Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (32nd cycle)

Security and trust in a Network Functions Virtualisation infrastructure

Marco De Benedictis

* * * * * *

Supervisor

Prof. Antonio Lioy, Supervisor

Politecnico di Torino
July 6, 2020

. . . .

Marco De Benedictis
Turin, July 6, 2020

Summary

Modern digital infrastructures are undergoing a significant evolution thanks to the advantages offered by virtualisation techniques in terms of flexibility, scalability and the overall reduction of hardware-related costs. More specifically, the Cloud computing paradigm foreshadows large scale virtualisation as a viable technology to manage on-demand allocation and distributed deployment of computing resources in a dynamic environment. More recently, virtualisation has gained momentum in the networking domain as well, where network operators are exploring technologies to enhance the flexibility of their infrastructure and to reduce the overall provisioning, maintenance and upgrade costs associated to traditional appliances. In this regard, the latest trend concerns the implementation of networking functions (i.e. routers, switches, Network Address Translation boxes) in softwarised instances that run on top of commodity hardware in a data-center. This aims to achieve a higher degree of scalability of network functions when compared to traditional hardware-based infrastructures, wherein each topology change and service deployment typically imply a physical manipulation at the appliance level. Moreover, the operators are interested in reducing the vendor lock-in, which often leads to substantial upgrade and maintenance costs.

From a security perspective, virtualisation exposes network infrastructures to different families of threats. In particular, software modules may include vulnerabilities that can be exploited remotely, compromising both the network itself and its clients' privacy, both at virtualisation and networking level. Additionally, the softwarisation of the network makes it more prone to software bugs introduced by the developers. Given the privacy sensitive nature of public networks, security and trustworthiness of the platform are considered paramount. Because of this, network virtualisation should be supported by appropriate means to ensure that the software domain is protected against manipulations and that an attack can be detected by the monitoring systems.

In this thesis, we propose a platform to assess the trustworthiness of a softwarised network infrastructure. This offers a generic approach to integrity verification so that heterogeneous virtualisation platforms can be protected. This is particularly relevant in today's cloud infrastructures, that adopt different virtualisation strategies ranging from traditional virtual machines to more light-weight forms of virtualisation (i.e. containers). In the proposed approach, adherence to existing standards on network softwarisation and hardware platform trust is considered paramount to ensure market readiness of the solution, and ease its application by existing frameworks. We have developed the system within the SHIELD Horizon 2020 project, that aimed to the definition of a secure platform based on an interplay of network softwarisation, trusted computing, and artificial intelligence to both support the deployment of networking functions in a operator network and to monitor their life-cycle against external attacks or malfunctions. Compared to existing approaches, the system offers a generic approach to network integrity verification, making it applicable to heterogeneous hardware platforms. Moreover, it targets elements acting both at the physical and virtual level to protect the entire cloud software stack. This work addresses the current limitations in the state of the art in the field of security and trust of a virtualised network infrastructures with the following contributions: (1) a trust architecture tailored for a highly-virtualised environment that targets both the physical and the virtual domains of execution; (2) an integrity verification technique that enables run-time attestation of lightweight virtualised instances against manipulation by external attackers; (3) a monitoring process that enhances the threat response capabilities in a softwarised network infrastructure by integrating the previous contributions in a cloud practical scenario.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computer networks have been of significant importance since their initial definition, addressing the need of inter-communication between computers, and ultimately the end users. Although computers have revolutionised the world we live in since their initial mass production, by defining new ways to perform human activities and even creating completely new use cases, networking has been a key factor towards the global exploitation of Information and Communication Technology (ICT). In fact, without computer networks there would not be a way to offer a remote service to users, as all software would strictly need to be executed locally on the machine. Distributed computer architectures would not exist, as all the computing would be centralised and there would be no way to exchange information among machines. Ultimately, there would not be *Internet*, which is by many considered the latest industrial revolution in human history.

During the last few years, networks have evolved by gaining a prominent role in the way information technology is consumed by end users. In fact, more and more technological services are not directly managed by the consumers anymore, as they are provisioned by often humongous companies that own the largest portion of the market, such as Facebook, Amazon, Google, and Netflix. The end user typically owns a client device, such as a smartphone, a laptop, or a smart TV, which grants him access to these services via the network. In fact, many software development companies have completely abandoned the target of creating *native* applications that run on an Operating System (OS) in favour of *web apps* that are easily accessible via a modern browser and therefore can reach a variety of end user devices, such as traditional personal computers, mobile phones and even smartwatches. This paradigm shift also means that information is often managed and stored outside of the user device, as in the *"cloud"*. This approach makes the end user a consumer of his own data (e.g. photos, documents) rather than the sole owner of such data.

1.1 Cloud Computing

Cloud Computing is a key enabler for this digital transformation, as it defines novel service models wherein the computing, storage, and networking resources can be allocated on-demand, according to the needs of the services and applications that are often intertwined with the activity of the end users. Large-scale virtualisation and dynamic allocation of resources are at the base of the inherent *scalability* of cloud infrastructures, as each service can either be replicated or granted more resources in a flexible manner. With respect to data, large clouds have *latency* and *availability* guarantees that are enabled by the geographical distribution of their infrastructure, which allows the Cloud Service Provider (CSP) to provide information at the edge of the network faster and more consistently. Traditionally, the cloud paradigm addressed the need for sharing computational power and storage among different services, in an effort to optimise the overall utilisation of physical resources made available by the hardware. In this regard, several service models have been identified in literature to address the end users' needs to have different levels of control towards the cloud resources: Infrastructure as a Service (IaaS), wherein the CSP offers a share of the cloud *raw* resources to its client, i.e. a *tenant*, so that a fully customisable virtual environment can be set up, including the selection of virtualised nodes, their storage volumes and the local network links that interconnect them; Platform as a Service (PaaS), wherein the cloud infrastructure takes care of the execution environment so that the user can focus on the development and deployment of applications on top of it; Software as a Service (SaaS), where the CSP fully manages the software itself and just provides access to the end user — either anonymous or subject to authentication. In this regard, both private and public clouds are feasible. The first requires the infrastructure to be entirely managed by its user, i.e. the company or organisation that aims to leverage the cloud services. The latter, as effectively demonstrated by solutions such as Amazon Web Services and Google Cloud Platform, allows the end user to just use the cloud resources made publicly available by the CSP in return of a fee. Technology-wise, cloud service models are exposed to heterogeneous forms of virtualisation. Traditional virtualisation options are enabled by hypervisors, managers that allocate resources to virtual machines and are responsible for their life-cycle. Typically, hypervisors leverage the isolation capabilities offered by the OS kernel so that each Virtual Machine (VM) has a limited view of the resources available at the physical level, yet they are presented to the VM as standard components (e.g. devices, memory, disk). More recently, OS-level virtualisation has gained momentum, proposing a novel mechanism to isolate processes rather than the complete execution environment in lightweight virtualised instances, i.e. *containers*. The target virtualisation technology is relevant to the CSP as it drives the selection of a target cloud deployment framework. Different solutions are available both as open-source tools (e.g. OpenStack, and Kubernetes) and proprietary services (e.g. OpenShift, VMware).

1.2 Network virtualisation

Compared to modern computing architectures, networks have traditionally been grounded in physical infrastructures composed of hardware *appliances*, specific to vendors of networking equipment and typically subject to non-negligible maintenance and upgrade costs. This approach has been driven by the need for high network performance and resilience, at the cost of flexibility and of limited service provisioning. In fact, whenever the network had to support new services or just update their configuration, this would require its administrator to log to the specific appliance console and to issue commands in a proprietary language so that the network could be adapted. Although this approach is still relevant today in certain scenarios such as home environments or small enterprises, it cannot keep up with the ever increasing needs of flexibility demanded by modern ICT services.

Because of this, the scientific community has recently addressed the limitations of traditional network infrastructures by proposing novel technologies that leverage the same key enablers of cloud computing, namely large-scale virtualisation, on-demand orchestration of resources and centralised dynamic configuration.

Network Functions Virtualisation (NFV) is a novel paradigm, standardised by the European Telecommunications Standards Institute (ETSI), that proposes a framework to deploy network components in virtualised instances on top of a cloud infrastructure. Compared to the traditional network scheme, NFV allows to reduce the overall maintenance and upgrade costs by switching from proprietary appliances to commodity hardware, wherein each networking capability is implemented in software by a Virtual Network Function (VNF) that runs on top of the hardware. Similarly, Software-Defined Networking (SDN) proposes software mechanisms to configure network components via a centralised management plane that is separated from the forwarding plane, in contrast with traditional packet processing. NFV and SDN are considered complementary to each other, given their common goal in addressing the limitations of traditional networks in terms of vendor lock-in, on-demand deployment and reconfiguration of network components. Moreover, they both define novel orchestration and management elements that reside in the administrative domain of the Internet Service Provider (ISP) so that they centralise the services' deployment and configuration. Among the recent trends, Edge Computing paradigm has also proposed technologies to distribute computing as close as possible to the end user at the edge of the network, so that both the service latency is reduced and a more optimised management of network resources can be achieved. Because of this, Edge is particularly relevant for operators and is considered a key enabler for the *5G* network, particularly in combination with NFV. Internet of Things (IoT) and smart cities can benefit from network *softwarisation* as well, as traditional networks cannot cope with the requirements of the diversity of IoT devices and the dynamic grid of services. In a software network scenario, the ISP may own a cloud infrastructure — acting as a CSP — wherein the users' traffic flows through network services implemented in software and managed by a centralised orchestration domain.

1.3 Security and privacy of virtualised networks

Security and privacy regulations are very relevant as of today, as both national and international bodies are discussing means and rules to protect the users' information against misuse by the private companies that store this data. In fact, data is considered an extremely valuable information in modern ICT infrastructures: many organisations have a significant interest in data harvesting and mining so that they can leverage this data to improve their reasoning processes and use them to solve problems. For instance, profiling of users' habits and interests is very profitable for online e-commerce sites so that they can suggest — or *trick* (depending on the perspective) — the users about purchasing more products. As data is exchanged on the communication networks, their resilience, security and privacy is paramount to protect the investments by organisations on the digital market. In fact, if networks were compromised by malicious users, the services and data distributed on them would be at risk, possibly causing severe economic losses for the digital businesses. Moreover, given that communication networks may be used for privacy-sensitive operations, such as *e-banking* transactions and electronic identification, their violation may cause significant damage to the end users themselves. For instance, leaks of credentials and credit card details by malicious users may lead to loss of personal funds and identity theft. Because of this, techniques to address security properties on the network infrastructure are needed. In this regard, integrity of data and services have to be guaranteed by the ISP, which deploys and maintains the networking equipment and communication links for the exchange of the users' traffic.

In the softwarised scenario, integrity of the network is even more critical, as the users' traffic is processed by software components that run in VNFs on top of untrusted commodity hardware, rather than specialised appliances. More specifically, the networking-specific software runs in a more generic execution environment along with other services and applications, hence the attack surface is increased by the possible vectors introduced by malfunctions or targeted attacks to these elements. Software bugs may be introduced by the VNF developer and not be detected by functional testing, harming the overall functionality of the network function and possibly damaging the users' data and communications. Moreover, compared to traditional networks, the NFV scenario suffers from threats specific to virtualisation as well. In fact, scientific literature has demonstrated that certain vulnerabilities can be exploited by malicious users to break the isolation enforced by the *hypervisor* on the virtual instance, allowing an attacker to gain access to virtual resources belonging to a different tenant, and even to gain root privileges of the hypervisor itself. The conjunct presence of both virtualisation and networking threats, along with the impact of software bugs on the software codebase, make the exploitation of the NFV paradigm in a production environment still a challenge from the ISP perspective. With respect to virtualisation technologies, the scientific literature targets lightweight forms of virtualisation as the most fitting solutions for the implementation of each VNF, as their stateless nature and smaller footprint make them highly scalable and suitable for the service composition needs of NFV.

1.4 Motivation of this work

The work described in this dissertation is motivated by the context that was just presented. More specifically, we believe that networks must be protected for integrity against manipulations by attackers and misbehave due to software bugs or incorrect configuration by administrators so that the network operation is not compromised and the privacy of the end users is not harmed. In this regard, scientific literature has proposed Remote Attestation (RA) as a viable solution to the problem. This is an integrity verification method that requires a *target* platform to record measurements about its own execution state so that they can be verified against known-good values by a trusted verifier. This method is suitable for different implementations, wherein both the measurement and the integrity verification scheme vary depending on the actual target. Traditional RA schemes consider a measure as a cryptographic hash of software executed by the target platform, so that the resulting digest is compared against a white-list of hashes that has been previously defined by the verifier. This scheme is known as *binary attestation*, and has been proposed as one of the key enablers of Trusted Computing (TC). This paradigm, as defined by the Trusted Computing Group (TCG), has proposed RA protocols based on the integrity verification of the whole software stack of the target platform, starting from the boot process up to the OS user space. Moreover, TCG has standardised dedicated hardware components that provide hardware-level protection for secrets owned by the target platform, so that the integrity measurements can be securely stored and protected by malicious users that have access to the physical system. In particular, several iterations of a cryptographic device, i.e. the Trusted Platform Module (TPM), have been produced by the TCG partners. This acts as a hardware Root of Trust (RoT) that offers secure storage, authentication credentials and cryptographic acceleration to effectively support the RA scheme. The TPM is commonly used for many applications such as trusted boot, full disk encryption (e.g. Microsoft BitLocker), and digital right management.

More recently, integrity measurements have targeted the behaviours of the programs executed on the target platform so that the control flow of their execution is attested against misbehaviours. Although promising, the latter approach does not easily scale to the protection of the whole target system, as it would require each program to include specialised instructions at the OS level so that their execution can be measured. Because of this, traditional TC patterns for RA are particularly relevant whenever the target platform needs to run a large and ever evolving software codebase, as traditional integrity measurements are not tailored to specific components or binaries. In this regard, the cloud nodes of an NFV platform are suitable targets for these mechanisms, as they are expected to run several software elements that may be implemented by many software developers. Nonetheless, binary attestation inherently suffers from limitations with respect to scalability in case of large white-lists. Moreover, the reference measurements must be kept updated by the infrastructure maintainer to avoid false positives. Finally, traditional RA schemes as proposed by TC do not easily translate to virtualised environments, as the hardware-level protection offered by the TPM is typically limited to a single owner (whereas virtual instances are many and may belong to different tenants). More recently, platform trust has been addressed by the proprietary technologies proposed by microprocessors' vendors such as Intel, ARM, and AMD. Each of these companies has embedded proper instructions and data structures in their CPU to implement a Trusted Execution Environment (TEE). This consists of a secure area for both data and code execution that offers hardware-level protection with respect to confidentiality and integrity. Attestation of the secure area is typically supported so that a verifier can ensure that its state has not been tampered with. This process leverages TEE capabilities and cryptographic keys that are protected by the hardware, making it a viable alternative to TPM-based attestation schemes. The commercial interest by hardware vendors in hardware platform trust makes it readily available in heterogeneous computing systems, although this requires the need to support non-uniform architectures from a verifier's perspective.

The final goal of this work is to propose remote attestation as an effective technology to secure a cloud NFV platform by demonstrating that (1) hardware platform trust can provide a higher degree of security in a privacy-sensitive infrastructures, such as NFV, when compared to software-based mechanisms; (2) different virtualisation and trust technologies should be addressed to make integrity verification a viable solution in heterogeneous scenarios; (3) the orchestration and reacting capabilities of NFV infrastructures can be significantly improved by platform trust assessment, even enabling novel use cases that revolve around cloud security services.

In order to achieve this goal, this research has focused on the design of protection mechanisms to secure the NFV platform, addressing both the need for support of heterogeneous cloud environments and application to novel forms of virtualisation. The initial contribution of this work to the state of the art in the field of NFV security is the design of an abstract trust monitoring architecture tailored for the softwarised networks and based on TC principles and technologies, i.e. the Trust Monitor. The key aspects of this solution, in contrast with the existing designs from literature, are the integration with the reference ETSI NFV framework, in particular within the administrative domain, and the generic approach to attestation that enables heterogenous TEEs to be adopted by the network operator. Starting from the abstract design, this work proposes a novel technique for attesting the integrity of software in a generic VNF that can be generalised to a generic cloud environment to provide a reliable, hardware-based protection mechanism for virtualised instances. This technique is based on open-source tools and Linux kernel capabilities, focusing on a specific TEE, i.e. the TPM, to protect both the measurements of the host and the containers. As final contribution, this work addresses the integration of the aforementioned technique in the abstract security architecture and the definition of specific processes to enrich the orchestration and management capabilities of the NFV platform.

A driving factor of the overall research work is to propose mechanisms that can be effectively implemented and integrated in a reference NFV framework, as they are grounded in technologies that are readily available in existing hardware architectures. In fact, although scientific research in this area has already addressed security mechanisms tailored for the NFV scenario, they either lack a strong binding to hardware protection mechanisms, such as the TPM, or they don't assume a generic approach to integrity mechanisms so that they could be applied to heterogeneous computing infrastructures. In this regard, a practical validation of the contributions of this research has been achieved in the scope of the SHIELD[1] Horizon 2020 European project. This proposes a universal solution for deploying virtualised security infrastructures into operator and corporate networks, leveraging platform integrity as a guarantee that the virtual resources are not tampered with by malicious users. The project started in September 2016 and ended with excellent results in February 2019.

1.5 Bibliographic foundation

The contributions that compose this research work have been originally discussed and validated in scientific publications that are available at the time of writing:

[1]SHIELD project website - `https://www.shield-h2020.eu`

- "NFV-based network protection: the SHIELD approach" [62], focusing on the early study of security requirements of an NFV platform with respect to the SHIELD architecture;
- "On the establishment of trust in the cloud-based ETSI NFV framework" [19], presenting an early design of our security monitoring architecture in the scope of the ETSI reference framework for NFV;
- "Integrity verification of Docker containers for a lightweight cloud environment" [18], that discusses in depth our novel technique for runtime integrity verification of software components running in containers;
- "A proposal for trust monitoring in a Network Functions Virtualisation Infrastructure" [17], that presents the final architecture of the Trust Monitor and the workflows that are in place to include integrity verification as part of the NFV administrative and orchestration domain.

1.6 Organisation of the thesis

The rest of the thesis is organised as follows: Chapter 2 analyses the scientific literature and industry standards wherein hardware platform trust and softwarised networks lay their foundation, as well as the open issues in addressing security and trust in the NFV scenario; Chapter 3 introduces our novel architecture of the trust monitoring entity tailored for the NFV scenario and also details its generic approach to attestation; Chapter 4 describes our novel technique for VNF attestation by presenting its requirements, the design and implementation along with its experimental evaluation; Chapter 5 demonstrates how our novel trust monitoring architecture, together with the aforementioned attestation technique, can be used to effectively support the NFV threat mitigation strategies against tampering attacks; Chapter 6 draws the conclusion about this work and envisions future activities in the field of hardware-backed integrity verification for the NFV scenario.

Acknowledgements

I would like to thank prof. Antonio Lioy for being a sincere mentor during my Ph.D activities, providing advice and direction of the work yet helping me to independently tackle the challenges in my research and drive the final outcomes.

I also thank the people in the Computer and Network Security lab who helped me to grow — both as researcher and as a person, which is more important to me

— during my time as Ph.D student. I fondly remember the time we have spent together.

I have to acknowledge the incredible opportunity I had to work on international projects together with talented researchers and to experience different places abroad, thanks to my involvement in such activities. In particular, I would like to thank all the partners in the SHIELD project for their significant efforts.

Chapter 2

Background

This chapter outlines the state of the art in the field of Network Functions Virtualisation security, with particular attention to trust assurance and active integrity monitoring. Because of this, a concise introduction over the NFV paradigm is presented first, along with a throughout discussion on the usage of lightweight forms of virtualisation in the context of NFV. Then, a security analysis of the paradigm is presented, followed by an outline of the proposals available in literature to address its weaknesses and a discussion on the limitations of existing approaches. Finally, the key concepts around platform trust assurance are detailed, as they are required to understand the technical aspects of the approach proposed in this dissertation.

2.1 The NFV paradigm

NFV [27] aims to overcome the limitations of existing networked infrastructures by leveraging standard virtualisation. Compared to traditional networks, hardware appliances are dropped in favour of a software-based design, made possible by the extensive use (and capabilities) of standard virtualisation in today's ICT platforms, as depicted in Figure 2.1.

Since 2012, the ETSI NFV Industry Specification Group (ISG) started its specification activities by defining the requirements of the paradigm and consolidating its architecture and target use cases. The most significant benefits that have been identified since the beginning were the reduced cost for network equipment and its power consumption due to consolidating equipment, the reduced time to market, more flexible multi-tenancy and multi-version support, targeted service definition and openness to the software market, when compared to vendor-specific fixed solutions [25]. In 2017, the ETSI NFV ISG had reached over 290 organisations in total, including 38 telecom operators [26].

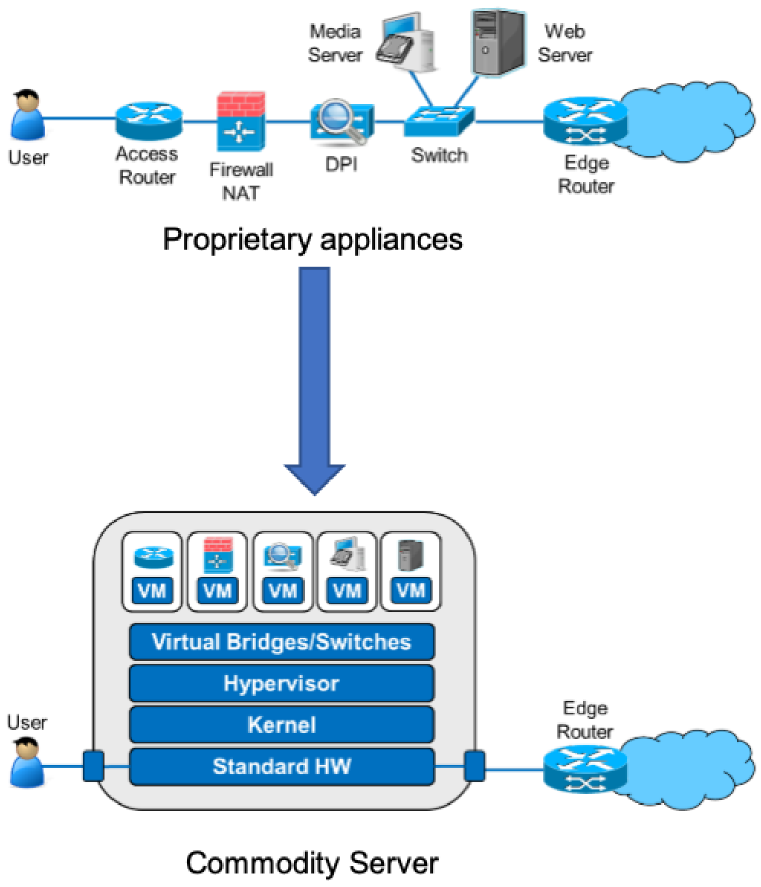

Figure 2.1: Transition from traditional networks to NFV

Given the momentum gained by this paradigm, in the last few years several ISPs have publicly announced their interest in the NFV technology, such as Telefonica Spain [57], AT&T [6] and Deutsche Telekom [65]. More recently, the paradigm has been sponsored by ETSI as one of the key enablers for the definition of the 5G networks, as these are expected to leverage the higher degree of flexibility ensured by softwarised networks. More specifically, the NFV concepts on VNF service composition can be leveraged to implement and manage the 5G *network slices*, i.e. heterogeneous network service layers that share the same physical infrastructure.

The main capability offered by NFV is the Management and Orchestration (MANO) of the virtualised resources for the provisioning and the life-cycle of network functions that are to be chained together into more complex network services. The VNFs [36] are deployed within a standard virtualised infrastructure, comprising several hardware compute nodes that share their computation, storage and networking resources among the *tenants*. This platform, i.e. the NFV Infrastructure (NFVI), is built upon commodity hardware resources and standard virtualisation

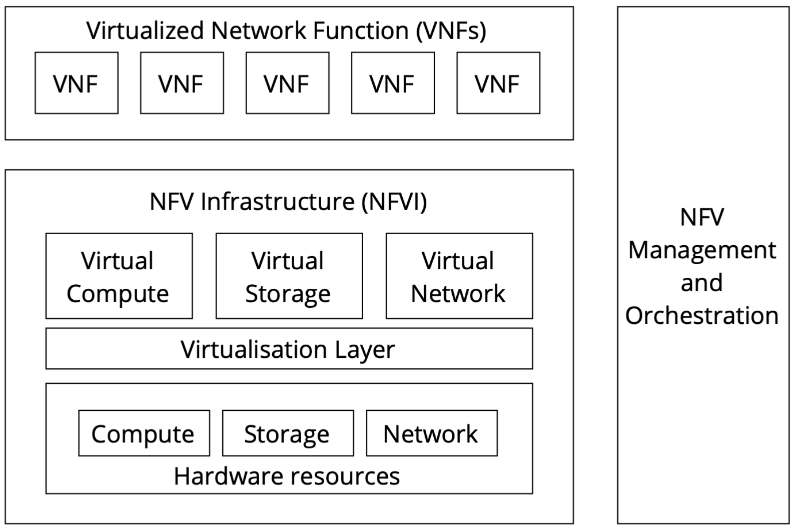

Figure 2.2: NFV high level architecture

technologies, such as OpenStack [72], Kubernetes [52] and VMware [95]. A high level view of the NFV architecture [29] is depicted in Figure 2.2. This clearly shows the separation between the operation environment, wherein the VNFs are executed, and the management plane where the resources are orchestrated. This is achieved by introducing a centralised entity, i.e. the Network Functions Virtualisation Orchestrator (NFVO), as a high-level controller of the virtual instances made available by the underlying physical infrastructure. The NFVO manages the life-cycle of VNFs and their composition into more complex network services by interacting with the VNF manager. This, in turn, issues commands to the Virtualised Infrastructure Manager (VIM), the software abstraction that manages the virtualisation domain. This approach is shared with other network softwarisation paradigms, such as SDN [84], that aims to separate the network forwarding plane from the control plane through software abstractions. Differently from traditional networks, the NFV ecosystem requires a cooperation between the ISP and the CSP, the latter being the organisation that provides the cloud infrastructure to host the instances of VNFs. Although the MANO functionalities are typically served by the same organisation serving the NFVI, effectively merging the ISP and CSP roles in the same organisation, alternative approaches may be possible in case of federated arrangements between telco providers.

