every packet is checked in the data plane of a path-aware internet," in *Proceedings of the 29th USENIX Conference on Security Symposium*, 2020, pp. 541–558.

[121] N. Alhebaishi, L. Wang, and S. Jajodia, "Modeling and mitigating security threats in network functions virtualization (NFV)," in *IFIP DBSec*. Springer, 2020.

[122] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet of Things*, vol. 12, p. 100273, 2020.

[123] N. Agarwal, A. K. Singh, and P. K. Singh, "Survey of robust and imperceptible watermarking," *Multimedia Tools and Applications*, vol. 78, pp. 8603–8633, 2019.

[124] M. S. Niaz and G. Saake, "Merkle hash tree based techniques for data integrity of outsourced data," *GvD*, vol. 1366, pp. 66–71, 2015.

[125] Amazon AWS, "Common elasticache use cases and how elasticache can help," 2023, available at: https://docs.aws.amazon.com/AmazonElastiCache/latest/mem-ug/elasticache-use-cases.html.

[126] AWS, "Building lambda functions with python," 2023, available at: https://docs.aws.amazon.com/lambda/latest/dg/lambda-python.html.

[127] Netresec, "Publicly available PCAP files," 2023, available at: https://www.netresec.com/?page=PcapFiles.

[128] A. Habibi Lashkari., G. Draper Gil., M. S. I. Mamun., and A. A. Ghorbani., "Characterization of tor traffic using time based features," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - ICISSP*, INSTICC. SciTePress, 2017, pp. 253–262.

[129] S. Jorgensen, J. Holodnak, J. Dempsey, K. d. Souza, A. Raghunath, V. Rivet, N. DeMoes, A. Alejos, and A. Wollaber, “Extensible machine learning for encrypted network traffic application labeling via uncertainty quantification,” *IEEE Transactions on Artificial Intelligence*, pp. 1–15, 2023.

[130] S. Liu, X. Xu, and M. Claypool, “A survey and taxonomy of latency compensation techniques for network computer games,” *ACM Comput. Surv.*, vol. 54, no. 11s, sep 2022.

[131] Federal Communications Commission, “Measuring fixed broadband - twelfth report,” 2023, available at: https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-twelfth-report.