The NFV paradigm is appealing for the operators to implement several novel use cases [28] wherein the network programmability can effectively overcome the limitations of traditional networks. First, the NFV infrastructure itself may be implemented by a large operator and offered as a service to its clients, i.e. smaller operators or enterprises, so that they can run their network service. In this regard, ETSI foresees the Security as a Service (SECaaS) approach as a novel approach to target cybercrime in the public networks, thanks to the creation of security services that leverage NFV technologies to address threats occurring in real time in the network so that the end users, i.e. the ISP customers, can be freed from acquiring and managing security equipment in their own premises. Moreover, VNF aims to virtualise the mobile core network — in the scope of 5G — to reduce the Radio Access Network (RAN) footprint and energy consumption thanks to smart and dynamic resource allocation and load balancing, in addition to faster time-to-market and operation. Other use cases involve virtualisation of the home network devices, so they can be programmed to offer heterogeneous services to the end users with reduced deployment and upgrade costs, and leveraging the NFV technology to dynamically deploy nodes of a virtual Content Delivery Network (CDN) to offer data (e.g. movies, music streams) as close as possible to the demand.

2.2 Exploitation of containers in NFV networks

Lightweight forms of virtualisation are gaining momentum in modern ICT infrastructures, and both the scientific literature and the ETSI specification efforts [30] in the field are addressing containers for the NFV platform.

As previously stated, containers propose to offer a more agile life-cycle management and service composition than standard VMs, as their main goal is to sandbox specific sets of applications rather than a complete system. Moreover, they are typically defined as stateless entities by declarative *descriptors* that allow them to be deployed on demand from scratch in heterogeneous virtualisation platforms, without the need of exporting large images and managing them in ad-hoc storage solutions. Their stateless nature makes containers more scalable than VMs as they can be dynamically replicated and decommissioned depending on the load of the infrastructure. Micro-service architectures benefit from containers as well, as their smaller memory footprint and modular approach better suits the composition of complex services than VMs. Different lightweight virtualisation architectures exist in literature, i.e. *process* (or application) containers and *machine* (or full system) containers, and are represented in Figure 2.3.

The first approach lets the virtual instance to sandbox a single application so that it is isolated from the other processes running on the host system, following the *one process per container* paradigm. This approach maximises the service composition capability of the architecture, as a complex service may be built from a series of applications chained together. Because of this, process containers are typically meant to be stateless, as their persistent data should be managed by an external database service (being it another container or a standard application). In case persistence is required in the instance, each process container can mount portions of the host file-system so that it is visible from inside the container instance.

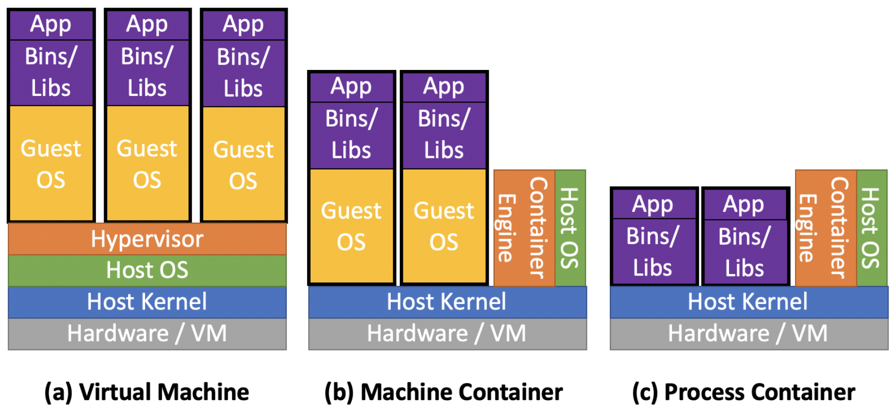

Figure 2.3: Lightweight virtualisation architectures

Machine containers, on the other hand, allow to sandbox a full OS user-space. Because of this, they are more similar in use to the standard VMs, as they are based on a stand-alone disk image and persistent file-system. They do not provide easier service composition than traditional approaches, as they must be built from a base image and they must be exported as disk images in order to be moved to another host.

Differently from VMs, both process and machine containers share the same kernel as the host system. In fact, they isolate the user space from the kernel by leveraging standard functionalities available in the Linux kernel, namely *namespaces* [61], control groups (or *cgroups* [60]) and *root capabilities* [59]. These represent the building blocks that are shared among the different container technologies [43], providing the basic mechanisms that allow each container to have a specific view of the execution environment that does not take into account other instances nor the underlying host platform. Namespaces represent abstractions of system resources that can be exposed to a virtualised instance as if it was the sole owner of such resource. To date, seven namespaces exist:

- *Inter Process Communication (IPC)*, which manages process-level message exchange;
- *Mount*, which manages the data volumes mounted by the instance;
- *Network*, which gives to each container an abstracted network stack and interface;
- *IPC*, which restricts the visibility of processes to the ones executed by the instance;
- *UNIX Time Sharing (UTS)*, which allows each container to have its separate hostname and domain name;

- *User*, which allows each container to be run by a separate user (so that it cannot interfere with other containers' or host's users);
- *cgroup*, which enables each container to have its separate configuration of control groups.

Cgroups allow to specify constraints for the usage of resources shared by all processes belonging to a separate group, such as CPU, memory, storage, network bandwidth. For instance, a system administrator could specify the maximum usage of memory for each process or pin a specific CPU to it (if more than one are available on the platform). Finally, kernel capabilities can be configured for each container so that it can have access to certain privileged resources, i.e. the host network stack, without being run in its entirely as a privileged process (i.e. as a super user). Each container runs by forking a process with proper namespaces, cgroups and kernel capabilities, so that it is isolated from the host system and other containers. The isolated process is then used as an *entry-point* to the container instance, in charge of running either the target application in case of process containers or a standard Linux *init* process (or very similar) in case of machine containers.

Docker [23], Podman [75] and Rkt [79] are well-known technologies that implement the process container paradigm, while Linux Containers [58] and Canonical LXD [63] are popular machine container alternatives. To date, Docker is considered as a de-facto standard for containerisation of processes [16], being largely utilised in both the scientific community and production environments. In fact, humongous CSPs such as Amazon Web Services, Microsoft Azure and Google Cloud Platform already support the Docker container runtime, allowing for easier integration of container-based workflows in public clouds. Moreover, the vastly popular OpenStack private cloud solution supports Docker [37, 101], along with the Google-backed Kubernetes cloud management engine, that is rapidly evolving as a de-facto standard in cloud systems. The success of Docker, when compared to other alternatives, is due to the efficient definition of each container (thanks to a declarative approach via a textual descriptor, i.e. the *dockerfile*) and its management (via a well documented and largely supported command line interface). Moreover, Docker offers a public registry of images, i.e. the Docker Hub, which allows to download pre-made images of well-known software (such as databases, web services, caches and even desktop applications).

Docker is discussed by ETSI as a viable solution for deploying lightweight forms of virtualisation on a Linux platform in the NFV ecosystem [30]. In this regard, the Open Source MANO (OSM) reference framework for NFV [70], whose development is backed by ETSI, offers built-in support for Docker containers along with OpenStack and other container management solutions. Nonetheless, both OSM and other MANO solutions (such as the university-driven Open Baton project [68] and the Linux foundation's OPNFV [73]) typically offer a better support for traditional forms of virtualisation. Hypervisor-based technologies are still perceived as more secure solutions, as they are more mature and they may provide more isolation when compared to containers. In fact, VMs encapsulate their own kernel, which may be completely different from the host's kernel. On the other hand, this approach has a larger overhead when compared to containers both in the image footprint and in the management efforts, as VMs may be slower to run and they must be stored by ad-hoc cloud image services. The overhead is particularly apparent if we consider a scenario where isolation of the kernel is not a primary goal. Application of lightweight virtualisation technologies to NFV has been broadly discussed in scientific literature as well. Anderson *et al.* [2] propose Docker as an enabling technology to effectively deploy and manage VNFs among different lightweight cloud environments, given its portability and small footprint. The authors also discuss the benefits introduced by the Docker image registry as a viable solution for sharing containers by the different VNF developers. Moreover, the Dockerfile itself is considered a powerful solution for the specification of a VNF package, as it abstracts the definition of the container from the underlying physical infrastructure. With respect to performance, Cziva *et al.* [14] state that the deployment time of container VNFs is reduced by 68% over a popular virtualisation technology, i.e. Kernel-based Virtual Machine (KVM) [56] on average, when tested within a VNF framework, i.e. Glasgow Network Functions (GLANF), that manages the life-cycle of complex network services. In another work, Cziva and Pezaros [15] also discuss the applicability of containers to the Edge Computing paradigm thanks to the GLANF framework, specifying that containers are the most suited technologies that would allow the operators to benefit from next generation mobile networks in the scope of 5G. M.Raho *et al.* [76] compare Docker with traditional hypervisor-based technologies, such as KVM and Xen [99], for the implementation of VNFs tailored for the ARM CPU architecture. The authors' results display that containers perform better in CPU-bound workloads and in networking interactions based on request and response by two peers. On the other hand, hypervisors typically perform better in disk input and output operations and TCP streaming, due to their caching mechanisms. Bonafiglia *et al.* [9] present a comparison between Docker and KVM in case of VNFs deployed on the same server running Open vSwitch [71], a widespread SDN solution that allows to define the forwarding rules of packets exchanged by virtual instances in a flexible way. The authors evaluate both approaches and show that Docker containers are well suited for VNFs in case of applications associated with a specific thread or process, as they perform similarly to KVM from the networking perspective and they also require less resources due to the sharing of the host kernel. The authors also note as a drawback that containers provide less isolation than VMs.

2.3 NFV security analysis

Given the extensive usages of NFV technologies within the telco operator domain, and considering the security and privacy implications of data that are exchanged on public networks by the end users, it is important to identify the potential security weaknesses inherent to the paradigm and to define countermeasures that can effectively mitigate such threats.

The NFV ISG group has addressed the potential areas of concern of the NFV architecture since its initial specifications, laying the foundation on the definition of protection mechanisms to address problems that are specific of this paradigm, when compared to traditional networks [31], along with scientific literature [38, 54, 53]. In this regard, the scientific community typically elaborates solutions that are based around the standards on security and trust proposed by the ETSI ISG. From a high-level perspective, the NFV platform can be subdivided into three major domains, each of which may be targeted by specific attacks: the NFVI, the MANO and the virtual Network Security Function (vNSF)s.

On the NFVI side, the network topology is critical because of the isolation required between the infrastructural nodes and the virtual instances running on top of them. In this regard, it is to be noted that both physical and logical segregation of topologies shall be considered, as cloud-based services typically rely on network protocols that logically separate different kinds of traffic, e.g. via Virtual Local Area Network (VLAN) or tunnelling. Because of this, the topology enforced on the NFVI should be validated so that unauthorised users cannot eavesdrop the traffic. In this regard, technologies such as SDN require additional validation, to ensure that an SDN controller has not been attacked. Lal *et al.* [54] discuss the disadvantages of the software-based approach to network, stating that it may lead to improper separation between the operator network and the virtual networks that are reserved to each tenant, with the risk of exposing details about the physical infrastructure to a malicious user that accessed the VNF instance. Another work [53] discusses the limitations of traffic security among current cloud providers, proposing that privacy-sensitive virtual instances, such as VNFs, should leverage communication channel protection mechanisms such as Virtual Private Network (VPN) to secure the packets shared among the network service against eavesdroppers. Moreover, integrity assurance of the hardware and software is paramount, as it is discussed in depth in this dissertation. In fact, it is both required that the network operator can trust the virtualisation platform sufficiently to run its VNFs and that the platform can ensure that VNFs are genuine. With respect to integrity assurance, TC-based attestation of a platform by means of a hardware RoT is a viable solution to ensure that the platform has not been tampered with during its life-cycle, as it will be discussed in depth in Section 2.5. Along with integrity preservation, resilience and management of bugs is important to not compromise end user data. Because of this, the NFVI shall handle crashing in a graceful way, clearing data and state information that is not relevant anymore in case of a component crash. This allows to reduce the amount of information that is exposed to an attacker in case he manages to crash a system by means of a targeted attack to it. In this regard, it is to be noted that the hypervisor is in charge of deleting the *dangling* references to virtual instances that have been terminated. Performance isolation is much more important on the NFVI when compared to a traditional hardware network platform, as the same physical resources are shared by many software appliances that may belong to different tenants, with the risk of introducing privacy violations. With respect to privacy, Authentication, Authorisation and Accounting (AAA) facilities in the NFVI platform must take into account the different users that have access to it: the physical infrastructure maintainers, that administrate the hardware resources of the cloud nodes; the network operator, that has full control over the virtual domain that is running on top of the physical infrastructure and can create virtual tenants for its clients; the tenant user, which has a limited view over a portion of the virtual infrastructure, comprising a series of network services composed by VNFs; the VNF developer, which implements and shares its VNF on the platform (if made available by the specific use case).

AAA management is particularly important in the MANO domain too [34]. in this regard, as MANO is considered as an enclosed system, all malicious users are by definition insider attackers, having legitimate access to the system and introducing possible vulnerabilities to it. Protection against insider threats is not trivial, as it must take into account both physical and logical access to the NFV services. In this regard, both integrity verification of MANO entities and active monitoring by means of intrusion detection systems represent countermeasures to the aforementioned vulnerabilities. Jaeger [50] defines an architecture, i.e. the NFV Security Orchestrator, as a focal point of the platform to monitor and gather security information about the deployment and configuration of network services in the NFVI. This is meant to interact with the MANO domain as an external, pluggable element that can provide on-demand security services to the platform administrator.

Finally, VNFs represent another significant attack vector as they are the front elements of the NFV network, directly managing the end users' traffic. Because of this, software bugs in the VNF code are critical as they could open back doors to malicious users. Depending on the virtualisation technology, weak isolation of the virtual instance may cause significant damage to other instances, even belonging to a different tenant, and to the underlying physical infrastructure as well. Lal *et al.* [54] state that the vulnerabilities associated to the weak isolation of virtual instances represents a major risk introduced by NFV when compared to a traditional network, as a malicious user could break into the hypervisor domain by leveraging a form of privilege escalation, harming both the physical infrastructure and the other tenants' instances. The VNF software package itself may be modified by an attacker, hence integrity verification of the image and its descriptors is important as well [35]. In this regard, software-based validation involving digital signatures is the base countermeasure to these attacks, although it implies that proper certificate management is put in place so that each VNF signature can be verified properly. A significant improvement over signature-based validation is represented once again by attestation, although existing approaches from literature are limited towards specific virtualisation technologies. Ravidas *et al.* [77] propose an architecture to bind the life-cycle management of the computing nodes in the NFV cloud platform together with integrity verification of the images and configurations utilised by the network services. Finally, vulnerabilities that are specific to standard networking and virtualisation technologies and are not altered by the virtualisation of network functions should be considered. For instance, incorrect topologies may lead to loops that delay the communication between VNFs.

2.4 Impact of containers on NFV security

With respect to the utilisation of lightweight virtualisation technologies in the NFV platform [30], additional aspects shall be considered from the security perspective. The U.S. National Institute of Standards and Technology (NIST) has developed a classification of the risks related to containers regardless of the particular technology [67], given the overall similarities between the available solutions from the technical point of view. Analyses of specific container technologies have been discussed in literature as well, mainly focusing on Docker due to its popularity [64, 11, 8].

The different areas of concern with respect to containers: the images, their distribution, the orchestration process, the runtime for the execution of virtualised instances. The scientific literature has broadly discussed the vulnerabilities of specific Docker container engine has been proposed in a work by Martin *et al.* [64]. As aforementioned, each container is deployed starting from a software image, i.e. a packaged set of applications and configurations, that is typically built from a base, minimal execution environment, i.e. the *base image*. The first risk related to images in the context of NFV is the possibility that the developer installed malicious software in order to run the VNF application. This is particularly critical in case of open-source VNF development, as there are almost unlimited software sources available on the web. Then, a software may become vulnerable over time because a vulnerability is discovered. In order to address the possible threats associated to images, both integrity verification (as already stated in Section 2.3 and a periodic refresh of the image shall be performed. In case of Docker, all containers that share the same image would carry the same vulnerability unless the image itself is explicitly re-built in the local container image service. Then, the way container images are distributed among different execution environments must be carefully considered. Similarly to a public VM image service, the container registry, i.e. Docker Hub, is used by thousands of developers to pull software images. Hence, the access to such registry should be secured by means of mutual authentication between the parties involved in the communication so to mitigate Man-in-the-Middle (MitM) attacks. Moreover, a monitoring process should be put in place to ensure that stale images are automatically removed, so to avoid that they run outdated software that may carry a serious vulnerability. Regarding image distribution and verification, Docker Content Trust architecture is recommended so that container developers are required to sign their images, although this introduces a complex management of cryptographic keys. The orchestration of container instances represent another significant element from the security perspective. In this regard, trustworthiness of the orchestrator should be verified along with the access policies, so that no malicious users can leverage it to gather access to the virtual instances. In the context of NFV, a container orchestration platform such as Kubernetes acts as a VIM, in charge of the execution of virtualised instances that are managed at a higher level by the NFV orchestrator, i.e. OSM. Finally, the container runtime should be verified against manipulations or misconfigurations, so that it can be effectively used only by authorised entities, such as the container orchestration platform. In this regard, poor configuration may also reduce the security of each container instance, granting root privileges via unnecessary kernel capabilities (Section 2.2). With respect to Docker, although the remote control of the container engine is made possible through a TCP socket, this should be secured against attackers by means of TLS secure channel and mutual authentication. This is due to the fact that the Docker daemon is run with root privileges, hence it could be used by an attacker to run a vulnerable image with root privileges and try to gain access to the physical host. Other container technologies, such as Podman, try to address this issue by not requiring a privileged daemon to run the container instances. As discussed in literature [64], privileged containers can expose the host system to attacks, hence they should not be adopted in a production environment. In this regard, limitations of the containers' resources — via cgroups — should be enabled on a per-container basis, given the expected load of the virtual instance.

Given the possible risks introduced by containers in an information-sensitive domain such as NFV, protection mechanisms should be put in place at infrastructure level to ensure that each element in the chain has not been manipulated: first, the image software must be checked for malware and out-to-date software; then, the container runtime on the host OS must be verified against manipulations that could lower the security level of all containers; finally, the underlying host OS must be checked against manipulations that could signify that an attacker has gained access to it (both by exploiting a container vulnerability or an external attack vector).

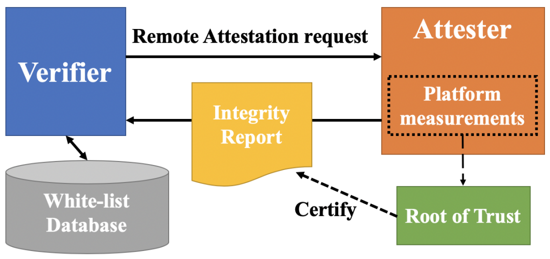

Figure 2.4: The Remote Attestation workflow

2.5 Trust assurance of softwarised networks

Trust assurance is a key element towards the exploitation of the NFV platform in production environments. In fact, both the network operators and their clients would benefit from trustable technologies that enable a secure validation of the execution environment. In this regard, hardware RoTs are typically perceived as more secure than software-based approaches, as they often provide tamper-resistance by design. This means that the cryptographic material used by the security process, i.e. a digital signature scheme, is stored in the hardware RoT and is protected against a series of attacks occurring at the physical level. Moreover, a hardware-based cryptographic device may have been certified by an external authority, as the TPM compliance to the FIPS 140-2 certification [90]. The ever increasing role of hardware-based trust is apparent in the consumer market as well, wherein the largest technology providers advertise the hardware-level security of their devices, as in the case of Apple with its T2 chip [4] or Google with the Titan M [42].

According to TC, the RA work-flow, as depicted in Figure 2.4, allows a Trusted Third-Party (TTP), also known as *Verifier*, to query the status of an *Attester* platform and compare the resulting Integrity Report (IR) [88] against reference well-known values that are previously stored in a Whitelist Database. The IR is certified by the RoT by means of cryptographic operations, such as digital signature. This allows the Verifier to detect tampering at boot level or even in software executed at runtime.

ETSI discusses trust in the NFV platform in several specifications [31, 32, 33], with particular attention to the TC methodology. In particular, the *Trustworthy Boot* process is introduced. This encompasses the technologies that allow the validation of the boot process of the following entities: hardware; hypervisor and NFV virtualisation platform; virtual instance wherein the VNF is deployed; VNF operating system; VNF application. The process may include features such as the Unified Extensible Firmware Interface (UEFI) *Secure Boot* [98] and the TC *Measured Boot* workflow [89].

The first allows to validate that the firmware and software run by the machine are trusted either by the platform manufacturer and/or by the platform owner by means of digital signatures. Each component in the boot process is verified by computing its signature and comparing it to a known-good value, either in software or by leveraging an available RoT. In case of an untrusted element of the chain, the boot process is aborted. The latter, on the contrary, aims to record information about the boot process by transitively compute measurements (i.e. cryptographic hashes) on both the firmware and software elements executed at boot and to store them in the TPM.

As briefly introduced before, the TPM is a discrete cryptographic coprocessor that is already available in commodity machines and can serve as an enabling technology for several security workflows. Since its beginning, the TPM has included basic functions such as key generation, secure storage and reporting. The first widely available TPM release was 1.1b, although the 1.2 version has been the most used for a number of years. To address the early privacy issues that arose from using a TPM, the 1.2 specification included a privacy Certification Authority (CA) to assess that a key was generated by a TPM without disclosing the TPM instance it came from. Later, new cryptographic protocols have been proposed in literature to enhance the privacy of TPM-based integrity verification, such as Direct Anonymous Attestation (DAA) [10]. Although being inexpensive, the TPM embeds several multi-purpose security features, described as follows. The Platform Configuration Register (PCR) is defined in the TPM memory to store the integrity of boot time measurements (e.g. BIOS, boot-loader). The TPM includes a number of PCRs, whose size depends on the supported hashing algorithm digest output length. These registers could be securely reported by an identity key generated by the TPM so that an external entity could verify the integrity of platform. The 1.2 release provided a standard interface to be implemented by all the vendors, which helped platform manufacturers to adopt the TPM device in their machines. A particular feature of the TPM is the way the PCRs are updated. In fact, they cannot be overwritten by arbitrary data, as the TPM cannot directly write in their memory. In particular, the TPM implements an *extend* operation, described as follows:

$$\text{PCRnew} = \text{SHA\_function}(\text{PCRold} \mid\mid \text{measured\_data}) \tag{2.1}$$

where *PCRold* is the value present in the register before the extend operation, *SHA_function* is the cryptographic hash function, || is the concatenation operator, and *measured_data* is the new measure to be inserted.

Moreover, the TPM should include some Non-Volatile Random Access Memory (NVRAM) space that can store any type of data. A limited portion of the TPM is usually used to store the TPM's vendor certificate of the TPM Endorsement Key (EK) [86]. Up to, and including, the 1.2 release, the TPM specification only supported the RSA, AES and SHA-1 cryptographic algorithms. The TPM supports secure storage by *sealing* data to a cryptographic storage key so that it can be decrypted only in case some conditions are met (e.g. the PCRs have predefined values). The latest TPM 2.0 specifications [91, 92, 93] deeply differ from the previous version, as TCG has redefined the internal structures, commands and algorithms (particularly cryptographic agility to allow for multiple ciphers, such as SHA-2 family and Elliptic Curves Cryptography (ECC)) with respect to the previous chip iteration. Nevertheless, the security capabilities of the device have been kept intact and extended with new features. The TPM implements the Root of Trust for Storage (RTS) and Root of Trust for Reporting (RTR) fundamental services that are required in any TC-compliant Trusted Platform (TP), i.e. the computing platform that has a trusted component as a foundation of trust for software processes [89]. In addition to these, each TP requires a Root of Trust for Measurements (RTM), i.e. a computing engine that can make reliable integrity measurements about the platform status. The standard RA scheme proposed by the Measured Boot process requires the target host to run a particular instruction set, i.e. the Core Root of Trust for Measurements (CRTM), that may be stored within the BIOS and starts the transitive trust process. Then, the measurements are included in an IR that is signed by a cryptographic key whose private part never leaves the TPM secure storage, i.e. the Attestation Key (AK), and is presented to an external verifier. Compared to Secure Boot, this allows the external verifier to both ensure that the target machine was not tampered with and to notify external entities in case a manipulation was detected. Moreover, Secure Boot requires the platform administrator to properly update the UEFI configuration on all the target machines in case their software changes, i.e. because of an OS update. This is particularly apparent in case of some Linux distributions, which may not have associated a UEFI cryptographic key, hence they are practically unable to boot. Finally, Measured Boot allows the verifier to have a more flexible approach to integrity verification. In fact, it could decide whether to disallow the target platform to boot, or to still boot with restricted access to other entities and privileges (or flag for investigation).

Apart from the boot phase, a computing platform would require periodic monitoring of its runtime execution to ensure that its software is not corrupted at later stages. In this regard, Sailer *et al.* proposed Integrity Measurement Architecture (IMA) [80] as a practical solution to record measurements about software events happening during the life-cycle of the computing platform. As described by the authors, IMA can benefit from an available hardware RoT such as the TPM to securely store such measurements in its secure storage, and to provide them to a TTP for continuous verification. IMA has been implemented in the Linux kernel as a standard module that keeps track of each software event executed in a platform (e.g. a binary that has been run, or a file opened for read) and securely stores its measurement, i.e. a cryptographic hash computed over the software event, in a PCR. IMA is readily available in many distributions both in the consumer and in the enterprise markets. Although widely popular, TC-based integrity verification does not easily translate to a generic virtualised domain by keeping the same security assurance than a physical domain. This is due to the nature of the TPM, which is limited to a single platform owner and hence cannot be shared among several virtual instances.

Application of Trusted Computing mechanisms to the softwarised network environment, comprising both SDN and NFV, has been discussed in the work by Jacquin *et al.* [49] that propose RA to be implemented by the NFVI, individual VNFs and the MANO sub-systems. More specifically, the authors discuss the challenges in Virtual Machine attestation and propose IMA as a practical implementation of a run-time integrity measurement architecture. The work by Yan, Zhang and Vasilakos [100] proposes an NFVI Trust Platform middleware to be embedded in the virtualised infrastructure by an authorised party, e.g. the ISP. This acts at the virtualisation layer and it leverages a hardware RoT, so that executed components are measured in a secure way. In this context, the TPM is referenced as an example implementation of RoT that is readily available in several hardware architectures. Faynberg and Goeringer [38] describe the establishment of a hardware-based RoT among the execution components of the NFVI, and propose Remote Attestation as part of a security monitoring and management component in the NFV environment. Schear *et al.* [81] have proposed a solution based on TPM, IMA and the Xen hypervisor that allows to attest a private cloud environment hosting VMs equipped with a virtual Trusted Platform Module (vTPM), i.e. a software implementation of a TPM. Its original design was developed by Berger *et al.* [7], focusing on the 1.2 specification and the Xen hypervisor. The authors proposed that a vTPM manager application, running in a Xen privileged VM, served as an access broker between the underlying physical TPM (pTPM) and the different vTPM instances. The direct access to the pTPM by the vTPM manager is provided by the Xen hypervisor. In this scheme, the vTPM manager manages the vTPMs persistent data (including their cryptographic keys) in the privileged domain and seals them with pTPM storage keys. Wan *et al.* [96] discuss the limitations of traditional TPM design when applied to a virtualised environment. In fact, the TPM is originally designed to support a single host and cannot handle simultaneous access by multiple entities. The authors discuss alternative designs for vTPMs in literature, including the original proposal [7] and an alternative approach based on para-virtualisation of pTPM [24]. Compared to previous solutions, the latter tries to overcome the security issues of fully virtualised TPMs when exposed to VMs, although it does not overcome the limitations in current TPM design.

Alternative proposals to Trusted Computing that focus on hardware-based methods to secure the NFV platform exist in literature. TEEs such as Intel Software Guard Extensions (SGX) [48], AMD Secure Encrypted Virtualisation (SEV) [1] and ARM TrustZone (TZ) [5] have been proposed as a solution to secure software run in virtualised instances. Wang *et al.* [97] have proposed protection of a virtualised instance of TPM 2.0 (based on the open-source *libtpms* library) in a SGX enclave, a shielded memory area wherein only trusted code is allowed. The work by Shih *et al.* [83] proposes a protection scheme, named S-NFV, that aims to isolate the states of NFV applications (i.e. the VNFs) via SGX. States are internal data of the VNF that are leveraged to enable cross-packet and cross-flow analysis. The authors propose the split of each NFV application in a S-NFV enclave and a host processing part. The S-NFV enclave includes the states of the VNF, while the rest of the processing is done outside of it. The communication between the trusted and untrusted parts of the VNF should be minimal, as it can deeply affect the performance of the solution. The authors also propose the SGX RA feature to monitor S-NFV applications remotely, although this approach requires the ISP to implicitly trust the Intel attestation service. Poddar *et al.* [74] also discuss application of SGX to the NFV platform as part of the SafeBricks, a system that shields VNFs from each other and from the untrusted CSP. In this platform, each VNF is run in a multi-threaded SGX enclave, hence it inherits the limitations in memory size, the impossibility of run system calls and to create timestamps, and the vulnerability to certain side-channel attacks. Nonetheless, this approach enables flexible remote attestation at VNF level, which has a lower impact on performance than TPM-based Measured Boot. Lefebvre *et al.* [55] discuss the blocking factors on integration of TEEs within the NFV and SDN domains, and propose an universal abstraction layer to overcome these issues. Moreover, the authors describe AMD SEV and Intel SGX as most preferred candidates for trust management in softwarised infrastructures, as they enable attestation without the need of measuring the entire platform. In this regard, the authors criticise the TPM-based integrity verification as it only supports a static RoT and requires the whole system to be measured, making its use not practical in a cloud setting. Coughlin, Keller and Wustrow [13] also discuss the usability of SGX in arbitrary NFV applications and propose an extension of the Click modular router to perform secure packet processing in SGX. Although promising, SGX requires end-user applications (e.g VNFs) to be aware of the enclave mechanism, as their code has to be designed to interact with an enclave (in case of the VNF untrusted part) or to export proper method calls from the enclave itself (in case of the VNF trusted code).

2.6 Open issues

The scientific literature has addressed the security and trust of an NFV platform broadly, as presented in the previous sections. This is mainly due to the ever increasing interest in the paradigm both by the research and development community and by the telco operators. Nonetheless, there are several open issues that are still not entirely tackled by the literature, and they represent the motivation for the research work that is being discussed in this dissertation.

Existing approaches lack an integrity verification solution that is generic enough to suit different execution environments, hence it allows remote attestation of cloud nodes equipped with heterogeneous TEEs and even software-only techniques, as in resource-constrained environments such as IoT fleets.

In addition, existing approaches consider NFV security architectures as elements that are external to the existing platform. In contrast to this approaches, the proposal of this work is to tightly couple the trust assurance mechanisms within the life-cycle of the NFV platform, with particular attention to the NFVI and the VNFs. Because of this, this work aims to integrate the proposed solution within the MANO stack, similarly to the Trust Manager defined by ETSI [32].

Hardware-based platform trust for virtualised instances is a major limitation of existing approaches to cloud security, as discussed in Section 2.5. In this regard, we believe that a security mechanism tailored for the NFV environment must ensure that both the physical and the virtual infrastructures are checked against manipulations by attackers. In this regard, the limitations of TC technologies when applied to a virtual environment are to be addressed by the research.

Finally, the impact on performance and scalability of trust assessment must be addressed as well. This is already important in an environment wherein several virtual instances are sharing the same physical infrastructure, and it becomes crucial in the networking domain, wherein the latency and load balancing are key performance indicators.

In the next sections, a proposal for a security and trust monitoring solution tailored for the NFV environment will be discussed in depth, i.e. the Trust Monitor. First, the architecture design will be presented so to clarify the main requirements that have been defined, along with the components that are expected to implement these requirements. Then, a novel technique for container VNF attestation will be presented, along with its experimental evaluation in a lab environment. Finally, the integration of the proposed architecture and VNF attestation technique will be presented, as a practical approach to NFV threat mitigation. This is made possible by the deep interaction between the Trust Monitor and the MANO domain, which allows the infrastructure to timely react in case of manipulations to both the computing nodes and the VNFs that are deployed on them.

Chapter 3

Architecture

This chapter details the novel architecture for centralised integrity verification of an heterogeneous computing platform, comprising both physical and virtualised instances, i.e. the Trust Monitor. The overall goal of the design is to theoretically support any cloud-based ICT infrastructure as it abstracts from the practical use of the virtualised resources. Nonetheless, the softwarised network scenario, as it was discussed in depth in Chapter 2, is referenced in this chapter as a practical use case for the proposed architecture. NFV requirements on the security management process and life-cycle of virtual resources are considered for the design, as they fit into a more general demand for security of cloud architectures.

Compared to existing literature, the designed architecture does not fit a single use case and a specific integrity verification technique. On the contrary, it aims to define a generic trust framework that can be specialised to specific cloud use cases, such as NFV, regardless of the underlying technologies. The design work has been driven first by the definition of target use cases specific of cloud-based architectures. Then, we have derived a series of requirements (both functional and non-functional) from such use cases. Finally, we have defined an abstract design to comply to the requirements, and later developed a software architecture composed of several building blocks, detailed as follows.

3.1 Target use cases

The work described in this dissertation focuses on different use cases that are enabled by the creation of a security architecture tailored for the NFV cloud scenario. These differ by the deployment model adopted by the CSP, and they are not specific to a pre-defined cloud service model. In fact, the approach proposed in this work is meant to be transparent to the services that are offered by the CSP to its end users. They are all significative for the NFV scenario, wherein the telco operator is typically in charge of the maintenance of the physical infrastructure and leverages it to offer softwarised network and security services to its clients.

3.1.1 Security-as-a-Service

CSPs may leverage the SECaaS paradigm when offering novel security services to their clients. Compared to in-house deployments, SECaaS allows the complexity of the security analysis to be hidden from the client, who can therefore be freed from the need to acquire, deploy, manage and upgrade specialised equipment. In this use case, the CSP would be able to insert new security-oriented functionalities directly into the network of the customer, through its provided gateway or in the core infrastructure. In order to fully exploit SECaaS, the CSP should be able to ensure that the platform has not been tampered with, so that the security services are protected as wall. Hence, both the CSP and its clients would benefit from practical means to asses the trustworthiness of their resources.

SECaaS is detailed by ETSI [28] as one of the key use cases that are enabled by NFV, as the flexibility and dynamic resource allocation introduced by the paradigm can effectively support the needs of on-demand protection as envisioned by it. This use case is motivated by the need of protecting ICT infrastructures effectively in an evolving threat environment, where vNSFs can be exploited to both monitor and react upon detected attacks. An ISP could build SECaaS services to secure his clients' networks, freeing them from the costs of managing, operating and upgrading dedicated network and security devices. In this scenario, big data analytics can play a significant role to fulfil the anomaly detection and define a mitigation strategy.

A high-level architecture of the SECaaS use case is depicted in Figure 3.1. It includes the following components:

- **Data Analysis and Remediation Engine (DARE).** It performs threat detection using analytics, cognitive intelligence and monitoring of the infrastructure, in order to define a mitigation strategy for an attack.
- **Monitoring vNSF.** It monitors the traffic of the network, acting as a network probe, event generator or honeypot. The relevant information of monitored traffic is fed to the DARE.
- **Reaction vNSF.** It applies a mitigation strategy defined by the DARE, with the aim of preventing or stopping a threat.
- **vNSF store.** A catalogue of vNSFs that can be instantiated in the network.

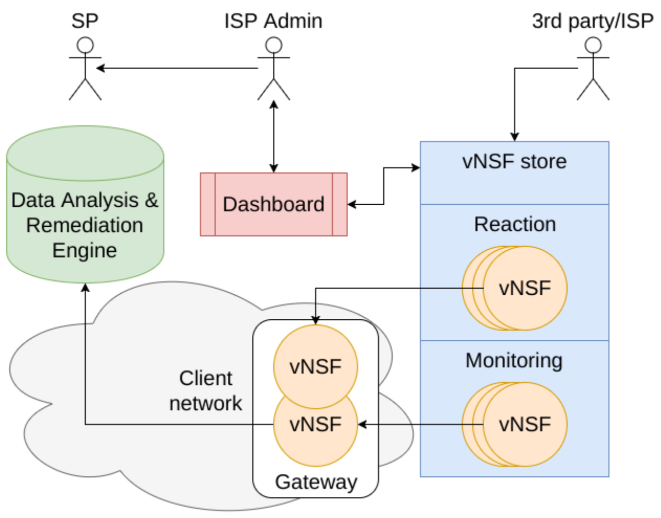

Figure 3.1: The Security-as-a-Service use case in NFV

- **Dashboard.** A visualisation component that can be accessed by the infrastructure administrator, to display the analytics result and recommend mitigations for incoming threats.

The vNSFs can be deployed onto the client gateway or in the ISP network infrastructure. In addition to these components, the SECaaS use case also specifies vNSF and infrastructure attestation as a necessary step to ensure that the SECaaS service is trustworthy. In this scenario, the design and development of a container-based vNSF should take into account the need for providing monitoring and/or reaction capabilities for specific security threats, in addition to reporting the relevant network and security information to trusted third-parties for analytics.

The SECaaS use case introduces relevant challenges in terms of correctness, availability and scalability of the vNSFs, given their critical role in a ISP infrastructure. These requirements are typically implemented in traditional deployments of security middle-boxes, hence they should be considered by the NFV paradigm as well.

3.1.2 Protection of the CSP infrastructure

In order to protect their own infrastructure, NFV telco operators usually have to deploy specific hardware that is very expensive, since this hardware has to be maintained by specialised operators. Furthermore, the operators may need to initially invest time to figure out the attack before being able to stop it. In this use case, the virtualisation offered by the cloud paradigm aims at dramatically reducing both costs by replacing specific hardware for virtualised instances, as well as providing centralised interfaces to present the implications of the gathered data, its analysis, and then act in the infrastructure. Because of this, integrity verification of the core infrastructure wherein the cloud services are deployed is paramount for the telco operator.

3.1.3 Security information and event management

Any telco provider requires to deploy a Security Information and Event Management (SIEM) to gather security informations about his platform. Often, it also require ways of sharing threat information with third-parties who wish to synchronise information and research on measures to be taken on recent attacks, suffered by others. Currently, if a Cybersecurity agency wants to retrieve statistical information about a network, it has to agree with the ISP to deploy specific hardware on the infrastructure. This is a very costly procedure — both in terms of time and money — that makes it prohibitive in the current market situation. Particularly, attacks are constantly evolving and require a fast, reactive and flexible solution. Using NFV and trustworthy security services, Cybersecurity agencies can establish agreements with the ISP and deploy VNFs quickly and without specific hardware in the infrastructure.

3.2 Requirements

The requirements detailed in this section have been defined as a consequence of the use cases proposed in Section 3.1, in addition to the analysis of the scientific literature and of specifications in the field of cloud security, as done by ETSI in case of NFV.

Security. The architecture has to follow the principles on attestation as detailed in [12], so that it does not incur in security weaknesses that have been already discussed and overcome in literature. In this regard, the platform should support both on-demand and periodic integrity verification. Moreover, the architecture must be designed by ensuring that AAA principles are implemented on the interfaces that allow its interaction with external entities and its actors, i.e. the CSP.

Privacy. The information gathered on the attested platform should not allow tracking of end users of the system by storing their private information (i.e. IP addresses, as in the case of VNFs that forward users' traffic in a telco network). With respect to interactions of the platform with external actors and systems, communications should be secured against eavesdroppers by means of well-known communication security technologies, such as TLS.

Expandability. The system must be able to support heterogeneous attestation workflows, depending on the available security technology available on the target platform. Moreover, the system should manage remote integrity verification of target platforms with a variable level of assurance, according to the desired attestation technique. For instance, a core system deployed in the internal network may be verified for boot-time manipulation only, while the systems that are directly exposed to the internal network may be subject to a run-time inspection. In this regard, it is important that the specific RA workflows can be scheduled independently from each other.

Interoperability. The system should be designed in a way that its notification and reporting capabilities are separate from the core integrity verification process, so that it can be easily extended to push information about the trustworthiness of the target platforms to different actors. Hence, it is important to define a well-known communication protocol and data format for the exchange of integrity reports.

Scalability. The server-side part of the system must be able to scale horizontally to support attestation of multiple target platforms in parallel. With respect to the client part of the RA process, each target node is expected to host a single Attester.

Traceability. The system must log information about the attestation requests issued to the different target nodes, allowing a trusted party (i.e. a cyber-security agency) to process such information in case of off-line forensic analysis.

Availability. The system should be able to recover in case of software failures so that a single failure does not compromise its life-cycle within the cloud infrastructure.

Compliance to international standards. In order to be effectively considered as a close-to-market solution for adoption in a production-level NFV environment, the system must comply to international standards in the field of security and privacy.

3.3 Design overview

This section introduce the trust architecture designed as part of the research work. The assumptions that motivate the design are presented, along with a high-level description of the architectural building blocks and a generic cloud attestation workflow.

Assumptions. The target use case is a cloud service model, i.e. IaaS, wherein the components that provide the end users' with virtual resources to be exploited by

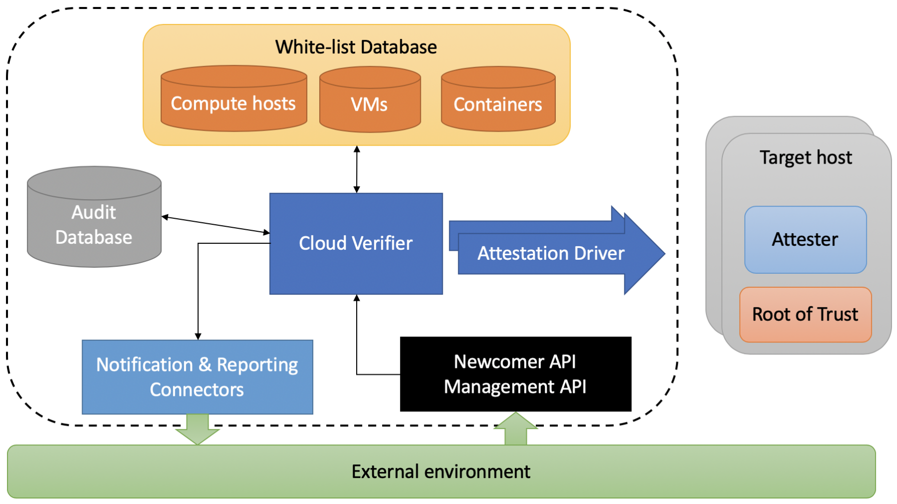

Figure 3.2: The Trust Monitor high-level architecture

cloud native services and applications are interconnected in a Local Area Network (LAN). This assumption is motivated by the exploitation of the remote attestation process, as each of the software elements that are subject to verification should be reachable via standard network protocols by a remote TTP living in the administrative domain of the platform. More specifically, following traditional networking models exploited by wide-spread cloud management solutions (e.g. OpenStack [72]), the remote attestation process should take place on the management network that is internally managed by the CSP, and which is separated from the provider external network by means of network isolation mechanisms (e.g. VLAN tunnels). Moreover, the proposed architecture leverages a client-server architecture wherein the Verifier, acting as a server, is responsible of requesting a proof of integrity to each Attester, i.e. a client, so that any unexpected malfunction at the client side can be detected in a timely manner by leveraging periodic polling. Moreover, this assumption enables the capability by the Verifier to request attestations for any Attester at a given time, on a on-demand basis (e.g. in case a critical operation was to be performed on the target node by the cloud infrastructure manager depending on the result of attestation). Finally, the software components that are executed in the administrative domain are implicitly trusted in the proposed design, so that the overall integrity verification process can be both simplified and targeted to the most exposed software components of the target infrastructure, i.e. the processes that are directly exposed on the provider public network.

Figure 3.2 depicts a high-level view of the proposed design, i.e. the *Trust Monitor*. This is composed of several elements, which can be grouped in the following areas:

- the core integrity verification logic, which orchestrates the RA workflow as a server-side entity and, hence, has internal knowledge over the target Attesters and their specific integrity verification scheme;

- the generic RA client-server model, which allows to remotely request a proof of integrity to heterogeneous target nodes, i.e. implementing different TEEs or software-based integrity mechanisms;
- the notification and reporting logic, which encompasses the modules that are either used to timely inform external entities about failed attestations or to store logs about the ongoing RA processes for further analysis.

The *Cloud Verifier* is the central element of the core integrity verification logic, in charge of initiating RA workflows depending on the supported scheme by each target node, being it a physical or virtualised instance. Moreover, this component is in charge of the verification process, which requires it to validate the proof of integrity, i.e. the IR, of the target node according to the specified RA scheme. It supports both on-demand and periodic attestation processes thanks to its cooperation with the *Scheduler*. This is a long-lived module that triggers the Cloud Verifier for periodic trust assurance of the target nodes, and is external to the verification logic. Finally, the Trust Monitor leverages a *Whitelist Database* as a centralised entity responsible for the long term storage of integrity measurements, so that they can be compared against those retrieved at run-time from the target nodes.

The client-server attestation process is modelled by an *Attestation Driver* and the target node, i.e. the *Attester*. This represents the server-side Verifier for a specific RA workflow, allowing the Trust Monitor to support Attesters based on different architectures (e.g. ARM, Intel) and RoTs. Each Attestation Driver is proposed as a plug-in to the central Cloud Verifier, and can be selected according to the target node. Software-based approaches are supported alongside hardware solutions, although they may provide lesser assurance about the integrity proof in principle.

With respect to the notification capability, the proposed architecture encompasses interfaces towards external modules so that it can either *send* information about the trustworthiness of the platform or *receive* commands regarding its functionalities. The first type of interaction is defined as a *Connector*, i.e. a software module that connects to an external service via its Application Programming Interface (API). Several Connectors can be implemented depending on the target environment. The second interaction is made available via APIs that are directly exposed by the Trust Monitor, i.e. the *Management API* and the *Newcomer API*: the first serves requests about the status of the Trust Monitor and of the trustworthiness of the target system; the latter allows registration and attestation of newcomer nodes in the platform. Finally, the reporting capability of the Trust Monitor is made available via an *Audit Database*, a centralised storage of integrity reports that can be leveraged by external entities (i.e. cybersecurity agencies) to retrieve historical information about the trustworthiness of any attested node.

The architecture that is proposed in this dissertation can fit different use cases and target platforms, such as NFV, by ensuring that the following operations are performed:

- develop the desired Attestation Drivers that fit the target Attesters;
- develop the proper Connectors to integrate the architecture within the target environment (e.g. to interconnect the Trust Monitor with the NFV Orchestrator, in case of NFV);
- populate the Whitelist Database with measurements that are relevant to the integrity verification process.

3.4 Process

The purpose of the Trust Monitor is to assess the trustworthiness of the nodes composing the target cloud infrastructure, in order to act on compromised nodes (e.g. exclusion from the platform) and validate the integrity state of newcomers. In order to do so, the Trust Monitor should be able to interact and cooperate with other components of the cloud platform, as it is expected to leverage existing information about the target platform that is already available in the administrative domain. A high-level description of the steps that are envisioned for the complete integrity verification process is depicted in Figure 3.2, and is hereby presented to the reader.

1. The cloud infrastructure manager registers a node to the Trust Monitor via its Newcomer API, providing the proper parameters to allow its further attestation (e.g. the proposed Attestation Driver, its hostname and/or IP address, its hardware architecture). Moreover, the manager defines the proper reference measurements for the target node and stores them in the Whitelist Database.
2. The Cloud Verifier initiates the RA process for the target node. This may happen in two different phases:
    - on-demand request via the Management API;
    - periodic request via the Scheduler.

3. The server-side logic of the Attestation Driver is instantiated, according to its original definition for the target node, so that it can request an integrity proof to the target Attester. This typically involves sending a nonce for the freshness of the response.
4. The Attester responds to the query by providing a proof of integrity that is ideally secured by a verifiable RoT.
5. The Attestation Driver validates the response and extracts software measurements from the integrity proof, forwarding them to the Cloud Verifier.
6. The Cloud Verifier verifies the measurements against the reference list provided in the Whitelist Database, storing the outcome of the process in the Audit Database.
7. The Connector presents the result of attestation to the cloud infrastructure manager and/or to a cloud administrative service for further operation (e.g. exclusion of the node in case of untrusted proof).

The process does not specify the actual RA technology supported by the target cloud node, as this strictly depends on the Attestation Driver chosen by the platform owner. Because of this, our design suits different integrity verification mechanisms, and it does not require an implementation based on TC technologies only.

The following sections give a detailed description of the features of the architecture's components.

3.5 Cloud Verifier

The Cloud Verifier is the central component of the Trust Monitor, responsible for the execution of the integrity verification process as a centralised entity. Because of this, it can be considered as the *workflow manager* of the architecture, as it is responsible to orchestrate interaction with the other components (e.g. the Whitelist Database, the Connector, the Audit Database) during its life-cycle. It manages the following functionalities:

- registration of a node to the *trusted domain*, i.e. the list of cloud nodes that are verified for integrity;
- instantiation of the Attestation Driver for each node in the trusted domain when attestation is requested;
- aggregation of integrity proofs determined by each Attestation Driver;

Attribute	**Type**	**Description**
`address`	String (16)	IPv4 address of the target
`analysis_driver`	String	Identifier of the Attestation Driver
`host_name`	String	Target hostname
`operating_system`	String	Operating System of the target node
`analysis_type`	String	Type selector for the Attestation Driver
`analysis_init_data`	Dictionary	Initial data for the Attestation Driver
`virt_technology`	String	Virtualisation technology

Table 3.1: Cloud Verifier host registration table.

- trigger notifications based on the result of attestation;
- store the attestation log for audit.

The registration phase is needed to properly setup the attestation process with each target node composing the infrastructure. The information required at this stage is detailed in Table 3.1. At minimum, the Cloud Verifier requires the IP v4 address of the target node and the identifier of the Attestation Driver, which must be uniquely defined in the Trust Monitor configuration. In addition, the user may specify the hostname of the node, if available, so that it can be used alternatively to the IP address during the attestation phase. This feature is particularly relevant in a cloud environment, wherein the physical location of an instance may vary over time because of migration of re-location in another cluster, hence using its IP address may not be as effective as relying on the DNS name resolution. At registration, the user may also specify the OS adopted for the target node. This is relevant in case of run-time integrity verification of software packages executed in the target machine, as they would be specific to a certain OS. In that case, the Whitelist Database may be previously populated with the known-good software for the specific system, allowing easier integration of nodes that share the same configuration. Then, the registration may include additional information to trigger a specific integrity verification routine in case the Attestation Driver allowed for different solutions (e.g. boot-time versus run-time checks), and an opaque *data blob* for initial data that may be required by the Attestation Driver at each attestation (specific to the RA workflow). Finally, the node virtualisation technology may be required in case of compute hosts, so that the Attestation Driver can apply specific verification logic depending on the hypervisor architecture.

With respect to attestation, the Cloud Verifier does not directly interact with each Attester that was previously registered. When prompted, it cycles through the list of registered nodes and instantiates the proper Attestation Driver by leveraging the identifier as defined in Table 3.1. In order to improve performance, each Attestation Driver is executed independently from each other by leveraging threading at

Attribute	**Type**	**Description**
`attestation_time`	Timestamp	Instant of the attestation request
`trust_level`	Boolean	Identifier of the Attestation Driver
`host_ir_list`	List of host integrity reports	List of Integrity Reports, one for each Attestation Driver

Table 3.2: Cloud Verifier global attestation status data.

Attribute	**Type**	**Description**
`node_name`	String	Name of the target node
`trust_level`	Boolean	Trustworthiness of the target node
`analysis_status`	Integer	Is different from zero in case of failure by the Attestation Driver
`analysis_extra_info`	Dictionary	Key-value pairs of additional data regarding attestation (e.g. list of untrusted binaries)
`analysis_driver`	String	Identifier of the Attestation Driver
`analysis_time`	Timestamp	Instant of attestation request
`virt_ir_list`	List of virtual integrity reports	List of Integrity Reports for each virtual instance run on the physical node
`host_remediation`	Object	Object that suggests a remediation option for the node in case it is found untrusted

Table 3.3: Cloud Verifier host integrity report data.

application level. In this regard, the Cloud Verifier manages a pool of threads by assigning to each of them a separate Attestation Driver instance, and then waits for all of them to terminate.

Although more efficient, parallelisation requires all the threads to agree on a common data format for the exchange of information with the main process. Hence, we have specified a proper data format for the exchange of information about the trustworthiness of each node. As reported in Table 3.2, the Cloud Verifier identifies each integrity verification of the whole target platform with the request timestamp, the global trust status and a list of the integrity reports, one for each host that was attested. Each of the Attestation Drivers is bound to return an integrity report that complies to this interface, as the Cloud Verifier is responsible to derive the global trust level of the infrastructure by analysing the content of each integrity report. Table 3.3 shows the full list of parameters that are specific to each host IR. They include the name of the target node (which defaults to its IP address

Column	**Type**	**Description**
`isolation`	Boolean	If true, the node shall be isolated from the network
`termination`	Boolean	If true, the node shall be terminated by either executing a command manually or via the hypervisor.

Table 3.4: Cloud Verifier remediation data.

in case no hostname was provided), its trustworthiness and various details about the actual attestation process. For instance, the Attestation Driver may signal that there was a software error that made the attestation unsuccessful, or it can provide details about the result (e.g. the list of software measurements that were not white-listed). Moreover, each host attestation is bound to a timestamp and a list of IRs belonging to the virtual instances (e.g. containers) that are verified along with the physical platform. Integrity verification of the virtual instances may be performed according to the defined Attestation Driver, and it may imply either static validation of the image or some form of TC-compliant RA scheme. In this regard, limitations on attestation of virtualised environments apply. Finally, the Attestation Driver can propose a mitigation strategy in case of an untrusted node, following another well-defined data interface that is defined in Table 3.4. As the Attestation Driver is the only component that is actually responsible for the verification of the Attester, it is considered as the most suitable entity responsible for defining a possible mitigation strategy. Two possible strategies have been defined in the scope of this dissertation, the first being the termination of the node and the latter its isolation from the management network.

As aforementioned, each virtual instance may be attested along with the physical platform. In this case, each host IR will include a list of data as specified in Table 3.5. This shares similarities with the host IR specified in Table 3.3, although it also allows the specification of additional data of the virtual instance, such as its tenant. The generic approach to the design of attestation data formats allows for further customisation depending on the practical use case (e.g. to bind each virtual instance to a specific network service, in case of a VNF).

Once the global attestation result has been generated by aggregating all the individual integrity proofs, the Cloud Verifier leverages the Connector to notify the result to interested third parties. Finally, the global attestation result is forwarded to the Audit Database for long-term storage.

Attribute	Type	Description
`node_name`	String	Name of the target node
`trust_level`	Boolean	Trustworthiness of the target node
`node_extra_info`	Dictionary	Key-value pairs of additional data regarding the node (e.g. tenant)
`analysis_status`	Integer	Is different from zero in case of failure by the Attestation Driver
`analysis_extra_info`	Dictionary	Key-value pairs of additional data regarding attestation (e.g. number of untrusted binaries)
`analysis_driver`	String	Identifier of the Attestation Driver
`analysis_time`	Timestamp	Instant of attestation request
`virt_remediation`	Object	Object that suggests a remediation option for the node in case it is found untrusted

Table 3.5: Cloud Verifier virtual instance integrity report data.

3.6 Whitelist Database

The Whitelist Database is the centralised source of reference measurements of the trusted domain, in charge of collecting the data relevant to each specific Attestation Driver. As aforementioned, the measure is typically a cryptographic hash computed on a software component that is executed by the target. Examples of measures include the BIOS, the boot-loader, the OS kernel, the services and applications run in user-space.

As the traditional RA process is tailored for a single target system, it is typically subject to a single whitelist. This approach cannot be applied as is to a cloud environment, wherein heterogeneous platforms may be verified for integrity by the same Trust Monitor instance. Because of this, this component should contain multiple data sources that can be specific to a certain OS or architecture, so that each Attestation Driver can leverage the correct database according to the target platform.

Moreover, the origin of such data may be different according to the target platform and attestation technique. In case of the physical nodes comprising the cloud infrastructure, they are expected to run a fixed and pre-defined firmware and software configuration for stability and ease of management. Hence, attestation of the physical platform is expected to rely on a limited number of stable whitelists, specific to a certain OS (e.g. CentOS server distribution) and hardware architecture (64 bit). At high level, an OS-level whitelist would include the following information for each file to be measured:

- its cryptographic hash, i.e. the digest;
- the full path of the file on the target file-system;
- the packages in which it is contained or it refers to (grouped by distribution and architecture).

Given the supported distributions and architectures, the database is initialised and updated periodically by downloading the packages' lists from their official repositories. Alternatively, the database can be updated with release information for components that do not come from public repositories. Additionally, the OS-level database should store the history of each package, reporting the information about its updates (e.g. the type of update). Given the packages' history, several trust levels can be specified by the Attestation Driver:

- **Level 1.** OS-level integrity verification in the node is running correctly.
- **Level 2.** Integrity verification is working correctly and all the software is found in the reference database, but there is at least one binary with a known security vulnerability.
- **Level 3.** The integrity verification process is running as expected and all the software is both well-known and without security vulnerabilities, but at least one binary has a known functional bug.
- **Level 4.** Integrity verification has successfully detected all the software as trusted, and neither security vulnerabilities nor functional bugs are found in the measured software.

On the other hand, virtualised instances are much more volatile in nature, as they rely on software images that may be updated, overwritten and deleted frequently. Moreover, they may be running custom software and configuration more probably than the physical system, as they are not required to be as stable as the OS at hypervisor level. In addition, the whitelists of virtual instances can be significantly different depending on the type of virtualisation. As aforementioned, containers share the same kernel as the host and focus on sandboxing the user-space. Because of this, the whitelist of a certain container is expected to focus on user space programs rather than firmware or kernel measures (which would be still attested as part of the host integrity verification).

With respect to the NFV environment, attestation of VNFs introduces a significant challenge regarding reference measurements. Several use cases, such as SECaaS, imply that developers can push their own VNF images to the centralised image store. These shall be attested against reference measurements that are either managed by the CSP (e.g. the database of the packages published for a certain Linux distribution) or provided by the VNF developer itself, e.g. in case of custom binaries or configuration. Hence, there is a need to establish a trust relationship with the VNF developer. It is to be noted that a successful attestation does not ensure that the software is not harming the system, but only that it behaves as expected originally.

3.7 Attestation Driver

The Attestation Driver is one of the key features of the Trust Monitor when compared to other existing trust assurance solutions for the cloud environment, as it aims to overcome their limitations with respect to the practical technique that is used for attestation. In fact, the cloud ecosystem comprises several devices with heterogeneous architectures, capabilities and very different computing power. Because of this, it is not possible to propose a single mechanism to secure a generic cloud platform, as it would greatly limit the applicability of the proposal.

The Attestation Driver serves as an interface between the Cloud Verifier and the Attester, that logically acts as the verifier in a generic RA scheme. Its goal it so offer a generic set of functions that are then implemented according to the specific integrity verification scheme. This flexibility enables different use cases, depicted in Figure 3.3 and detailed as follows:

- the Attestation Driver directly requests a proof of integrity to the remote Attester and verifies it (either in software or leveraging a hardware RoT);
- the Attestation Driver, acting as a proxy, triggers an attestation request to an external RA verifier, which manages the connection with the Attester.

Although both approaches are viable and ultimately result in the integrity verification of the target, they have very different costs from the CSP perspective: the first requires more development efforts but allows to centralise the attestation workflows as part of the Trust Monitor application, reducing the surface of attack of the platform; the latter requires a more distributed system but it significantly reduces development and integration costs, as it may leverage *off-the-shelf* software such as readily available attestation frameworks (e.g. Intel Open ATtestation (OAT) [46] and [47] open-source initiatives).

As aforementioned, the Cloud Verifier is the only component that instantiates Attestation Drivers and triggers commands on their interface. The Attestation Driver interface supports both registration and attestation of a node, and it should

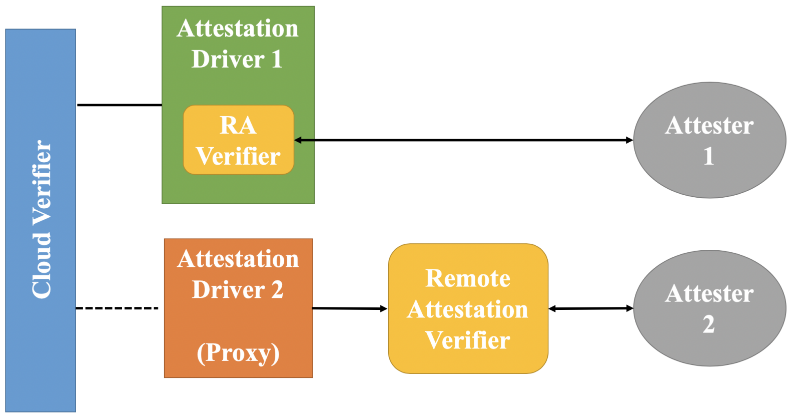

Figure 3.3: The Attestation Driver deployment options

provide a health-check value at runtime as well (that depends on the type of communication with the Attester). Although not strictly required, registration is provided in case the Attestation Driver acts as a proxy towards an external RA verifier so that it can provide to it the data of the new Attester. At this stage, the Attestation Driver receives data about the Attester has defined in Table 3.1. Apart from the origin of the target, additional input data and a type selector can be provided to the Attestation Driver. In fact, we have designed the Attestation Driver to support different types of integrity proofs with respect to the desired trust level. This can be helpful in case the CSP aims to deploy several instances of the same Attestation Driver to protect its nodes and to choose different trust levels depending on the end node (e.g. by enabling TC-compliant run-time verification only for the mission critical services to spare computing power). Attestation is performed according to the specific integrity verification scheme of the Attestation Driver, and it may leverage the additional data inputs provided during registration. For example, in case of TPM-based RA, the PCR 0 value should be provided as external input to the integrity verification, so that the Attestation Driver can properly start the verification of the chain of trust from the boot phase (via the CRTM) up to the run-time software events. All the Attestation Drivers must comply to the same data format with respect to the attestation result, as already presented in Table 3.3 and Table 3.5 depending on the type of node.

3.8 Attester

The Attester represents the client-side of the integrity verification scheme, whose server part is implemented via the Attestation Driver (both internally or as a proxy, as detailed in Section 3.7). Given the distributed target environment, the Attester should communicate via the network with the server by leveraging standard protocols that can effectively ensure a secure, privacy-sensitive exchange of data. As for the Attestation Driver, the Attester is not tailored for a specific attestation scheme, hence it is a more general definition than the one provided by TC. In fact, it does not imply the use of a hardware-based RoT, nor a TC-compliant IR format. From the logical perspective, it acts as a passive component that is able to provide a proof of trust to the Attestation Driver once requested to do so. The way the proof is constructed is entirely up the specific technology that was chosen by the CSP. Practical examples of Attesters can be found in existing attestation frameworks, such as Keylime [51].

The deployment of the Attester can significantly differ depending on the node it is protecting. In case of a physical host, it is expected to run a single instance of Attester in the user-space of the OS, with sufficient permissions to access the hardware RoT, if any available. In case of a virtual host, it could both be specific to a single instance or be still deployed in the host space, depending on the virtualisation technology (e.g. VMs or containers) and the conjunction with a hardware RoT. In fact, solutions like vTPMs in the Xen hypervisor [7] allow each VM to have its own RoT and, therefore, its own Attester. On the other hand, our proposal that will be detailed in Chapter 4 leverages a single per-host Attester regardless of the number of virtual instances that are spawned on the host.

3.9 Application Programming Interface

The Trust Monitor is designed to be a stand-alone entity that runs within the administrative domain of the CSP infrastructure. Hence, it is part of a distributed architecture and, therefore, it relies on external APIs so that other components can interact with it. In the scope of this work, we have chosen the Representational State Transfer (REST) architectural style [39] for the API specification. This allows a straightforward definition of functions based on resources and HTTP methods. A resource is a specific functionality or state of the application, and it is uniquely defined by a URL. The HTTP methods differentiate the operation that is performed on each resource: *GET* refers to the retrieval of the resource; *POST* is used to add a new resource (or to modify it, although *PUT* is also possible); *DELETE* is used to remove the resource. The proposed design includes two separate APIs, namely the Management API and the Newcomer API, that deal with different resources and with operations that can be performed on them.

The Management API exposes commands to check the status of the Trust Monitor, and to retrieve relevant information about the attestation of the infrastructure, as it is fully detailed in Table 3.6. The resources that are managed by this API are the status of the application, the trust level of the infrastructure, the digests

API call	**HTTP method**	**Parameters**	**Description**
`/status`	GET	None	Retrieves a health-check report about the TM application.
`/trust`	GET	None	Retrieves the latest global attestation status of the whole infrastructure.
`/trust`	GET	Node name	Retrieves the latest node attestation status.
`/digest`	GET	None	Retrieve the list of digests that are whitelisted at a given time.
`/digest`	POST	Digest Path name OS data	Adds a new digest for a given file (identified by its path name) for a certain OS.
`/digest`	DELETE	Path name	Remove the digest for a given file from the whitelist.
`/audit`	GET	None	Retrieve all the audit logs for the whole infrastructure.
`/audit`	GET	Node name Start date End date	Retrieves the audit logs for a certain node and for a specific period.

Table 3.6: Trust Monitor Management API

in the Whitelist Database and the audit logs. Apart from the reference measurements' functionalities, the majority of the API calls are read-only. With respect to the calls related to attestation, they may be implemented by retrieving the latest attestation result that is already available in the Audit Database, especially in case of frequent periodic integrity checks.

The Newcomer API is focused on the on-demand registration and attestation of newcomers to the infrastructure. Differently from the Management API, it may be utilised by the CSP at each stage a new physical node is added to the existing compute hosts. Moreover, it can be utilised whenever a virtual instance is executed, so that its trust level is assured at an early execution stage. The API calls are detailed in Table 3.7 as follows. With respect to registration, the data that is either retrieved from the Trust Monitor or provided for newcomer complies to the format specified in Table 3.1. Differently from the Management API, the attestation API call triggers a fresh integrity verification for the target node.

API call	HTTP method	Parameters	Description
`/registration`	GET	None	Retrieves all nodes that were registered to the Cloud Verifier.
`/registration`	POST	Host registration data	Adds a new node to the list of registered nodes in the Cloud Verifier.
`/registration`	DELETE	Node name	Removes a node from the list of registered nodes.
`/attestation`	POST	None	Retrieve the list of digests that are whitelisted at a given time.

Table 3.7: Trust Monitor Newcomer API

3.10 Connector

The Trust Monitor has a plug-in approach to interaction with external services via Connectors. Each of them acts as an independent module that links the Trust Monitor core verification logic, i.e. the Cloud Verifier, and other services belonging to the CSP. The interactions envisioned in this design belong to two main areas:

- notification of attestation results and remediation strategies to any high-level management process of the cloud infrastructure, e.g. the NFVO in case of the NFV paradigm;
- gathering of information from the other services of the cloud infrastructure that are relevant to the attestation process, e.g. the tenant that owns a specific set of virtual instances of a target compute host.

In order to achieve both results, each connector is designed as a three-tier entity as detailed in Figure 3.4. On one side, its inbound interface is used by the Cloud Verifier to request or provide information at any stage of the verification process. On the other side, the outbound interface implements the API of the external service either to provide or to request data from it, depending on the type of connection that is required by the Trust Monitor. In the middle, a business logic takes care of data manipulation (e.g. filtering) if required, and format manipulation between the two interfaces. In this regard, it is expected that the data formats adopted by the Trust Monitor and the CSP services are not consistent to each other, hence format manipulation allows integration of heterogeneous external services with the Trust Monitor core logic.

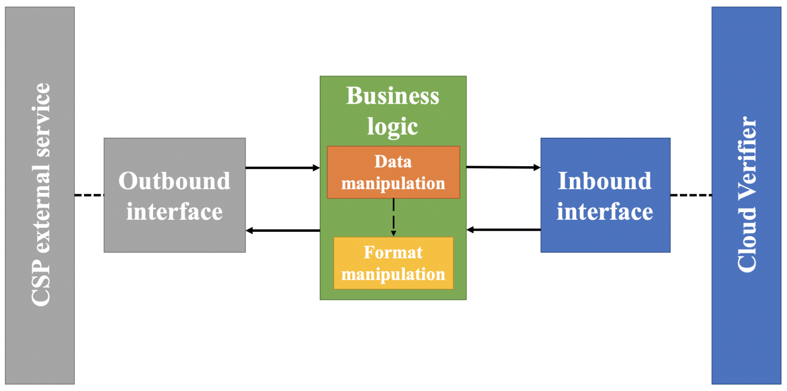

Figure 3.4: The Connector architecture

3.11 Audit Database

In addition to the immediate action when detecting a misbehaviour by one component (isolation of the component, migration of the untrusted virtual instance to a trusted component, modification of the network topology), the Trust Monitor centralises the measurement in the Audit Database through an Enterprise Service Bus (ESB). This is the base component to control the security of the infrastructure over a period of time.

As aforementioned in Section 3.9, the Audit Database records attestations for all the registered nodes and allows a third party to filter data about past verifications. This can be particularly helpful for forensics analysis, in case a node is found untrusted and the cyber-security analyst is interested in recollecting the trust history of the node. As in the case of the Whitelist Database, the Audit Database is a passive component of the proposed architecture, as it acts as a persistent data source. Its inputs are human-readable representations of the global attestation status, as described in Table 3.2.

3.12 Scheduler

The Scheduler is an independent module that periodically invokes the Cloud Verifier for attestation of the registered nodes by leveraging the Management API. This can be entirely skipped in a use case wherein periodic attestation is not demanded, although constant monitoring of the trustworthiness of the platform is desirable in most cases.

3.13 Compliance to requirements

The architectural proposal described in Section 3.3 has been elaborated with the aim of achieving the general high-level requirements of Section 3.2. In this context, Table 3.8 explains how the proposed design is compliant with the requirements set.

Requirement	**Components**	**Justification**
Security	Cloud Verifier Attestation Driver Attester API	The TM enables secure integrity verification via heterogeneous RoTs that can be bound to the hardware. The APIs must be protected against misuse by unauthorised entities.
Privacy	Whitelist Database Attestation Driver Attester Audit Database	The TM databases do not carry specific information about the users of the platform. The data that is exchanged during the attestation procedure is protected for confidentiality and integrity.
Expandability	Attestation Driver Connector	The TM supports heterogeneous integrity verification mechanisms.
Interoperability	API Connector	The TM can fetch information by external actors with customisable outbound interfaces, and can be invoked by external actors with a standardised API.
Scalability	Cloud Verifier Whitelist Database Audit Database Scheduler	The server-side components of the TM can be easily scaled as the data sources are centralised in few entities that can be replicated.
Traceability	Audit Database	The TM records logs of all its operations so that further analyses (e.g. forensics) can be performed on its data,
Availability	Cloud Verifier Whitelist Database	The TM is mostly composed of stateless components that are resilient to failures. The few data sources can be replicated for increased availability.
Compliance to standards	Attestation Driver Attester API	The TM supports TC-compliant remote attestation with tools that are readily available on the market and leverages de-facto standards on data formats and communication (e.g. REST, TLS).

Table 3.8: Trust Monitor mapping to requirements

Chapter 4

Container-based VNF attestation

This chapter drills down on the technical details of the VNF attestation technique that we have developed and integrated within the Trust Monitor as part of this thesis, i.e. Docker Integrity Verification (DIVE). The goal of this part of the work is to offer a practical mechanism that is readily available to implementers to secure the run-time life-cycle of VNFs by assessing their execution state at any given time.

4.1 Requirements

We have defined a specific set of requirements for the design of the run-time integrity verification scheme for container-based VNFs. These address the current limitations of the state of the art on container security, and try to overcome them by encompassing TC principles in the design.

Hardware-level security and privacy. The integrity verification scheme must rely on a hardware RoT, i.e. its security and privacy must be guaranteed by secrets and mechanisms that are protected in a secure, tamper-resistant (or tamper-proof, ideally) device.

Run-time validation. The system must allow for run-time validation of containers during their whole life-cycle, so to ensure that the chain of trust is not invalidated at any stage after the initial deployment.

Scalability. Given the highly virtualised target environment, the integrity verification scheme must be built to scale at the increase of the number of containers.

Technological readiness. The solution must rely on software tools that are consolidated in the market, so that it can be effectively implemented in a production-oriented environment.

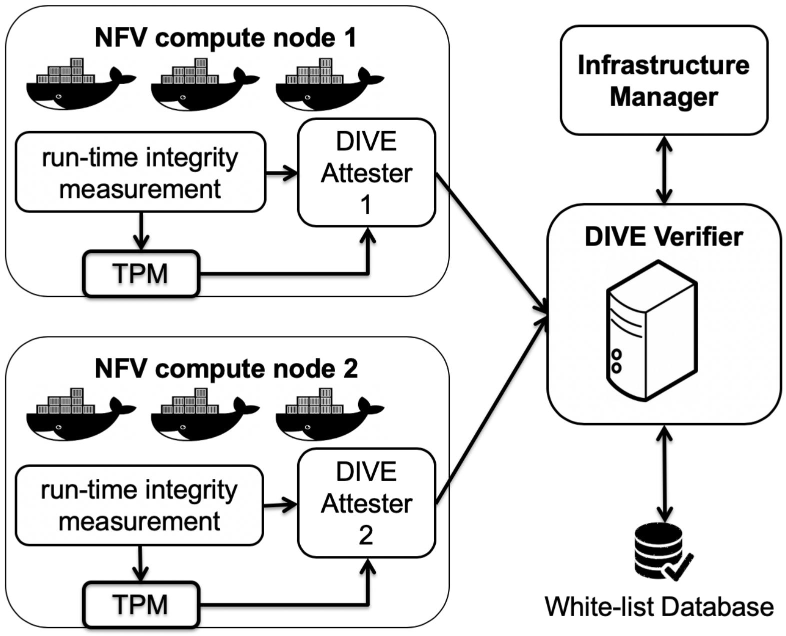

Figure 4.1: DIVE high level architecture

Interoperability. The solution should abstract from the practical deployment use case, so that it can be applied to generic integrity verification schemes tailored for lightweight virtualisation. Hence, the technique itself should not depend on domain-specific information, such as VNF data in the case of NFV.

4.2 Overview

The DIVE architecture complies to the TCG standardised RA scheme, which includes the Attester and the Verifier as the two main actors. The first is the attesting party that runs on the target node and measures its state, leveraging the hardware RoT as a *trust anchor* for storing measurements in a secure way and to authenticate the proof of integrity, i.e. the IR. The latter is the remote party that initiates the RA scheme and validates the IR, allowing trusted third parties to discriminate about the trustworthiness of the target node. In this regard, the concept of trust does not address whether the node is *good* or *bad*, but only if it behaves according to a known-good state. Figure 4.1 represents the DIVE architecture, including the Attester, the Verifier, the Whitelist Database, and the Infrastructure Manager.

The Attester is deployed on the container compute node, which runs the container runtime (acting as a local hypervisor) and is able to run virtualised workloads via containers. It is equipped with a hardware RoT, i.e. the TPM, that secures the run-time measurement architecture. The Verifier is deployed in a trusted domain together with the Infrastructure Manager, and is connected to a Whitelist Database as a centralised data source of reference measurements. The Infrastructure Manager represents a logical monitoring entity that sits in the administrative domain of the virtualised infrastructure, in charge of issuing RA requests to the Verifier for any Attester available in the infrastructure. Being an administrative entity, the Infrastructure Manager is capable of taking action in case any container or the underlying host is detected as untrusted. For instance, the physical host may be segregated from other compute hosts in case it has been tampered with. On the other hand, an untrusted container may be simply terminated by the container runtime in case the underlying host is in a trusted state. Finally, as the Infrastructure Manager is responsible of the scheduling of RA requests for each compute host, it has to keep records about all the physical nodes available in the infrastructure, alongside with the containers that are running at any given time. In order to achieve this goal, the Infrastructure Manager may interact with the container runtime available on each target host. The goals of the DIVE verification scheme performed by the Verifier are multifold:

1. first, the Verifier has to check the boot-time integrity of the target host, ensuring that its BIOS and firmware were not tampered with since the last startup;
2. then, the Verifier ensures that the run-time software events performed by the host comply to reference measurements that were previously white-listed, including the execution of the container runtime;
3. finally, the Verifier validates the run-time software events performed by each container that is running at the RA request instant, ensuring that they do not deviate from a white-list of known good measurements.

Our proposal aims to overcome limitations of existing boot-time integrity validation solutions, that are typically performed with a static methodology. Technologies such as Docker Content Trust (DCT) [22], Intel Trusted Docker Containers can only detect whether images have not been tampered prior to launch [101] by verifying a digital signature over the image.

A similar approach is adopted by the Cloud Integrity Technology (CIT) [47], a cloud attestation framework which enables load-time integrity verification for VM and container-based virtual instances, and integrates with the OpenStack cloud management system. Unfortunately, these solutions protect against fake or manipulated images but do not cover the whole service lifetime, because the image and its internals may be changed at run-time by the host or by external attackers exploiting some vulnerability. For instance, an attacker that has gained a privileged access to a running container may compromise it by modifying service configurations and binaries, launching malicious scripts, or starting new processes, all without being detected by the aforementioned techniques (since they are concerned only with integrity at load time). In order to tackle this problem, we propose a solution for software integrity attestation of a Docker environment at run-time, which covers both the host and the services in the containers. Our focus is to demonstrate whether the software that is running in the host and the containers has not been tampered with, in order to support trust in the correct behaviour of the node.

4.3 Technology

This section describes the technological selection for the DIVE solution that fulfils the *Technological readiness* requirement specified in Section 4.1. This encompasses several open-source tools that are readily available on standard Linux distributions, so to maximise their applicability to practical use cases.

4.3.1 TPM-based integrity measurement architecture

Containers leverage Linux kernel functionalities that allow to segregate a process (or a group of them) from the others that are running on the host OS, letting each container instance to have a limited view of the execution environment as if it was a completely separate host. From the application point of view, a container is similar to a VM as it is able to access certain resources independently from the other instances and the underlying host. In practice, each container is executed on the same kernel, which both enables better performance (i.e. smaller memory footprint, faster spin-up time) and less overhead when integrated with hardware security mechanisms. In this regard, we have investigated the feasibility of extending the IMA technology [45, 80] to containers. IMA is a kernel module that is readily available in modern Linux distributions (since 2.6.30) and implements the TCG Integrity Measurement Log specification [87]. This consists of an extension of TCG measurement process (that leverages the TPM secure registries, the PCRs) to software that is run dynamically at application layer via the *binary attestation* approach. This consists of the recording of software events, i.e. binaries run on the system or file open for read, via their cryptographic hash. The list of digests, alongside the corresponding file path and name, is provided to the Verifier for verification against a white-list. In this context, the TPM secure storage is used as a trust anchor for implicit integrity verification of the measurement list, as its PCRs are not directly compared to known-good values. In fact, as each PCR is only updated via an extend operation that depends on the previous history, it would

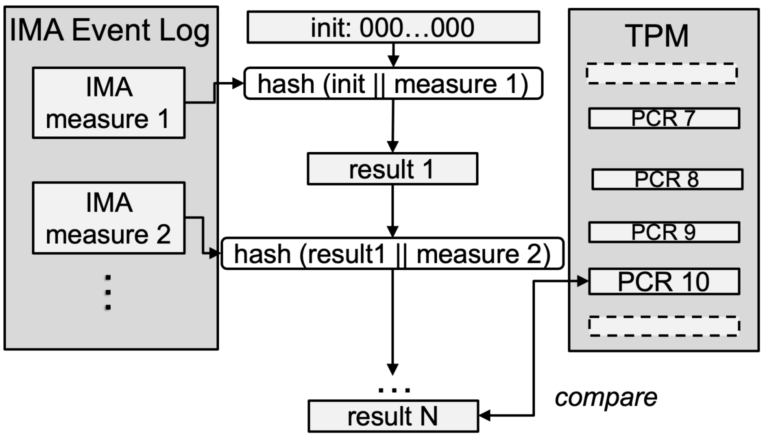

Figure 4.2: The Integrity Measurement Architecture process

```
PCR# template-hash  template  filedata-hash     filedata-path
10   ddee...b59b    ima-ng    sha1:9797...45ee  boot_aggregate
10   180e...8a52    ima-ng    sha1:db82...cf3a  /init
10   ac79...ea65    ima-ng    sha1:f778...f4db  /bin/bash
10   0a0d...bb05    ima-ng    sha1:b0ab...59a7  /lib64/ld-2.27.so
10   0d6b...0011    ima-ng    sha1:ce82...c838  /etc/ld.so.cache
```

Figure 4.3: IMA event log

be impossible to ensure that software events are recorded in the same exact order within different executions.

The IMA process, depicted in Figure 4.2 is straightforward: once activated at startup, it starts the recording of software events by computing their digests and appending them to an internal data structure, i.e. the IMA Event Log, which is exposed to the host as both binary and textual files via the *securityfs* virtual filesystem. An excerpt of a standard Event Log is presented in Figure 4.3, and it includes the following information:

- the PCR index (that defaults to PCR-10);
- the template hash, which consists of a cryptographic hash computed over the pathname and software component;
- the template identifier, selectable from a list of pre-defined templates available at kernel level (e.g. `ima-ng`);
- the file content cryptographic hash;
- the pathname of the software component that is measured by IMA.

```
measure func = FILE_MMAP mask = MAY_EXEC
measure func = BPRM_CHECK mask = MAY_EXEC
measure obj_type = CONFIG
```

Figure 4.4: IMA application policy

The default cryptographic hash algorithm used by IMA is SHA-1, although recent development efforts are extending support to the SHA-2 family. The list of software components that are measured by IMA depends on the *policy* that is specified by the system administrator in the configuration files of the module, and that is read at each boot by the kernel. Figure 4.4 presents an application policy that prescribes the measuring of all files mapped in memory as executables (e.g. binaries, scripts), any file executed by the `execve()` system call, and any file labelled as configuration. Although IMA could be theoretically enabled in a system that is not equipped with a TPM, its security is significantly enhanced in presence of the hardware RoT. In fact, IMA automatically extends each measurement in the target PCR before the corresponding software component is accessed by the system, guaranteeing that the digest cannot be tampered by the component itself. In order to achieve this capability, IMA leverages the Linux Security Modules (LSM) framework. In particular, the LSM kernel system hooks allow processes executed in kernel space to take action in case specific policies are met, such as the execution of a system call. Compared to user space solutions, LSM hooks have a lower performance overhead and indeed they represent the building block at the base of Mandatory Access Control solutions such as SELinux.

IMA is considered by the TCG as the preferred solution for run-time integrity verification in a RA scheme: on the Attester side, the IMA Event Log is presented as part of the IR alongside a quote of the PCRs that include the register used by IMA for aggregation (typically the PCR-10); on the Verifier side, the verification consists of re-computing the PCR aggregate by extending all measurements in the IMA Event Log and comparing the final value with the PCR extracted from the quote. Compared to TCG standard Measured Boot, IMA is more flexible as it does not discriminate about the order of execution of software components, making it more practical in real use cases.

When applied to a container compute host, we have observed that IMA is not capable of discriminating software events of the host from containers. In fact, all the software events are recorded by IMA regardless of their execution environment, as they leverage the system calls of the same kernel. This capability is exclusive to containers based on the Linux Container (LXC) paradigm, and it cannot translate as is to VMs (that encapsulate their own kernel). Because of this, we base DIVE on the exploitation of LSMs as kernel system hooks to record measurements in containers, alongside the host. In order to achieve this goal, the following limitations must be addressed:

- neither the LSM framework or IMA have built-in support for container namespaces, hence all measurements are recorded regardless of their execution environment;
- the built-in IMA templates present all measurements in the global Event Log, hence a Verifier cannot easily guess if a container or the host is not trusted at any given time.

Both of them have been overcome in this research by investigating low-level evidences of container instances that are available at kernel level and can be exposed to IMA during the measurement phase. The result of this investigation, applied to the well-known Docker container engine, is presented as follows.

4.3.2 The Docker Device Mapper storage driver

Docker [23] is an open source project that originally started as a minimal wrapper around the LXC technology [58], which provides built-in support in the Linux kernel for sandboxing at application level. Docker has rapidly become the de-facto standard of containers, it implements a custom runtime that is different from LXC and is the technology that moves the Kubernetes [52] cluster management engine, which is a well-known solution for deploying workloads in a cloud environment.

Any Docker container is spawned from an image, which comprises a series of read-only layers and a top read-write layer in which the container processes are executed. Once spawned, each container is identified via a Universal Unique IDentifier (UUID). This layered approach, depicted in Figure 4.5, allows multiple containers to share a large part of the image thanks to a *union* file-system that implements a union mount of different file-systems. The container runtime is responsible for the merge of the image layers once the instance is deployed thanks to a *copy-on-write* operation. This is enabled by the *storage driver* component, which is part of the Docker engine and implements different mechanisms depending on the underlying Linux distribution and host file-system. Depending on the Linux distribution, Docker uses different storage drivers as the default option (e.g. Aufs on Ubuntu, Device Mapper on CentOS), but this is not a constraint as it is a configurable parameter during installation. Aufs operates at file level: if a file is going to be modified, then the file is entirely copied inside the read-write layer and modified there. Device Mapper operates at block level: when a file is going to be modified, only the blocks of interests are moved to the read-write layer. Thus, the latter one is a better choice for performance, because the latency for the copy-on-write operations is reduced.

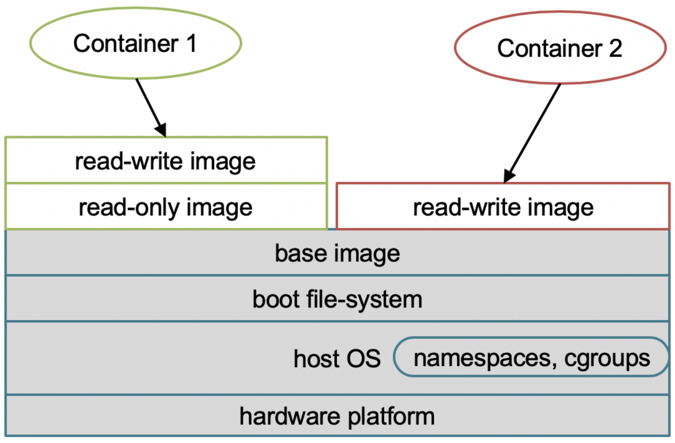

Figure 4.5: Docker file-system layer structure

Device Mapper creates a new virtual device for each container created by the Docker daemon, giving to the virtual device a fixed major number and a growing minor number. In fact, any Linux device is identified by a 32 bit identifier composed by the major number (the most significant 12 bit, typically associated with the driver used for managing the device) and the minor number (the least significant 20 bit, identifying the exact device). In particular, Device Mapper uses either 252 or 253 as major number, depending on the package compiled for each Linux distribution. For example, considering 253 as the major number (as in CentOS), the starting pool will have the device number `253:0`, the first container will have `253:1`, the second `253:2` and so on.

The virtual device identifier is an important parameter in our solution as it is used in measuring the integrity state of different containers at run-time, allowing the Verifier to differentiate among the measurements recorded in different containers. In fact, once a file is measured via its *inode* according to the IMA policy, the Device ID that is associated to it can be retrieved from the kernel data structure of the inode and stored in the IMA Event Log. This extension of the IMA logic made it possible to implement a custom template for container run-time attestation, as presented in Figure 4.6. In this example, the measurements belong to the host system (`8:19`) and two separate containers (`253:1` and `253:2`). The Device ID can be then used at application level by the Attester to link each measurement to a different container. This mapping step is necessary as, at the best of our knowledge, there is not a direct way to access the Docker container run-time information (e.g. its UUID) in kernel space.

```
PCR# templ-hash  template dev-id filedata-hash    filedata-path
10   ddee...b59b ima-dev  8:19   sha1:9797...45ee boot_aggregate
10   180e...8a52 ima-dev  8:19   sha1:db82...cf3a /init
10   ac79...ea65 ima-dev  253:1  sha1:f778...f4db /bin/bash
10   0a0d...bb05 ima-dev  253:1  sha1:b0ab...59a7 /lib64/ld-2.27.
    so
10   0d6b...0011 ima-dev  253:2  sha1:ce82...c838 /etc/ld.so.
    cache
```

Figure 4.6: IMA container attestation template

4.4 Compliance to requirements

Table 4.1 describes the mapping between the requirements described in Section 4.1 and the DIVE architecture.

4.5 Implementation

Several TPM-based RA implementations are available from the open source community, namely the Intel OAT and CIT frameworks, and the Keylime architecture (sponsored by RedHat and integrated in latest RedHat Enterprise Linux distributions). The DIVE Proof-of-Concept (PoC) is based on the OAT framework version 1.7 [46], as it is the only project that implements a TCG-compliant IR. It is a Java project that implements both the Verifier, i.e. the OAT Attestation Server, and the Attester, i.e. the OAT HostAgent. The OAT HostAgent is deployed on each monitored node, it runs as a daemon with administrator privileges and is initialised with the digital certificate of the OAT Attestation Server so that it can encrypt its IR with the Verifier public key. The OAT Attestation Server offers both a REST API, which is the primary tool for integration of the RA scheme in a monitoring architecture, and a web interface to display the history of attestations. The REST API leverages TLS for communication encryption and mutual authentication of the parties. This allows only trusted third parties to query the attestation of registered nodes. DIVE leverages the TPM 1.2 cryptographic device, although a more recent version exists, i.e. TPM 2.0. In fact, the TCG-compliant IR does not support the latest iteration of the device [88]. Moreover, the current IMA implementation only extends measurements using the SHA-1 algorithm, hence it does not leverage the extended PCR banks (supporting the SHA-2 family) of TPM 2.0. We have modified the OAT HostAgent application so that it includes additional information about attested containers in the IR, namely the mapping between their container UUID and Device ID. To achieve this, we have integrated this component with the Docker command line, which interacts with the local Docker daemon running on the

Requirement	**Justification**
Hardware-level security and privacy	DIVE remote attestation relies on the TPM 1.2 as the hardware Root of Trust for establishing a chain of trust from the boot phase up to the run-time software events in both the host and the containers.
Run-time validation	The DIVE Integrity Report includes the IMA event log, comprising run-time measurements about the host and the containers. The integrity of this log is ensured by the TPM, whose PCR 10 contains an aggregate value of all the entries recorded by IMA.
Scalability	The DIVE Verifier can request integrity proofs to multiple Attesters simultaneously to limit the overall integrity verification latency. Moreover, the DIVE attestation scheme can scale horizontally by increasing the number of Verifier instances, as they are mostly stateless entities.
Technological readiness	The DIVE architecture leverages the TPM 1.2 cryptographic device, readily available in several commodity hardware devices. Moreover, the IMA module is supported by the kernel of a wide variety of Linux distributions. Finally, Docker represents the de-facto standard of process containers.
Interoperability	The DIVE architecture enables run-time integrity verification of container workloads regardless of the application domain.

Table 4.1: DIVE mapping to requirements

compute host. This, together with the IMA Event Log presented in Section 4.3.2, allows the Verifier to map each measurement to a different container.

We have extended the Docker command line tool to provide a mapping between each container UUID (provided to the Verifier by the Infrastructure Manager) and its Device ID. Before release 1.10, Docker created a sub-folder inside the `/dev/mapper` directory for each running container. The name of this folder included both the container and its device identifier. Since release 1.10, this is no longer true, as the mapping between identifiers must be retrieved by manually inspecting the `lsblk` command output and the Docker command line tool (`docker inspect` and `docker ps` commands). This process may introduce a non-negligible delay at the Attester side at attestation time. Hence, we have added a new command, named `docker raInfo`, to reduce as much as possible the number of I/O operations and provide the mapping between container UUID and Device ID for each Docker container. The `raInfo` function leverages the Docker library bindings for the Python programming language to inspect Docker containers in execution at a given time via a UNIX socket, which allows to map the container UUIDs with the Device IDs. This optimisation allows to bypass the latency required for the invocation of the `docker inspect` and `docker ps` commands, and the subsequent parsing of the commands' output. Nevertheless, the `raInfo` function still needs to execute the `lsblk` command (and parse the output) to retrieve the list of Device IDs.

The default IMA process measures every file that respects the IMA policy only the first time that it is loaded into memory, via its inode. Hence, the inode of the file must vary if the file has been modified. This can be obtained by mounting the host file system with the `-i_version` flag. IMA internally implements a hash-table that keeps measures of software events together with their template entry. In order to be included in the measurement log, a file must conform to the IMA policy, its digest must not be included in the hash-table, and its inode must not be already measured by IMA. This behaviour is due to performance reasons, but it can have an undesirable effect in a dynamic environment, where a specific device ID may be reused by a different container at execution time. To overcome this issue, we have patched the IMA source code by adding two kernel boot flags: `ima_cache1` to re-measure each file by resetting the integrity information associated to its inode; `ima_cache2` to skip the check of the digest against the internal hash-table. The `ima_cache1` flag can be used to ensure that all software events are re-measured, even if run by devices (i.e. containers) with the same identifier. The `ima_cache2` flag can be used to ensure that all software events executed by a device (i.e. container) are present in the measurement log. This feature brings another advantage to our solution, compared to the default IMA process: if more than one container share the same Device ID at different stages of execution, their measurements will still be included in the host measurement log, allowing the Verifier to reconstruct the history of software events executed in each container.

4.6 Experimental evaluation

We have evaluated the DIVE PoC in a laboratory test-bed comprising two separate machines: the first, equipped with a dual-core Intel Core i7-4600U CPU at 3.3 GHz and 16 GiB of RAM, that runs the OAT HostAgent, the Docker Consumer edition version 18.03.1-ce with Device Mapper version 1.02.110, driver 4.34.0, and the CentOS 7 server distribution with a modified Linux kernel 4.4 that includes the

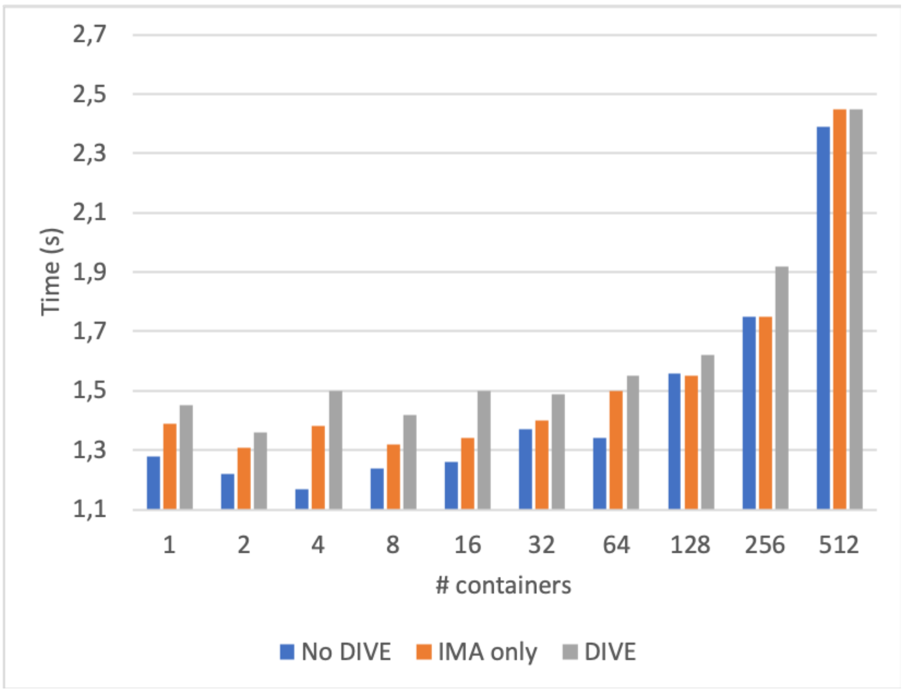

Figure 4.7: DIVE performance drop at container spin-up

IMA modifications; the latter, running in a VM with a dual core CPU at 2.4 GHz and 4 GiB of RAM, that runs the OAT Attestation Server.

The first test focuses on the evaluation of the performance drop on Docker containers when DIVE is enabled on the container compute host. In particular, we have run a variable number of instances (up to 512 containers, as higher values cause failure in resource allocation on the test-bed) and we enable an IMA policy to measure all executables and files open for read in each container. Then, we have measured the latency introduced by DIVE on specific phases of the container life-cycle, namely the instance startup, its termination and removal of resources (e.g. network interfaces, disk storage). The results are averaged over ten separate runs, and are depicted in Figure 4.7, Figure 4.8, and Figure 4.9. These results are to be interpreted as the latency introduced by the operation (run, stop and remove) on a baseline of already active containers, whose number is specified on the X-axis.
The results in Figure 4.7 show that the average startup latency of the container with DIVE enabled requires 170 ms more than in an insecure environment, and is not dependent on the total number of containers. This is due to the fact that a new container spin-up directly adds workload to the attesting platform, and at each startup there is a number of measurements that have to be extended into the TPM and added to the IMA Event Log. In this test, we estimate that any new measurement at container startup requires 11 ms. With respect to container termination and removal, Figure 4.8 and Figure 4.9 show that the DIVE impact

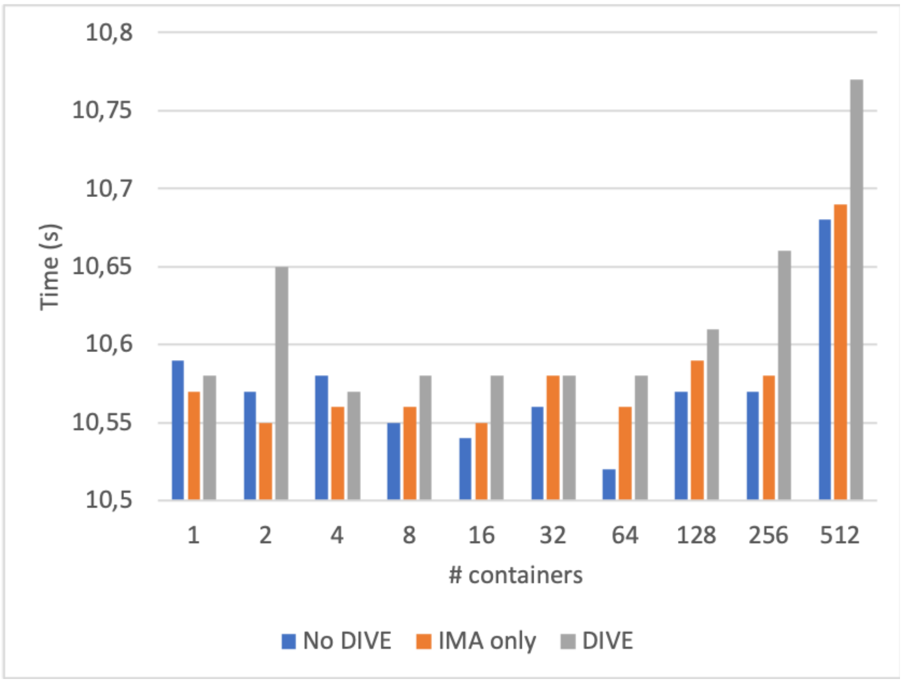

Figure 4.8: DIVE performance drop at container stop

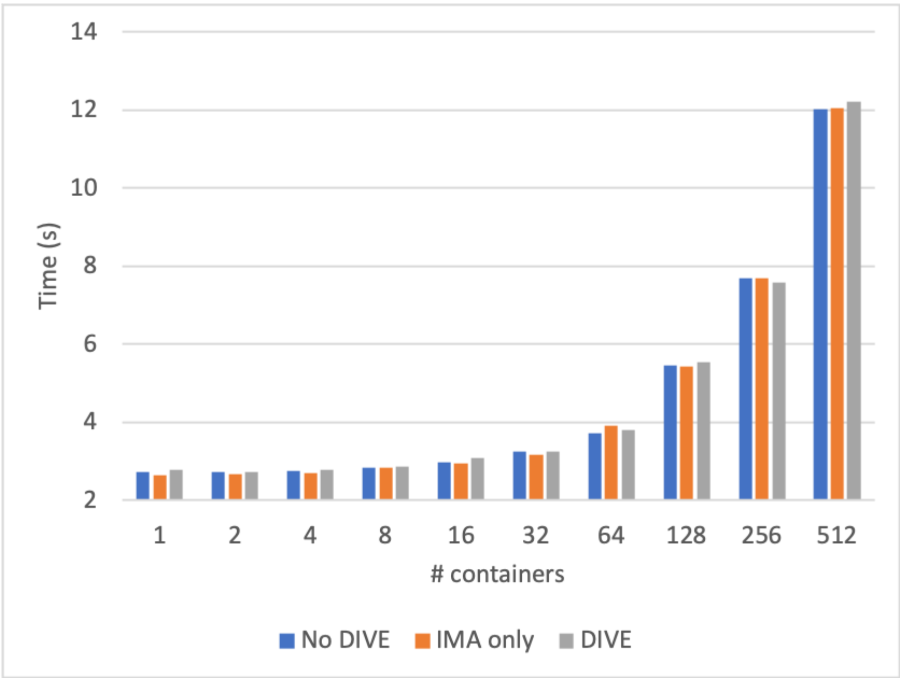

Figure 4.9: DIVE performance drop at container removal

	No DIVE	**IMA only**	**DIVE**
Min	134855	133849	133799
Avg	135284	134623	134440
Max	135602	135626	134769
Index	100/100	99.51/100	99.37/100

Table 4.2: DIVE performance index

on performance is minor, if any, and it may depend on the LSM system hooks triggered by IMA.

Then, we have simulated a real world use case, with the containers set to perform operations similar to those of an HTTP server reacting to incoming requests. In this test, each container creates a single 1 MiB file and then starts an infinite loop to repeatedly compute the SHA-512 digest of this file. At every computation a counter is updated on disk. This approach mimics the case of a server reading a request, performing a computation and writing log data on disk. To avoid considering the impact of the network interface performance (which could be a bottleneck with many containers) we deliberately decided to perform only local I/O operations. To start and stop all operations simultaneously (since all containers must be started sequentially in our test and this takes a lot of time), all container are mapped to a volume linked to the same folder, waiting for a trigger file to be present. Once the trigger is created, all the containers start working simultaneously and at every step test if the trigger file still exists. If not, the container enters a sleep mode, stopping the computation and checking periodically if the trigger is re-created to repeat the process again. In this test we started 256 containers to execute the aforementioned task. Same as the previous test, we repeated the test ten times for each of three different settings: plain Docker environment, Docker with IMA enabled, and finally Docker with the DIVE process. Each run of the test lasts 300 s, and the recorded results are presented in Table 4.2. It is clear that the differences among these three settings are minor: the overall performance is only slightly affected by the introduction of the Remote Attestation. Considering as a reference the average value of performed operations without DIVE, the performance only drops by 0.63% in the full case. So we conclude that the performance impact of our proposed integrity verification feature is nearly optimal and suitable for application in real-world cases, with respect to the overhead on the attested platform.

Table 4.3 presents the results of our test about the global performance of the integrity verification process with different number of active containers (i.e. different number of IMA measures, since the relation is linear). The test addresses the baseline latency that is recorded when no containers are being attested, as the underlying host is measured for integrity nonetheless. The time is given as a total and also split into two components related to the Attester and the Verifier,

	# containers	**0**	**32**	**128**	**256**	**512**
time (s)	**Attester**	3.71	3.80	4.05	4.99	6.56
	Verifier	1.56	1.63	1.77	1.95	2.35
	total	5.27	5.43	5.82	6.94	8.91

Table 4.3: Latency of DIVE integrity verification

since the time for transmitting the IR is very small and hence negligible: with 512 containers the number of IMA measures in our test is around 4500 and the size of the IR is about 1.38 MiB, which only needs around 0.1 s to be transmitted. The total time (and also the individual components) grow roughly linearly with the number of active container (given that in this case each container produces the same number of IMA measures). The time taken by the OAT HostAgent is related to the quote operation and the generation of the IR, while the time of the Verifier includes the effort to verify the signature of the IR and to compare the IMA measures with the reference database. The other steps have small influence on the overall time. Even with 512 active containers, the time needed to attest all their individual measurements is less than 10 s that we deem a good result weighting the integrity guarantee provided by our solution. In case this performance is not considered adequate, there are margins for improvement. On the Attester side, the hard limit is the time taken to perform the digital signature over the IR by the TPM, which requires an approximate time of 2 s. The rest of the time is spent in creating the IR and this time can be reduced by creating differential reports (i.e. containing only those measures added since the last IR) that would benefit also the Verifier as it would have to perform less queries to the golden database. In case a high-frequency verification is desirable, then dedicating a whole core to the Docker host system would be an option, so that the OAT HostAgent would not have to compete with the containers for the CPU. Additionally re-implementing the OAT HostAgent as native code would improve performance as current implementation is in Java which pays some price on speed and size.

Then, we evaluate the impact of our Docker command line modification on the OAT HostAgent. The significant reduction of the latency needed to retrieve the Docker container mapping to its Device ID with the new command, compared with the non-optimised solution (comprising `lsblk` and standard Docker command line tool) is reported in Table 4.4.

Finally, the evaluation of the resource consumption of the OAT Attestation Server is presented in Table 4.5. We have analysed the CPU and RAM utilisation of the Verifier during the RA process, both for host-only and container verification, up to 512 instances, in ten separate runs and we have averaged the results. The Java program utilises a single core of the CPU on the Verifier VM, and its CPU utilisation is significantly affected by the container integrity verification. This is due

No. of containers	**Non-optimised time** (ms)	**raInfo time** (ms)
1	23	13
2	32	16
5	61	19
10	115	22
20	211	31
50	510	58
100	1020	108
250	2746	364
500	5506	1087

Table 4.4: Evaluation of Docker latency to retrieve Device IDs

# containers	**0**	**32**	**128**	**256**	**512**
% CPU (1 core at 2.4 GHz)	3.51	13.01	14.55	18.10	26.33
% RAM (4 GiB)	7.74	8.09	9.11	12.01	15.10

Table 4.5: Performance analysis of the DIVE Attestation Server

to the fact that the Verifier has to validate a larger IR, and it has to re-compute the aggregate value for PCR-10 over a significantly longer IMA Event Log. On the other hand, memory utilisation does not increase as much when containers are attested, as the Verifier does not keep in memory the Event Log at each verification.

4.7 Deployment as Attestation Driver

The DIVE technique is designed to be agnostic on the specific domain of application, as discussed previously in this chapter. Nonetheless, the final goal of this research is to enable integrity verification in a lightweight virtualisation platform so that it effectively address the lack of trustworthiness in its workloads. Because of this, a significant effort has been pursued in this dissertation in order to offer DIVE as a practical VNF run-time attestation technology that is readily available for the Trust Monitor. In order to achieve this result, we have integrated DIVE as an Attestation Driver, following the *proxy* architecture described in Section 3.7, allowing it to attest both the container compute host and its network services. In this scheme, the Trust Monitor represents the Infrastructure Manager, as it queries the DIVE Verifier for attestation of target nodes. As discussed from the architectural perspective in Section 3.7, the Attestation Driver should support both registration, attestation, and health-check functionalities.

The registration phase of the target node assumes that an OAT HostAgent is already deployed and running, and consists of the following operations, that are remotely issued from the Cloud Verifier to the OAT Attestation Server via its TLS API:

1. set the Original Equipment Manufacturer (OEM), the OS, and Measured Launch Environment (MLE) of the node, i.e. the hardware and software configuration of the node;
2. set the type of analysis to perform on the target node (e.g. the attestation of containers alongside the host), which requires it to be previously set on the OAT Attestation Server;
3. set the host connection data, such as its IP address and hostname;
4. set the host PCR-0 value, which must be retrieved beforehand and depends on the CRTM available on the device.

The OEM, OS, and MLE parameters are necessary to the OAT framework to gather multiple hosts in the same *host set*, i.e. a list of nodes that share the same hardware and software configuration.

The attestation command of the Attestation Driver forwards the call to the OAT Attestation Server endpoint that is responsible for initiating the RA scheme with the target OAT HostAgent. The sequence of steps performed at each attestation between the Cloud Verifier, the DIVE Attestation Driver and the OAT Attestation Server is depicted in Figure 4.10, and described as follows:

1. the Cloud Verifier triggers the RA command to the DIVE Attestation Driver;
2. the DIVE Attestation Driver requests attestation of the pre-registered node to the responsible OAT Attestation Server instance;
3. the OAT Attestation Server requests a TCG-compliant IR to the OAT HostAgent;
4. the OAT HostAgent provides the IR to the OAT Attestation Server, signed with the TPM AK;
5. the OAT Attestation Server verifies the signature of the TPM with the Privacy CA, re-computes the PCR-10 aggregate and ultimately extracts the measurements from the IMA event log, forwarding them to the Attestation Driver;
6. the Attestation Driver interacts with the centralised Cloud Verifier to verify whether all the measurements collected from the host and the VNFs are consistent with the known-good values from the Whitelist Database;

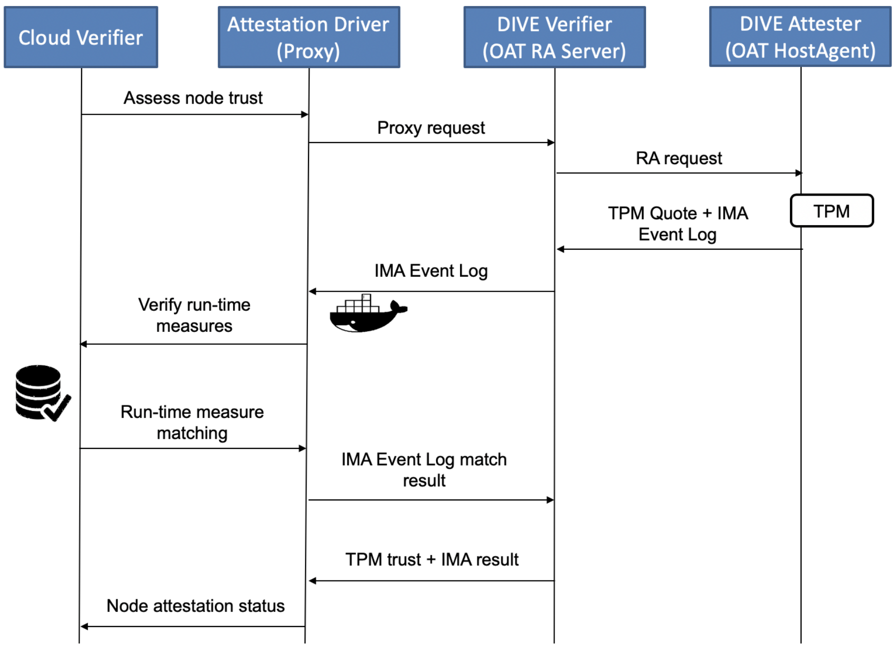
Cloud Verifier
Attestation Driver (Proxy)
DIVE Verifier (OAT RA Server)
DIVE Attester (OAT HostAgent)
Assess node trust
Proxy request
RA request
TPM
TPM Quote + IMA Event Log
IMA Event Log
Verify run-time measures
Run-time measure matching
IMA Event Log match result
TPM trust + IMA result
Node attestation status

Figure 4.10: DIVE integration as Attestation Driver

7. the Attestation Driver provides the result about the white-list matching to the OAT Attestation Server, as it acts as a proxy between the Cloud Verifier and the DIVE Verifier;

8. the OAT Attestation Server internally records the result of attestation, which includes the validation of both TPM signature, the boot-time PCRs and the run-time measurements, and replies to the Attestation Driver with the node attestation report;

9. the attestation report is returned from the Attestation Driver to the Cloud Verifier, that merges it together with the other reports retrieved from the different Attestation Drivers deployed in the infrastructure.

Although it is technically feasible, nor the Attestation Driver or the OAT Attestation Server internally perform the validation of run-time measurements against the Whitelist Database, as they delegate this operation to the Cloud Verifier. This is important to ensure interoperability in case of protection of the same host set by different Attestation Drivers, as each of them would ultimately require an ad-hoc connections to the Whitelist Database.

Finally, the health-check command polls a read-only API endpoint of the OAT HostAgent, i.e. the call to retrieve the list of registered hosts, and returns with either success or failure depending on the HTTP return code.

Chapter 5

NFV threat mitigation

This chapter discusses the requirements of the security management life-cycle as defined by ETSI, and proposes a solution based on the Trust Monitor and the container-based VNF attestation technique to address these requirements. In this regard, a threat model that is specific to the NFV domain is presented as well, in order to assess the effectiveness of the proposal hereby discussed. The SECaaS use case, as defined in Section 3.1.1, is considered for the definition of the threat model as it is one of the most security critical applications of NFV.

5.1 Requirements

As detailed by ETSI [32], the security management life-cycle in NFV is defined a process that follows the VNF instantiation, operation and retirement. The steps that compose such process are defined as follows:

- **Architect.** The security of the NFV platform requires the design of an ad-hoc architecture that is deployed within the infrastructure according to the *separation of duties* with other management entities.
- **Assess.** The assessment of security of the NFV platform requires the instalment of proactive processes that can limit the exposure window of resources to both internal and external threats, resulting in fresh proofs of integrity for further validation.
- **Validate.** The security architecture is in charge of validating the proofs of trust produced by the NFV infrastructure by leveraging strong authentication and integrity verification schemes.

- **Detect.** The security management life-cycle allows the platform administrator to detect anomalies and misuse of NFV resources, so that the actors involved (i.e. the end users, the ISP) can be notified.
- **Prevent.** The security architecture must be able to prevent attacks by performing proactive monitoring of sensitive resources before they can harm the entire system, i.e. by ensuring that VNFs have not been tampered with as soon as they are instantiated.
- **Respond.** Once a resource has been detected as compromised, the security architecture should help with the definition of a mitigation strategy, allowing the infrastructure to recover from the security breach.

The process is iterative, and it should be lasting as long as the life-cycle of the NFV platform.

The NFV threat mitigation scheme that is proposed in this dissertation complies to the aforementioned security management life-cycle. In fact, the Trust Monitor is presented as a long-lived entity that sits in the MANO domain and that periodically assesses the trustworthiness of the NFVI and VNFs by validating their integrity reports that rely on a hardware RoT, the TPM, for their authentication and integrity. This allows the Trust Monitor to detect and prevent incoming threats that occur at infrastructure level via manipulations to firmware and software executed both at the physical and virtual levels. Moreover, the threat mitigation process is engineered to actively respond to threats by initiating a remediation once a node is found untrusted. This leverages a deep integration between the Trust Monitor and other NFV components, such as the NFVO.

At a technological level, the proposed solution relies on certain assumptions that are hereby discussed. First, the target NFV platform relies on lightweight virtualisation for the deployment of VNFs. This solution allows for both scalable and fast instantiation of network services in the ISP infrastructure, and enables the runtime integrity verification scheme that we have presented in this work. In particular, the Docker container engine is deployed on each physical node that comprises the NFVI. Moreover, the VIM interface is expected to expose information about each container that is relevant for attestation, as it was discussed in Chapter 4. In particular, it is required that each VNF is associated to a specific container, so that the Trust Monitor can effectively signal its trust level to the NFVO at each verification. Then, the Trust Monitor is expected to be deployed on an already existing NFV platform, hence a preliminary registration of each physical node is required. This may leverage the API that was discussed in Chapter 3. Finally, the Trust Monitor should be provisioned with golden values for its Whitelist Database, in particular for the configuration and software packages that are deployed in each physical node of the NFVI. This process can be executed off-line at the initial commissioning of compute hosts, and should require interaction between the NFV platform owner, i.e. the ISP, and the CSP. Because of this, private cloud environments that are directly owned by the ISP are preferred. In case of public and externally managed cloud infrastructures, the ISP should leverage on the IaaS service model so that it has control over the resources that are available on the physical infrastructure. Nonetheless, the ISP may not rely on hardware-bound integrity verification in case it relies on virtual nodes only.

5.2 Threat model

The Trust Monitor assesses the trust in the network infrastructure bearing the deployed VNFs, namely each NFVI Point of Presence (PoP) that is in charge of deploying the virtual instances. The trustworthiness of the infrastructure is assessed by performing both authentication and integrity verification. Although attackers tend to exploit multiple vectors to breach into a system, the Trust Monitor focuses on intrusion detection in the network infrastructure, assuming the control and management plane components (VNF store, NFVO, DARE, Security Dashboard of the SECaaS use case) are implicitly trusted. From a technical standpoint, extending the Trust Monitor security concepts to assess the control and management plane, is feasible since they are based on the same kind of computer architecture (in terms of operating system, virtualisation technology and application packaging). Our proposed threat model considers the following threats, classified on whether the attacker has physical access to the infrastructure or not:

- Physical threats
  - (T1) physical eavesdropping: on network wire, bus probing;
  - (T2) physical modification of nodes: chip replacement;
  - (T3) physical introduction of a new/alternate control plane;
  - (T4) flashing of firmware/software of the infrastructure nodes (e.g. a compute host);
- Software threats
  - (T5) zero-day vulnerability exploitation;
  - (T6) malicious (or accidental) administration: configuration modification;
  - (T7) installation and execution of arbitrary firmware/software (e.g. in the VNF level);

The Trust Monitor aims at providing the network infrastructure with detection mechanisms against software-based and low-end physical attacks: T1 and T2 are clearly out-of-scope since the Trust Monitor does not provide any physical perimeter protection. Using TPMs, remote attestation and other TC mechanisms, the Trust Monitor protects the NFV network infrastructure against T3, T4, T6 and T7. Particularly, the TPM protected log of all binaries executed on a node allows the Trust Monitor to detect arbitrary code (T4 and T7). The same mechanism can be used to detect unwanted configuration modification (T6). If an attacker manages to introduce a new control plane entity in the network infrastructure (T3), the Trust Monitor does not detect it directly but instead would detect any unusual or modified behaviour of the computer or network nodes since it would not be correct compared to the genuine control plane components, mainly the NFVO. The Trust Monitor verifies each node against their expected state, as configured by the NFVO; if an attacker changes — even slightly — the configuration of one node, the Trust Monitor will detect it since it will not match the NFVO's view. Looking at T5, this cannot be detected by the Trust Monitor or regular Trusted Computing mechanisms. Nevertheless, zero-day vulnerability can be reduced by using code analysis tools and/or prevent their consequences by reducing the ability of the attackers. Mechanisms such as control-flow protection for instance, could help with that task. Even though, these kinds of attacks are usually the initial attack vector to install additional malicious software on the target, the execution log, verified by the Trust Monitor, permit to detect the subversion.

Each physical node must be successfully authenticated — using hardware-based cryptographic identities — and verified by the Trust Monitor before joining the NFV infrastructure. Leveraging the Remote Attestation workflow, the Trust Monitor can verify the integrity of the code being executed (e.g. running instances of VNFs, software directly managing virtualisation processes, etc) on each physical node, as well as its configuration, both at boot and run-time. The Trust Monitor acts as a continual verification engine for the physical infrastructure hosting the network services, capable of interacting with the rest of the vNSF ecosystem (NFVO, VNF store) as well as the DARE to provide an assessment of the trustworthiness of the infrastructure. Each NFVI node, being equipped with a TPM and suitable software, is able to collect the integrity measurements of both running code (starting from boot-time) and configuration, it is also able to report this data to a third party in a secure and trusted way. The resulting integrity report, which contains the logged software events — as measured by IMA for example — is validated by the Trust Monitor, which maintains a whitelist populated by measurements of known software signatures and their valid configurations.

5.3 Integration within the NFV SECaaS scenario

The Trust Monitor modular architecture leverages Connectors to query external services about information that is required for attestation and to notify results about the ongoing security process, as defined in Section 3.10. This includes the interactions that are specific to both integrity verification (i.e. the retrieval of white-lists for VNFs) and mitigation (i.e. the notification of a failed attestation). From a high-level perspective, the Trust Monitor embeds Connectors that offer two-way communication, either by consuming or producing data from/to the external services.

5.3.1 Inbound communication

With respect to consumption of external data, the Trust Monitor implements Connectors towards the following remote APIs:

- NFVO. Given a target node identifier, retrieves information about its placement in the CSP network (e.g. its IP address) and its activity status. In case of a physical node, retrieves information about the VNFs that are instantiated on it and the VIM that manages the node itself. Attestation of the entire infrastructure never relies on the list of nodes that are available on the NFVO side, but only on those that were pre-registered on the Trust Monitor by the system administrator. This allows flexible verification of a subset of nodes, in case the CSP does not provide hardware support for the entire compute platform.
- VIM. In case of VNF attestation, the VIM is queried for additional information that are specific to the virtualisation layer. In particular, the VIM Connector aims at retrieving the list of deployment details for each container that are specific to attestation (e.g. the device ID).
- VNF Store. This represents the data store for VNF packages, hence it is considered as the most suitable store for white-lists of known good software that is executed by each VNF. Given the decentralised nature of the NFV deployments, it is not practical to record and store all digests into a single database, hence a distributed approach is preferred.

Given its central role in the VNF attestation process, the interactions between the Trust Monitor, the NFVO and the VIM are described in depth as follows. In principle, as the NFVO is in charge of managing the VNFs life-cycle, its integration with the Trust Monitor could be sufficient to support attestation and mitigation

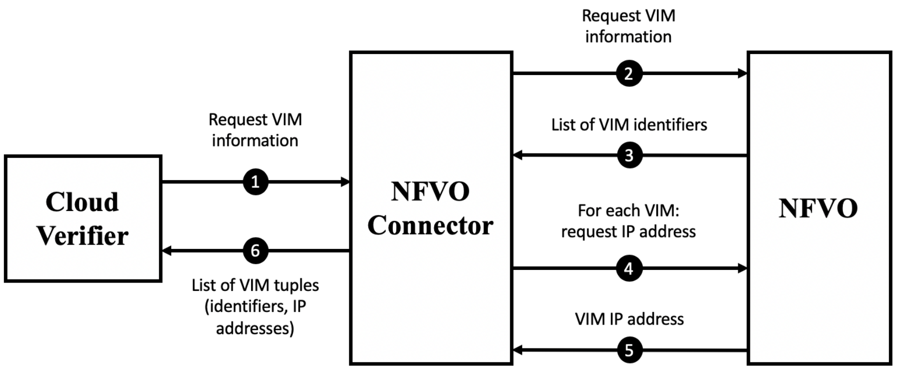
Cloud Verifier
NFVO Connector
NFVO
1 Request VIM information
2 Request VIM information
3 List of VIM identifiers
4 For each VIM: request IP address
5 VIM IP address
6 List of VIM tuples (identifiers, IP addresses)

Figure 5.1: Trust Monitor process to retrieve VIM data from the Orchestrator

strategies. Nonetheless, the Trust Monitor requires information about the virtualisation technology that is not available at NFVO level, as the VIM abstracts the physical layer from the logical resources that are required by service and resource orchestration. This limit makes it impractical for the Trust Monitor to implement an integrity verification process that is independent from the VIM. Because of this, the Trust Monitor initiates several communications towards the NFVO and the VIM whenever it has to attest a VNF. This applies to both periodic attestation, that typically affects the whole infrastructure, and to any targeted verification.

The first phase is depicted in Figure 5.1 and occurs between the Trust Monitor and the NFVO via the Connector. In this phase, the Trust Monitor queries the NFVO to retrieve the list of physical nodes that are available at infrastructure level (e.g. compute hosts, switches). The Connector outbound interface is used to retrieve identification data about each node, namely its IP address. Then, the Trust Monitor filters the list of nodes by retrieving the compute hosts, i.e. the VIMs, depending on whether they were registered to the Cloud Verifier database and they support the container runtime (as available in Table 3.1).

The second phase, as shown in Figure 5.2, requires the Trust Monitor to query each container-enabled VIM in order to retrieve run-time information about containers. In practice, the VIM interacts with the container runtime via an API that is specific to the virtualisation technology. At this stage, the Trust Monitor is expected to retrieve the set of container IDs that are available at each VIM, together with VNF information visible at VIM level (such as the VNF descriptor name, its software image and the network service name).

The final phase of the interaction is depicted in Figure 5.3. At this stage, the Trust Monitor requests to the NFVO a report of run-time information about each VNF that is currently running. Differently from the previous case, at this stage the Trust Monitor can request details about VNFs that are specific to a tenant, and it retrieves the run-time identifier of each instance. This information is necessary

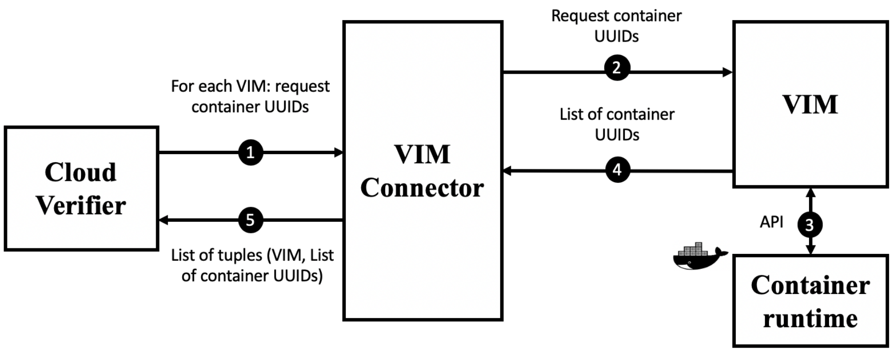

Figure 5.2: Trust Monitor process to retrieve container data from the target VIM

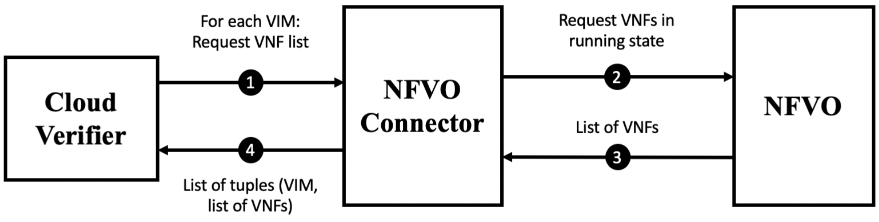
Cloud Verifier
For each VIM: Request VNF list
1
4
List of tuples (VIM, list of VNFs)
NFVO Connector
Request VNFs in running state
2
List of VNFs
3
NFVO

Figure 5.3: Trust Monitor process to retrieve container VNF data

to enforce a mitigation strategy in case a VNF is found untrusted. Moreover, the network service name is used to correlate the available data from previous phases, resulting in an in-depth view about the VNFs that are currently running on each container VIM. This will be later used for the verification phase of the attestation process.

5.3.2 Outbound communication

The Trust Monitor publishes data about the result of attestation via ad-hoc Connectors that are targeted towards the following SECaaS entities:

- NFVO. In case of a failed attestation, both for newcomer nodes and periodic assessment of the infrastructure, the NFVO is notified with a report about the failing node (either virtual or physical) and a proposed mitigation strategy, as previously defined by the security administrator (and introduced in Table 3.4). This is proposed as a solution to enrich telemetry and logging by the NFVO, and can even be used for automated remediation by the orchestration service. Nonetheless, the human factor is typically considered crucial in security operations, hence a proactive approach to remediation may not be desirable in most situations.

- DARE. Being the centralised log collector and the Big Data engine of the NFV domain, the DARE is expected to leverage attestation logs as part of its reasoning capabilities. In particular, it could leverage historical information about untrusted nodes (e.g. the tenant that owns them, or the type of manipulation that is detected by the Trust Monitor) to improve the knowledge about the infrastructure.
- Security Dashboard. As in most ICT infrastructures, NFV platform are expected to rely on manual processes by security analysts in order to address security threats. Because of this, the Trust Monitor publishes records about attestation to the Security Dashboard, that acts as a SIEM of the NFV platform. These can be leveraged at the *triage* phase by the analyst to address the possible security threat (e.g. a change in configuration file) and to decide whether it represents an actual violation of the infrastructure. In this case, the analyst should be capable of implementing a mitigation strategy via the Security Dashboard, that in turn would leverage the NFVO to effectively enforce the strategy on the compromised node.

5.4 Workflows

This section details the integrity verification workflows that we have designed in this work. These comprise both the RA process itself, which is delegated to each Attestation Driver implemented by the Trust Monitor, the verification logic (which requires both internal and external communications, e.g. via Connectors), and the notification and reporting mechanisms. Figure 5.4 shows a high-level view of the process enabled by the Trust Monitor that allows to periodically evaluate the trustworthiness of the NFV infrastructure and integrates with SECaaS components to support mitigation of threats.

5.4.1 NFVI centralised monitoring process

The periodic monitoring process that is implemented by the Trust Monitor relies on a pre-defined list of nodes that were registered to its Management API, and aims to report a complete picture of the integrity of the NFVI infrastructure. Parallelisation is adopted to ensure that the impact of integrity verification of a large set of nodes has a limited impact on performance and latency. The pseudocode of the workflow is reported in Figure 5.5 for completeness.

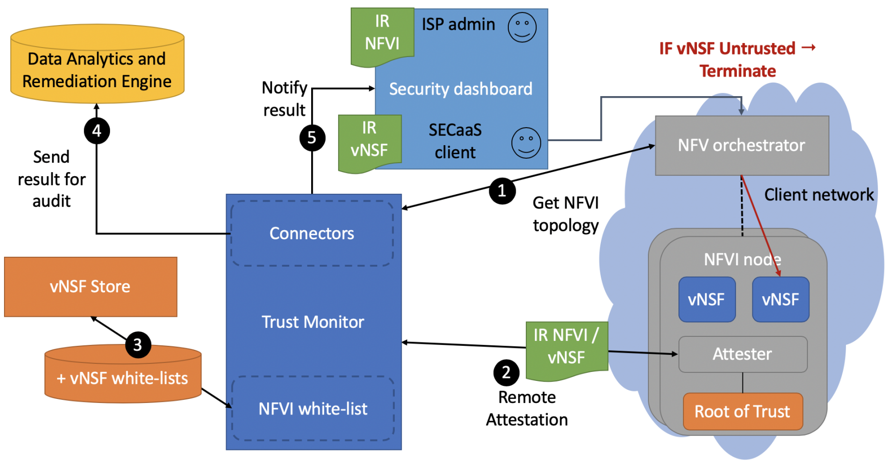

Figure 5.4: SECaaS integrity verification workflow

```
Data: Cloud Verifier node registration table, list of active nodes from
      NFVO
Result: Global attestation status
retrieve list of nodes in active state from VNFO;
foreach node in list of active nodes do
    if node is in set of registered nodes then
        retrieve node attestation configuration;
        instantiate node Attestation Driver;
        add Attestation Driver to list of parallel jobs;
    else
        discard active node from list of attested nodes;
foreach job in list of parallel jobs do
    run Attestation Driver job;
foreach job in list of parallel jobs do
    wait for result of Attestation Driver job;
    update global attestation status;
store global attestation status in Audit Database;
send notification to DARE, Dashboard and NFVO;
```

Figure 5.5: NFVI periodic attestation workflow

5.4.2 VNF integrity verification process via DIVE

The TPM-based Attestation Driver that we defined as part of this dissertation is instantiated alongside the others, and its result is aggregated with the others

```
Data: VIM and VNF information from NFVO, reference measurements
      from White-list Database and VNF Store
input  : Compute host registration data
output: Compute host attestation status
retrieve VIM information by IP address from NFVO;
if node is in VIM list then
    retrieve list of active containers from VIM;
    if list of containers is not null then
        get VNF run-time data from NFVO;
        foreach VNF in list of run-time instances do
            correlate VNF data with container ID;
            retrieve list of reference measurements from VNF Store;
        end
        attest compute host with containers;
        verify integrity proof against White-list Database and VNF
          measures from VNF Store;
    else
        attest compute host;
        verify integrity proof against White-list Database;
    end
else
    discard job;
end
```

Figure 5.6: Container compute host integrity verification workflow

in order to retrieve the global attestation status. The pseudocode detailing the compute host attestation workflow is detailed in Figure 5.6. The same workflow is applied in case of newcomer attestation, although it would typically be limited to host integrity verification.

5.5 Implementation

We have developed a PoC of the Trust Monitor in the scope of the SHIELD project, wherein it was deployed as a centralised monitoring entity for an NFV platform. The SHIELD solution consists of several modules that adhere to the SECaaS use case, allowing the deployment of network security functions, i.e. the vNSFs, that are protected against manipulations by the Trust Monitor and that can be used to protect the ISP client network. Moreover, big data analytics is implemented via the DARE so zero-day threats can be detected by analysing deviations from pre-classified traffic behaviour. In the scope of the SHIELD project, both compute hosts based on the Docker container engine and physical SDN switches are attested via TPMs in order to ensure that both the vNSFs and the network flows that interconnect them have not been tampered with.

The Trust Monitor prototype, made available as open source software [82], allows for deep customisation of its endpoints and interfaces:

- TLS encryption key/certificate for the web APIs;
- Whitelist Database with reference measurements of the NFVI platform nodes;
- different API Connectors to integrate with other SECaaS components, namely NFVO, VNF Store, Security Dashboard, DARE and VIM;
- different attestation drivers to verify the integrity of different NFVI elements (e.g. compute hosts, switches).

The Trust Monitor core application is based on the Django REST Framework [20], a Python framework that allows to create full stack web services based on the REST paradigm. We have selected this technology because of its built-in support for different API authentication methods, connection with heterogeneous databases, and serialisation of data exchanged with other web services. The application does not include a Graphical User Interface (GUI) as it leverages the Security Dashboard for visualisation of attestation reports. The internal database of the Trust Monitor, where the registered nodes are stored, is implemented via SQL. The Redis [78] in memory key-value storage is used as a memory cache to store VNF measurements during the integrity verification phase. This is helpful to mitigate problems with network interaction with the VNF Store. The Connectors are deployed as separate web modules via the Flask [40] technology, which allows to deploy minimal web services that have a limited memory footprint and require little development effort. The Management API and Newcomer API are authenticated and protected for integrity and confidentiality thanks to a reverse proxy, implemented via NGINX [66], that terminates TLS connections with the web clients. Data processed by the Trust Monitor (e.g. VNF white-lists, attestation reports) is described in the JavaScript Object Notation (JSON) format [85], which is easily processed by web-based applications and allows for greater flexibility when compared to formats such as XML, which rely on schemas to enforce a pre-fixed structure to data. An example of global attestation result in JSON format is presented in Figure 5.7. This report shows that a single physical node, i.e. `nfvi-node`, has been verified successfully without unknown or manipulated software packages. Nonetheless, the global attestation result shows that the infrastructure is untrusted, as one of the VNFs that are deployed on the node (an instance of `vnf-1`) is running software modules

```
{
  "hosts": [
   {
    "node": "nfvi-node",
    "status": 0,
    "time": "2020-02-28 10:47:24.218939 +0000 UTC",
    "trust": true,
    "driver": "OAT",
    "extra_info": {
     "n_digests_valid": 465,
     "n_packages_valid": 135,
     "n_digests_not_found": 2,
     "list_digests_not_found": [
      {
        "/usr/sbin/httpd":
          "b280b7d85bc0c00fbaa4b00aeb5a4a122dc20396",
        "instance":"4c01db0b339c"
      },
      {
        "/etc/httpd/httpd.conf":
          "4e1243bd22c66e76c2ba9eddc1f91394e57f9f83",
        "instance":"4c01db0b339c"
      }
    ],
    }
   "vnsfs": [
     {
      "vnsfr_id": "d6c9d9ee-efef-4c2f-be4f-dbcbafd84457",
      "vnsfd_id": "vnf-1",
      "ns_id": "ns-1",
      "container": "4c01db0b339c",
      "trust": false,
      "remediation": {
      "terminate": true,
      "isolate": true
      }
     }
    ]
   }
  ],
  "trust": false,
  "vtime": "2020-02-28 10:47:24.219215 +0000 UTC"
}
```

Figure 5.7: Attestation result with manipulated digests

whose digests could not be found in the VNF white-list. The overall number of offending digests is set in the `n_digests_not_found` field. In particular, as shown in

```
{
  "hosts": [
   {
    "node": "nfvi-node",
    "status": 0,
    "time": "2020-03-01 09:10:11.322931 +0000 UTC",
    "trust": false,
    "driver": "OAT",
    "extra_info": {
     "n_digests_valid": 465,
     "n_packages_valid": 135,
     "n_packages_unknown": 1,
     "list_digests_not_found": [
      {
        "/usr/sbin/my_malicious_script":
          "c21a709dccca01fcad5612afeAa4a122dc27654",
        "instance":"nfvi-node"
      }
    ],
    }
    "vnsfs": []
   }
  ],
  "trust": false,
  "vtime": "2020-03-01 09:10:11.421112 +0000 UTC"
}
```

Figure 5.8: Attestation result with unknown digest

the `list_digests_not_found` list, the `httpd` software has been manipulated with respect to its reference value in the container identified as `4c01db0b339c`, which implements the `vnf-1` application. In addition, the `httpd.conf` configuration file has been manipulated as well. The implementation does not differentiate among configuration files and binaries, as this information is not kept in the Whitelist Database nor in the VNF security manifest. It is to be noted that the attestation report includes additional information about each VNF, such as its network service identifier and a mitigation strategy for the threat (in this case, to either terminate or isolate the node).

Figure 5.8 shows another example of a JSON attestation result wherein an unknown package is found in the host. The `n_packages_unknown` field shows that one file named `my_malicious_script` has not been found in the Whitelist Database, and that its pathname is unknown as well. This means that the file that has been measured is not part of the *golden image* of the host.

The Whitelist Database, implemented via Apache Cassandra [3], contains the

```
{
  "digests": [
  {
    "/usr/sbin/httpd":
      "c9d0a9b168584c5c2bf780a699c68c14a138e5be",
    "instance": "vdu-1"
  },
  {
    "/usr/local/bin/custom_service":
      "bf151ae14b61a637cb48a5fa0d55c1f9a618c395",
    "instance": "vdu-1"
  }
  ]
}
```

Figure 5.9: VNF security manifest in JSON notation

complete data of the executables allowed on the attested platforms. More specifically, it contains the digest, the full path name, and the executable's packages (grouped by distributions and architecture). Given the supported distributions and architectures, the database is initialised and updated periodically by downloading the packages' lists from their official repositories. Alternatively, the database can be updated with release information for components that do not come from public repositories. Additionally, the database stores the history of each package, reporting the information about its updates (e.g. the type of update). With respect to the PoC, all compute hosts and containers are based on the CentOS distribution. We have integrated the Trust Monitor with the OSM orchestration tool, which implements a reference architecture of the ETSI NFV MANO stack. In this regard, integration activities have considered different OSM releases, from 2.0 to 5.0. Moreover, we have developed its Audit Database as a client to Hadoop Distributed File System (HDFS) [44], a distributed file-system built for scalability and high availability applications that is very popular for Machine Learning applications. The Docker-enabled VIM is represented by the VIM-Emulator project [94], a component that is implemented in the OSM solution as a container-based emulator for the OpenStack APIs. Other SECaaS components have been developed from scratch in the scope of the SHIELD project. For the integration of the Trust Monitor, the VNF Store persists the attestation data retrieved from the *security manifest* to provide it at the integrity verification phase. This manifest consists of a series of digests (computed via cryptographic hash algorithms, such as SHA-1, and corresponding paths of files that are measured inside each Virtual Deployment Unit (VDU), i.e. each container that belongs to a specific VNF. An example of the JSON security manifest is reported in Figure 5.9. From the network perspective, the Trust Monitor requires network access to the NFVI (compute nodes and switches, as they are the

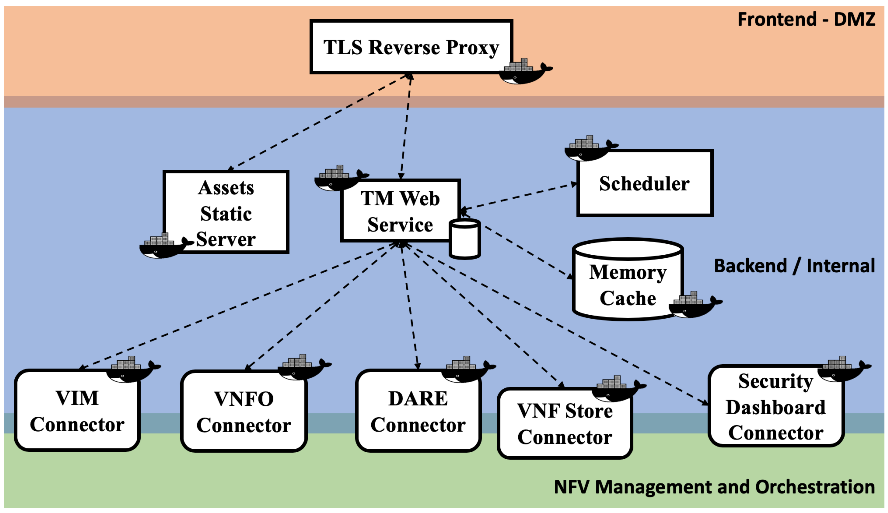

Figure 5.10: Multi-container deployment of the Trust Monitor application

target of attestation) and to the other SECaaS components (for proper integration in the platform).

The Trust Monitor application is deployed as a multi-container system which leverages the Docker Compose tool [21]. This is a tool that allows to specify multi-container applications in a declarative approach, by leveraging a descriptor file to reference all the services, networks and persistent data volumes that are required by the application. In order to install the application, the target environment should have both the Docker container runtime and Docker Compose installed. Installation of dependencies, setup of the different sub-components and deployment of the full-fledged Trust Monitor application require minimal interaction of the user thanks to the automation of the whole process via the Docker Compose tool. Figure 5.10 shows an architectural picture of the multi-container deployment of the Trust Monitor and separation in different tiers, belonging to different domains: the reverse proxy is the only component exposed on the frontend (ideally in a DeMilitarized Zone (DMZ) that is segmented from the internal network and protected by a firewall); the periodic attestation Scheduler, the memory cache, the Trust Monitor web service and the static assets server are deployed on an internal backend domain; finally, the Connectors are exposed on the MANO domain so that they can exchange information with the NFVO, the VIM, the VNF Store and the other SECaaS components (namely the DARE and Security Dashboard).

5.6 Experimental evaluation

We have tested the Trust Monitor prototype in the scope of the SHIELD testbed, comprising two different NFV PoPs on separate geographical locations interconnected via a site-to-site VPN. In this context, both TPM-equipped compute hosts and SDN switches have been targeted for attestation. With respect to the computing platform, the OAT [46] and CIT [69] frameworks have been adopted for the implementation of separate Attestation Drivers and Attesters, the first supporting TPM 1.2 and container attestation (following the technique described in Chapter 4), the latter supporting TPM 2.0 and run-time integrity verification of host only. Both the OAT and CIT Verifiers run on VMs equipped with a dual-core CPU at 2 GHz and 2 GiB of RAM. Both computing hosts have been provisioned with the Linux CentOS 7.5.1804 server distribution, Linux kernel 4.4.19 and with the IMA module enabled and configured for run-time measurement of software packages run in the OS. The software repositories of CentOS 7.5.1804 have been used as a reference white-list of known good software in the Whitelist Database. In this regard, both binaries and their dependencies (e.g. shared libraries) have been measured by computing their SHA-1 digest and storing it, alongside their pathname, in the reference database. Moreover, the container-based compute node leverages a custom version of the OSM VIM-Emulator that we ported to the CentOS distribution, as it only supports the Ubuntu OS by default. Finally, the Trust Monitor core application and the Whitelist Database are run on separate VMs equipped with a dual-core CPU at 2 GHz and 2 GiB of RAM.

The evaluation of our solution is focused on the run-time VNF attestation functionality, which represents the core contribution of this dissertation to the state of the art. Because of this, the rest of the discussion focuses on the interaction between the Trust Monitor, the OAT Attestation Driver, and the OAT Attester for TPM 1.2. With respect to performance, three separate metrics have been considered: the CPU utilisation and RAM consumption at the Trust Monitor side, in order to evaluate its scalability at the increase of the number of VNFs and uptime, and the latency of the whole NFVI attestation process, so to understand whether the Trust Monitor can effectively respond to incoming threats in a reasonable time. We have evaluated the performance by repeating each test for a number of times (10) and by averaging the results in order to reduce the impact of outliers. The plot depicted in Figure 5.11 reports the latency of the VNF attestation process at the increase of the number of containers on the same network service. In fact, a container-based network service is expected to run a significant number of containers, i.e. VDUs, each executing a specific function within the VNF. The highest number of concurrent VNFs in the experiment is 35, as higher values rendered the OSM orchestrator not responsive in the test environment. Moreover, we observed that OSM kills the network service as it saturates the available memory on the

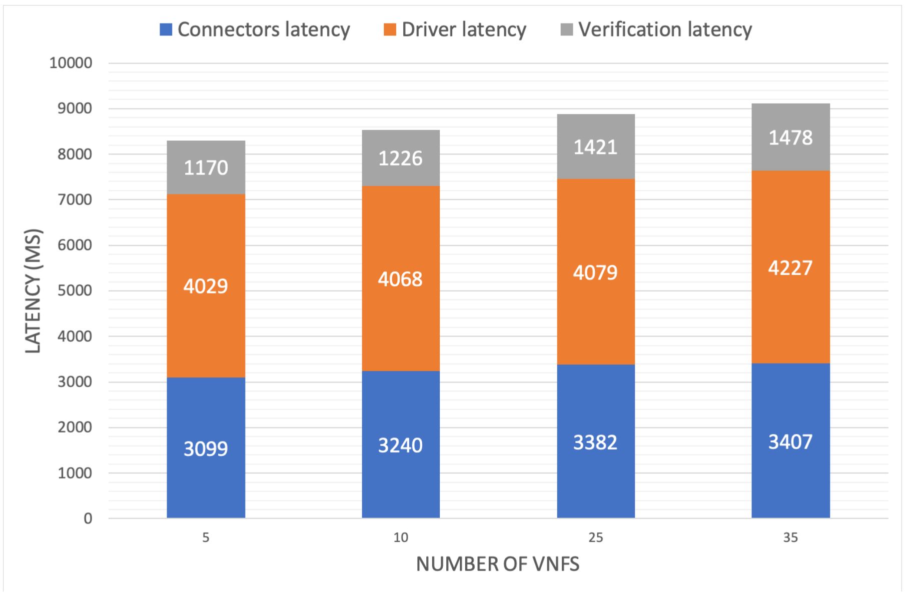

Figure 5.11: Latency of the Trust Monitor verification process

compute host. In order to implement this test, four different network services have been implemented with a variable number of VNFs, from 5 to 35. As shown, the overall latency is composed of three different components:

- Connectors latency. Average time spent by Connectors to interact with the other NFV components (both to request and to send data).
- Driver latency. Average time required by the OAT framework to provide an IR to the Trust Monitor and to verify its authenticity. This includes the client-server communication between the OAT Attestation Server and the OAT HostAgent and the IR signature verification.
- Verification latency. Average time required by the Cloud Verifier to process the Attestation Driver result and to verify VNF measurements against the Whitelist Database and additional data available from the security manifest in the VNF Store.

The plot shows that the overall attestation latency grows less than linearly at the increase of the number of VNFs. The Connector component is affected by the increase of VNF data that is retrieved by both the NFVO and VNF Store. The Attestation Driver latency is not as affected by the number of containers, as the signature verification is performed over the same amount of data (the PCRs), regardless of the number of containers. Nonetheless, the IR validation time increases as the IMA log includes more entries at the increase of the number of containers. Finally, the verification logic also increases as the Cloud Verifier needs to

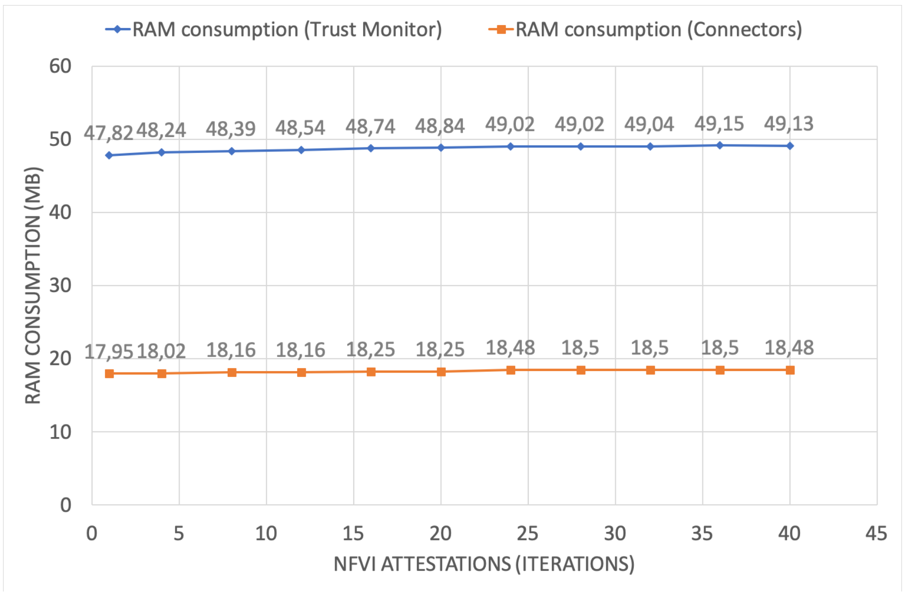

Figure 5.12: RAM consumption of the Trust Monitor at subsequent attestations

verify a larger number of measurements against the data available in the Whitelist Database. In this regard, the use of a key-value memory cache such as Redis aims to reduce the overall latency with respect to retrieving the full white-list from a relational SQL database, which is typically slower for managing a huge number of small queries.

The plots in Figure 5.12 and Figure 5.13 depict the average impact of attestations on the Trust Monitor application. We have monitored the Trust Monitor RAM utilisation during its life-cycle, showing that subsequent attestations do not affect the overall requirements of both the Trust Monitor core application and the Connectors in a significant way. This behaviour shows the mostly stateless nature of the Trust Monitor, which does not keep internal records of subsequent attestations as it mandates the long-term storage of integrity proofs to external databases. This is particularly significant in a long-lived cloud deployment, as it allows the Trust Monitor to have a consistent behaviour and resource utilisation when deployed in a production environment. In order to achieve this result, we have had to engineer the application runtime to explicitly call the Python garbage collector [41] to dispose of unnecessary memory (e.g. dangling pointers) during the application's life-cycle. In fact, unlike other languages such as Java, Python does not automatically release memory to the OS. With respect to CPU usage, Figure 5.13 shows the behaviour of both the Trust Monitor core logic and the Connectors ecosystem at the increase of the number of VNFs targeted for verification. It is shown that the average CPU required for the RA process via the DIVE Attestation Driver increases less than linearly at the increase of attestation targets, as the Cloud Verifier verifies each

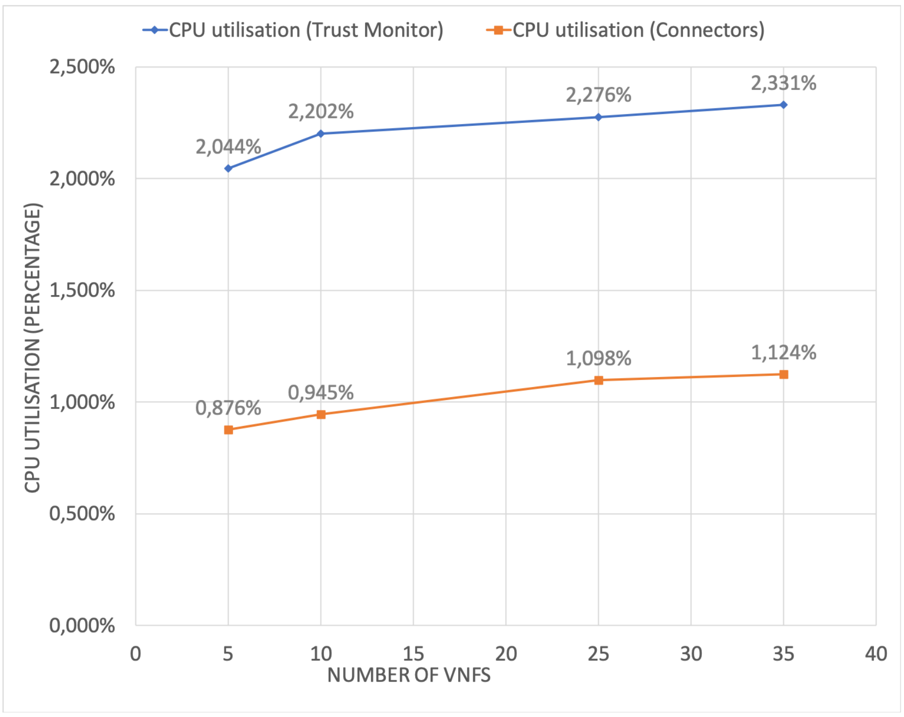

Figure 5.13: CPU utilisation of the Trust Monitor with a variable number of VNFs target sequentially.

Chapter 6

Conclusions and future work

The work described in this dissertation aimed at the definition of an integrity verification solution tailored for the NFV platform and based on TC principles and mechanisms, so to tackle the security issues of softwarised network infrastructures. In fact, given the high impact of privacy on telco network providers' services, security and trust in the platform that delivers those services is considered paramount in order to effectively exploit the NFV paradigm in production environments.

The general approach of the work was to define an abstract architecture for trust assurance that suits heterogeneous attestation mechanisms, i.e. the Trust Monitor. This defines the key principles of a generic integrity verification workflow within the NFV domain, specifying the high level interactions between the Trust Monitor core logic and external actors without discriminating on the practical trust assurance technique, which is left to the implementation. The architecture is designed by ensuring that its building blocks are clearly separated so to ensure both modularity and easier customisation by any implementer (e.g. a telco operator or CSP). In this regard, the solution is generic enough to be exploited in a more generic cloud environment, although NFV is considered as a target use case and motivates the work. The security of this solution relies in the utilisation of well-known attestation models for integrity verification: this flexibility allows the telco operator to rely on both hardware-based approaches, as envisioned by TC and practically implemented via TPMs or other proprietary TEEs (e.g. SGX, SEV), and software-only mechanisms, so that it could be exploited even in particular use cases wherein hardware-level protection is not possible. The flexible design of the Trust Monitor is therefore considered a key advantage of this solution when compared to existing approaches from scientific literature.

Then, a practical approach to VNF attestation was investigated by this dissertation so to enrich the proposed architecture with a viable solution to secure the NFV platform. In this regard, DIVE allows the services running in lightweight virtualised instances, i.e. containers, to be attested as if they were running on a physical compute platform. We have focused on containers as they are considered a more efficient form of virtualisation with respect to VMs, given their limited memory footprint and sharing of host kernel resources. Thanks to DIVE, a cloud attestation service can ensure that integrity reports produced by container-enabled hosts are protected against remote attacks (e.g. MitM) via a tamper-resistant cryptographic device, the TPM. Given its limited performance impact on the hosted services, DIVE represents a practical solution to VNF attestation that does not collide with the low latency and high throughput container applications, as required by the NFV paradigm. When integrated into a cloud scenario, DIVE allows a centralised verifier to identify which container or physical system has been compromised by running a limited set of attestations, as each integrity report provides a full picture regarding each host and all the containers that are running on it. This simplifies the verification logic and reduces the overall number of messages that are exchanged on the network. Moreover, the solution is transparent to the containers themselves, making it readily applicable to an NFV platform by applying a limited set of modifications to the host system. No changes are required on the VNF level, hence complex network services can be attested without any modifications on their side.

As final part of this dissertation, a concrete NFV threat mitigation process was derived by coupling the abstract architecture with the practical VNF attestation technique. Several threats were considered, ranging from the physical attacks to software-based malicious actions, both locally and remotely. At this stage, the research focused on the definition of workflows that could effectively integrate the Trust Monitor into an existing NFV infrastructure, with particular attention to the SECaaS use case. This is considered by ETSI as one of the most critical NFV scenarios with respect to security, as it proposes the deployment of vNSFs as virtualised security appliances (e.g. firewalls, intrusion detection and prevention systems, honeypots) within the ISP client network. The validation of this activity was performed in the scope of the SHIELD European project, wherein a PoC of the Trust Monitor was developed from scratch together with the implementation of an open-source DIVE prototype for the NFV domain. In this regard, integration of the proposed architecture with the reference NFV orchestration framework, i.e. OSM, was also performed so to propose a practical use case for NFV integrity verification as close to off-the-shelf solutions as possible. Moreover, the SECaaS components developed within the SHIELD project were also considered for integration with the Trust Monitor to enhance its visibility over the platform and enrich its attestation capabilities.

Concluding, integrity verification by means of remote attestation is a valuable solution for modern ICT infrastructures that exploit large-scale virtualisation, such as NFV. The main challenge towards its exploitation is represented by the large number of target hardware architectures and virtualisation techniques that are available on the market. Because of this, cloud attestation architectures need to acknowledge this diversity and propose generic methods to verify heterogeneous target platforms. In this regard, an even more significant challenge is the application of hardware-based attestation solutions, such as the TPM-based scheme proposed by TC, to virtualised instances. The presented solution consists of a generic trust assurance architecture that is independent from the underlying hardware architecture, and can be applied to different attestation schemes. With respect to TPM-based attestation, the proposed technique enables run-time integrity verification of software run within containers and, hence, it can be readily applied to a lightweight NFV platform. Moreover, the proposed solution can be integrated into even more complex security solutions because of its modular architecture and external interfaces, and could be even generalised to cloud deployments that are different from NFV. Limitations of our proposal are in the non-negligible development efforts of integrating heterogeneous Attestation Drivers to the core application and in the maintenance cost of updating and securing the Whitelist Database, whose integrity is fundamental to ensure that verification is performed correctly.

Future directions of this work may target either MANO attestation (which is considered as a minimal upgrade over the proposed solution, from the technical perspective) or the extension of integrity verification to heterogeneous virtualisation mechanisms. In this regard, ongoing research activities are targeting VM-based attestation by means of a vTPM. Compared to existing approaches from literature, we plan on leveraging a strong binding between the physical TPM and the different instances of vTPMs so that their persistent state and most security-critical elements (i.e. primary keys, sealed objects) are stored by the pTPM. Another extension of the work would target the IoT domain by proposing a software-only attestation mechanism that could allow verification of hardware-constrained devices in IoT fleets. Another challenge related to IoT exploitation of the Trust Monitor is the deployment model: compared to a standard cloud domain, IoT platforms rely on a large number of decentralised, heterogeneous nodes that are typically located at the edge of the network. Because of this, a distributed deployment of the Trust Monitor could be considered as well, leveraging separate Attestation Drivers deployed on the edge of the network and a centralised workflow manager that orchestrates their activities and white-lists.

Acronyms

AAA Authentication, Authorisation and Accounting. 19, 32

AK Attestation Key. 24, 67

API Application Programming Interface. 35, 45–47, 59, 67, 68, 70, 73, 74, 79, 82

CA Certification Authority. 23, 67

CDN Content Delivery Network. 14

CIT Cloud Integrity Technology. 53, 59, 84

CRTM Core Root of Trust for Measurements. 24, 44, 67

CSP Cloud Service Provider. 2, 4, 13, 16, 26, 29, 30, 32, 34, 43–47, 71, 73, 89

DAA Direct Anonymous Attestation. 23

DARE Data Analysis and Remediation Engine. 30, 71, 72, 76, 78, 79, 83

DCT Docker Content Trust. 53

DIVE Docker Integrity Verification. 51–54, 56, 59, 61, 62, 64, 66–68, 86, 89, 90

DMZ DeMilitarized Zone. 83

ECC Elliptic Curves Cryptography. 24

EK Endorsement Key. 23

ESB Enterprise Service Bus. 48

ETSI European Telecommunications Standards Institute. 3, 7, 8, 11–14, 16, 18, 22, 27, 30, 32, 69, 82, 90

GLANF Glasgow Network Functions. 17

GUI Graphical User Interface. 79

HDFS Hadoop Distributed File System. 82

IaaS Infrastructure as a Service. 2, 33, 71

ICT Information and Communication Technology. 1, 3, 4, 11, 14, 29, 30, 76, 90

IMA Integrity Measurement Architecture. 24, 25, 54–62, 64–67, 72, 84, 85

IoT Internet of Things. 3, 27, 91

IPC Inter Process Communication. 15

IR Integrity Report. 22, 24, 35, 39, 40, 45, 52, 56, 59, 65–67, 85

ISG Industry Specification Group. 11, 18

ISP Internet Service Provider. 3–5, 12–14, 25, 26, 30–32, 70, 71, 78, 90

JSON JavaScript Object Notation. 79, 81, 82

KVM Kernel-based Virtual Machine. 17

LAN Local Area Network. 34

LSM Linux Security Modules. 56, 57, 64

LXC Linux Container. 56, 57

MANO Management and Orchestration. 12, 13, 16, 18, 19, 25, 27, 70, 82, 83, 91

MitM Man-in-the-Middle. 21, 90

MLE Measured Launch Environment. 67

NFV Network Functions Virtualisation. 3–8, 11–14, 16–22, 25–27, 29–33, 36, 42, 47, 52, 69–73, 76, 78, 82, 84, 85, 89–91

NFVI NFV Infrastructure. 12, 13, 18, 19, 25, 27, 70–72, 76, 79, 82, 84

NFVO Network Functions Virtualisation Orchestrator. 13, 47, 70–76, 79, 83, 85

NIST National Institute of Standards and Technology. 20

NVRAM Non-Volatile Random Access Memory. 23

OAT Open ATtestation. 43, 59, 67, 84, 85

OEM Original Equipment Manufacturer. 67

OS Operating System. 1, 2, 5, 6, 15, 21, 24, 38, 41, 42, 45, 54, 67, 84, 86

OSM Open Source MANO. 16, 21, 82, 84, 90

PaaS Platform as a Service. 2

PCR Platform Configuration Register. 23, 24, 44, 54–56, 59, 66–68, 85

PoC Proof-of-Concept. 59, 61, 78, 82, 90

PoP Point of Presence. 71, 84

RA Remote Attestation. 5, 6, 22, 24–26, 33–38, 40, 41, 43, 44, 52, 53, 56, 59, 65, 67, 76, 86

RAN Radio Access Network. 14

REST Representational State Transfer. 45, 59, 79

RoT Root of Trust. 5, 18, 22–25, 35, 37, 43, 45, 51, 52, 56, 70

RTM Root of Trust for Measurements. 24

RTR Root of Trust for Reporting. 24

RTS Root of Trust for Storage. 24

SaaS Software as a Service. 2

SDN Software-Defined Networking. 3, 13, 17, 18, 25, 26, 79, 84

SECaaS Security as a Service. 13, 30, 31, 42, 69, 71, 75, 76, 78, 79, 82, 83, 90

SEV Secure Encrypted Virtualisation. 25, 26, 89

SGX Software Guard Extensions. 25, 26, 89

SIEM Security Information and Event Management. 32, 76

TC Trusted Computing. 5–7, 18, 22, 24, 25, 27, 37, 40, 44, 45, 51, 72, 89, 91

TCG Trusted Computing Group. 5, 24, 52, 54, 56, 59, 67

TEE Trusted Execution Environment. 6, 7, 25–27, 35, 89

TP Trusted Platform. 24

TPM Trusted Platform Module. 5–7, 22–26, 44, 52, 54, 56, 59, 62, 65, 67, 68, 70, 72, 77, 79, 84, 89–91

TTP Trusted Third-Party. 22, 24, 34

TZ TrustZone. 25

UEFI Unified Extensible Firmware Interface. 22, 24

UTS UNIX Time Sharing. 15

UUID Universal Unique IDentifier. 57–61

VDU Virtual Deployment Unit. 82, 84

VIM Virtualised Infrastructure Manager. 13, 21, 70, 73–75, 79, 82, 83

VLAN Virtual Local Area Network. 18, 34

VM Virtual Machine. 2, 14, 15, 17, 21, 25, 45, 53, 54, 56, 62, 65, 84, 90, 91

VNF Virtual Network Function. 3–5, 7, 8, 12–14, 17–20, 22, 25–27, 32, 40, 42, 43, 51, 52, 66, 67, 69–75, 79–82, 84–86, 89, 90

vNSF virtual Network Security Function. 18, 30, 31, 72, 78, 79, 90

VPN Virtual Private Network. 18, 84

vTPM virtual Trusted Platform Module. 25, 45, 91

Bibliography

[1] “AMD Secure Encrypted Virtualization project website”. URL: https://developer.amd.com/sev/

[2] J. Anderson, H. Hu, U. Agarwal, et al. “Performance considerations of network functions virtualization using containers”. In: *Int. Conf. on Computing, Networking and Communications (ICNC)*. Kauai (HI USA), Feb. 2016, pp. 1–7. DOI: 10.1109/ICCNC.2016.7440668

[3] “Apache Cassandra project website”. URL: http://cassandra.apache.org

[4] Apple. “About the Apple T2 Security Chip”. URL: https://support.apple.com/en-us/HT208862

[5] “ARM TrustZone project website”. URL: https://www.arm.com/products/security-on-arm/trustzone

[6] AT&T. “Building an Enterprise-Focused Cloud Environment”. Oct. 2017. URL: http://about.att.com/innovationblog/enterprise_cloud

[7] S. Berger, R. Cáceres, K. A. Goldman, et al. “vTPM: Virtualizing the Trusted Platform Module”. In: *15th USENIX Security Symposium*. Vancouver (Canada): USENIX Association, July 2006, pp. 305–320. DOI: 10.5555/1267336.1267357

[8] A. Bettini. “Vulnerability exploitation in Docker container environments”. Tech. rep. FlawCheck, 2015. URL: https://www.blackhat.com/docs/eu-15/materials/eu-15-Bettini-Vulnerability-Exploitation-In-Docker-Container-Environments-wp.pdf

[9] R. Bonafiglia, I. Cerrato, F. Ciaccia, et al. “Assessing the Performance of Virtualization Technologies for NFV: A Preliminary Benchmarking”. In: *2015 Fourth European Workshop on Software Defined Networks*. Bilbao (Spain), Sept. 2015. DOI: 10.1109/EWSDN.2015.63

[10] E. Brickell, J. Camenisch, and L. Chen. “Direct Anonymous Attestation”. In: *11th ACM Conference on Computer and Communications Security*. CCS ’04. Washington DC (USA): ACM, Oct. 2004, pp. 132–145. DOI: 10.1145/1030083.1030103

[11] J. Chelladhurai, P. R. Chelliah, and S. A. Kumar. “Securing Docker Containers from Denial of Service (DoS) Attacks”. In: *IEEE Int. Conf. on Services Computing (SCC).* San Francisco (CA USA), June 2016, pp. 856–859. DOI: `10.1109/SCC.2016.123`

[12] G. Coker, J. Guttman, P. Loscocco, et al. “Principles of remote attestation”. In: *International Journal of Information Security* 10.2 (June 2011), pp. 63–81. DOI: `10.1007/s10207-011-0124-7`

[13] M. Coughlin, E. Keller, and E. Wustrow. “Trusted Click: Overcoming Security Issues of NFV in the Cloud”. In: *ACM International Workshop on Security in Software Defined Networks and Network Function Virtualization.* Scottsdale (AZ USA), Mar. 2017, pp. 31–36. DOI: `10.1145/3040992.3040994`

[14] R. Cziva, S. Jouet, K. J. S. White, et al. “Container-based network function virtualization for software-defined networks”. In: *IEEE Symposium on Computers and Communication (ISCC).* Larnaca (Cyprus), July 2015, pp. 415–420. DOI: `10.1109/ISCC.2015.7405550`

[15] R. Cziva and D. P. Pezaros. “Container network functions: bringing NFV to the network edge”. In: *IEEE Communications Magazine* 55.6 (June 2017), pp. 24–31. DOI: `10.1109/MCOM.2017.1601039`

[16] Datadog. “8 surprising facts about real Docker container adoption”. 2017. URL: `https://www.datadoghq.com/docker-adoption/`

[17] M. De Benedictis and A. Lioy. “A proposal for trust monitoring in a Network Functions Virtualisation Infrastructure”. In: *2019 IEEE Conference on Network Softwarization (NetSoft).* Paris (France), June 2019, pp. 1–9. DOI: `10.1109/NETSOFT.2019.8806655`

[18] M. De Benedictis and A. Lioy. “Integrity verification of Docker containers for a lightweight cloud environment”. In: *Future Generation Computer Systems* 97 (Aug. 2019), pp. 236–246. DOI: `10.1016/j.future.2019.02.026`

[19] M. De Benedictis and A. Lioy. “On the establishment of trust in the cloud-based ETSI NFV framework”. In: *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN).* Berlin (Germany), Nov. 2017, pp. 280–285. DOI: `10.1109/NFV-SDN.2017.8169864`

[20] “Django REST Framework website”. URL: `https://www.django-rest-framework.org`

[21] “Docker Compose project website”. URL: `https://docs.docker.com/compose/`

[22] “Docker Content Trust description”. URL: `https://docs.docker.com/engine/security/trust/content_trust/`

[23] "Docker project website". URL: https://www.docker.com/

[24] P. England and J. Loeser. "Para-Virtualized TPM Sharing". In: *Trusted Computing - Challenges and Applications*. Ed. by P. Lipp, A.-R. Sadeghi, and K.-M. Koch. Villach (Austria): Springer, Mar. 2008, pp. 119–132. DOI: https://doi.org/10.1007/978-3-540-68979-9_9

[25] ETSI. "Network Function Virtualisation - An Introduction, Benefits, Enablers, Challenges and Call for Action". Oct. 2012. URL: https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[26] ETSI. "Network Function Virtualisation - Network Operator Perspectives on NFV priorities for 5G". Feb. 2017. URL: https://portal.etsi.org/NFV/NFV_White_Paper_5G.pdf

[27] "ETSI Network Function Virtualisation (NFV) website". URL: https://www.etsi.org/technologies-clusters/technologies/nfv

[28] ETSI NFV ISG. "ETSI GS NFV 001 Network Functions Virtualisation (NFV); Use Cases". Tech. rep. May 2017. URL: https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV%20001v1.2.1%20-%20GR%20-%20NFV%20Use%20Cases%20revision.pdf

[29] ETSI NFV ISG. "ETSI GS NFV 002 Network Functions Virtualisation (NFV); Architectural Framework". Tech. rep. Dec. 2014. URL: https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV%20002v1.2.1%20-%20GS%20-%20NFV%20Architectural%20Framework.pdf

[30] ETSI NFV ISG. "ETSI GS NFV-EVE 004 Network Functions Virtualisation (NFV); Virtualisation Technologies; Report on the application of Different Virtualisation Technologies in the NFV Framework". Tech. rep. Mar. 2016. URL: https://docbox.etsi.org/ISG/NFV/Open/%5C%5CPublications_pdf/Specs-Reports/NFV-EVE%5C%20004v1.1.1%5C%20-%5C%20GS%5C%20-%5C%20Virtualisation%5C%20technologies%5C%20Report.pdf

[31] ETSI NFV ISG. "ETSI GS NFV-SEC 001 Network Functions Virtualisation (NFV); NFV Security; Problem Statement". Tech. rep. Oct. 2014. URL: https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-SEC%20001v1.1.1%20-%20GS%20-%20Security%20Problem%20Statement.pdf

[32] ETSI NFV ISG. "ETSI GS NFV-SEC 003 Network Functions Virtualisation (NFV); NFV Security; Security and Trust Guidance". Tech. rep. Aug. 2016. URL: https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-SEC%20003v1.2.1%20-%20GR%20-%20Security%20and%20Trust%20Guidance.pdf

[33] ETSI NFV ISG. “ETSI GS NFV-SEC 007 Network Functions Virtualisation (NFV); Trust; Report on Attestation Technologies and Practices for Secure Deployments”. Tech. rep. Oct. 2017. URL: `https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-SEC%20007v1.1.1%20-%20GR%20-%20NFV%20Attestation%20report.pdf`

[34] ETSI NFV ISG. “ETSI GS NFV-SEC 014 Network Functions Virtualisation (NFV) Release 3; NFV Security; Security Specification for MANO Components and Reference points”. Tech. rep. Apr. 2018. URL: `https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-SEC%20014v3.1.1%20-%20GS%20-%20MANO%20Security%20Spec.pdf`

[35] ETSI NFV ISG. “ETSI GS NFV-SEC 021 Network Functions Virtualisation (NFV) Release 2; NFV Security; VNF Package Security Specification”. Tech. rep. June 2019. URL: `https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-SEC%20021v2.6.1%20-%20GS%20-%20VNF%20Package%20Security%20Spec.pdf`

[36] ETSI NFV ISG. “ETSI GS NFV-SWA 001 Network Functions Virtualisation (NFV); Virtual Network Functions Architecture”. Tech. rep. Dec. 2014. URL: `https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-SWA%20001v1.1.1%20-%20GS%20-%20Virtual%20Network%20Function%20Architecture.pdf`

[37] “Exploring Opportunities: Containers and OpenStack”. URL: `https://www.openstack.org/assets/pdf-downloads/Containers-and-OpenStack.pdf`

[38] I. Faynberg and S. Goeringer. “NFV Security: Emerging Technologies and Standards”. In: *Guide to Security in SDN and NFV: Challenges, Opportunities, and Applications*. Springer International Publishing, Nov. 2017, pp. 33–73. DOI: `10.1107/978-3-319-64653-4_2`

[39] R. T. Fielding. “Architectural Styles and the Design of Network-based Software Architectures”. PhD thesis. University of California, 2000. URL: `http://roy.gbiv.com/pubs/dissertation/top.htm`

[40] “Flask project website”. URL: `https://palletsprojects.com/p/flask/`

[41] A. Golubin. “Garbage collection in Python: things you need to know”. URL: `https://rushter.com/blog/python-garbage-collector/`

[42] Google. “Titan M makes Pixel 3 our most secure phone yet”. URL: `https://www.blog.google/products/pixel/titan-m-makes-pixel-3-our-most-secure-phone-yet/`

[43] A. Grattafiori. “NCC Group Whitepaper - Understanding and Hardening Linux Containers”. Tech. rep. NCC Group, June 2016. URL: `https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/april/%5C%5Cncc_group_understanding_hardening_linux_containers-1-1.pdf`

[44] “HDFS Architecture Guide”. URL: `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`

[45] “Integrity Measurement Architecture project website”. URL: `https://sourceforge.net/p/linux-ima/wiki/Home/`

[46] “Intel OpenAttestation project website”. URL: `https://github.com/OpenAttestation/`

[47] “Intel OpenCIT project website”. URL: `https://01.org/opencit`

[48] “Intel Software Guard Extensions project website”. URL: `https://software.intel.com/en-us/sgx`

[49] L. Jacquin, A. Lioy, D. R. Lopez, et al. “The Trust Problem in Modern Network Infrastructures”. In: *Cyber Security and Privacy.* Brussels (Belgium): Springer, Nov. 2015, pp. 116–127. DOI: `10.1007/978-3-319-25360-2_10`

[50] B. Jaeger. “Security Orchestrator: Introducing a Security Orchestrator in the Context of the ETSI NFV Reference Architecture”. In: *TRUSTCOM’15: 14th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications - Vol. 1.* Helsinki (Finland), Aug. 2015, pp. 1255–1260. DOI: `10.1109/Trustcom.2015.514`

[51] “Keylime project website”. URL: `https://keylime.dev`

[52] “Kubernetes project website”. URL: `https://kubernetes.io/`

[53] S. Lal, A. Kalliola, I. Oliver, et al. “Securing VNF communication in NFVI”. In: *IEEE Conf. on Standards for Communications and Networking (CSCN).* Helsinki (Finland), Sept. 2017, pp. 187–192. DOI: `10.1109/CSCN.2017.8088620`

[54] S. Lal, T. Taleb, and A. Dutta. “NFV: Security Threats and Best Practices”. In: *IEEE Communications Magazine* 55.8 (Aug. 2017), pp. 211–217. DOI: `10.1109/MCOM.2017.1600899`

[55] V. Lefebvre, G. Santinelli, T. Müller, et al. “Universal Trusted Execution Environments for Securing SDN/NFV Operations”. In: *Proceedings of the 13th International Conference on Availability, Reliability and Security.* Hamburg (Germany): ACM, Aug. 2018, 44:1–44:9. DOI: `10.1145/3230833.3233256`

[56] “Libvirt - KVM/QEMU hypervisor driver”. URL: `https://libvirt.org/drvqemu.html`

[57] LightReading. “Telefonica Unveils Aggressive NFV Plans”. Feb. 2014. URL: https://www.lightreading.com/carrier-sdn/sdn-technology/telefonica-unveils-aggressive-nfv-plans/d/d-id/707882

[58] “Linux Containers project website”. URL: https://linuxcontainers.org/

[59] “Linux Programmer’s Manual - Capabilities”. URL: http://man7.org/linux/man-pages/man7/capabilities.7.html

[60] “Linux Programmer’s Manual - Cgroups”. URL: http://man7.org/linux/man-pages/man7/cgroups.7.html

[61] “Linux Programmer’s Manual - Namespaces”. URL: http://man7.org/linux/man-pages/man7/namespaces.7.html

[62] A. Lioy, G. Gardikis, B. Gaston, et al. “NFV-based network protection: The SHIELD approach”. In: *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. Berlin (Germany), Nov. 2017, pp. 1–2. DOI: 10.1109/NFV-SDN.2017.8169869

[63] “LXD project website”. URL: https://linuxcontainers.org/lxd/

[64] A. Martin, S. Raponi, T. Combe, et al. “Docker ecosystem – Vulnerability Analysis”. In: *Computer Communications* 122 (June 2018), pp. 30–43. DOI: 10.1016/j.comcom.2018.03.011

[65] B. C. News. “Deutsche Telekom experimenting with NFV in Docker”. Feb. 2015. URL: https://telecoms.com/397152/deutsche-telekom-experimenting-with-nfv-in-docker/

[66] “Nginx project website”. URL: https://www.nginx.com/

[67] M. Souppaya, J. Morello, and K. Scarfone. “NIST Special Publication 800-190 - Application Container Security Guide”. Tech. rep. NIST, Sept. 2017. DOI: 10.6028/NIST.SP.800-190

[68] “Open Baton project website”. URL: https://openbaton.github.io/

[69] “Open Cloud Integrity Technology (Open CIT) project website”. URL: https://01.org/opencit

[70] “Open Source MANO project website”. URL: https://osm.etsi.org/

[71] “Open vSwitch project website”. URL: https://www.openvswitch.org

[72] “OpenStack project website”. URL: https://www.openstack.org/

[73] “OPNFV project website”. URL: https://www.opnfv.org/

[74] R. Poddar, C. Lan, R. A. Popa, et al. “SafeBricks: Shielding Network Functions in the Cloud”. In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton (WA USA): USENIX Association, Apr. 2018, pp. 201–216. URL: https://www.usenix.org/conference/nsdi18/presentation/poddar

[75] “Podman project website”. URL: https://podman.io

[76] M. Raho, A. Spyridakis, M. Paolino, et al. “KVM, Xen and Docker: A performance analysis for ARM based NFV and cloud computing”. In: *2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. Riga (Latvia), Nov. 2015. DOI: 10.1109/AIEEE.2015.7367280

[77] S. Ravidas, S. Lal, I. Oliver, et al. “Incorporating trust in NFV: Addressing the challenges”. In: *ICIN-2017: 20th Conference on Innovations in Clouds, Internet and Networks*. Paris (France), Mar. 2017, pp. 87–91. DOI: 10.1109/ICIN.2017.7899394

[78] “Redis project website”. URL: https://redis.io

[79] “Rkt project website”. URL: https://coreos.com/rkt/

[80] R. Sailer, X. Zhang, T. Jaeger, et al. “Design and Implementation of a TCG-based Integrity Measurement Architecture”. In: *13th USENIX Security Symposium*. San Diego (CA USA): USENIX Association, Aug. 2004. URL: https://www.usenix.org/legacy/publications/library/proceedings/sec04/%5C%5Ctech/full_papers/sailer/sailer.pdf

[81] N. Schear, P. T. Cable II, T. M. Moyer, et al. “Bootstrapping and Maintaining Trust in the Cloud”. In: *ACSAC ’16 Proceedings of the 32nd Annual Conference on Computer Security Applications*. Los Angeles (CA USA): ACM, Dec. 2016, pp. 65–77. DOI: 10.1145/2991079.2991104

[82] SHIELD. “Open-source prototype for Trust Monitor in NFV”. URL: https://github.com/shield-h2020/trust-monitor

[83] M.-W. Shih, M. Kumar, T. Kim, et al. “S-NFV: Securing NFV States by Using SGX”. In: *2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. New Orleans (LA USA), Mar. 2016, pp. 45–48. DOI: 10.1145/2876019.2876032

[84] “Software-Defined Networking (SDN) website”. URL: https://www.opennetworking.org/sdn-definition/

[85] “The JavaScript Object Notion data format”. URL: https://www.json.org/json-en.html

[86] Trusted Computing Group. “EK Credential Profile for TPM Family 2.0; Level 0”. Tech. rep. Dec. 2018. URL: https://trustedcomputinggroup.org/wp-content/uploads/TCG_IWG_Credential_Profile_EK_V2.1_R13.pdf

[87] Trusted Computing Group. "Guidance for Securing Network Equipment Using TCG Technology Version 1.0 Revision 29". Tech. rep. Jan. 2018. URL: https://trustedcomputinggroup.org/wp-content/uploads/TCG_Guidance_for_Securing_NetEq_1_0r29.pdf

[88] Trusted Computing Group. "Integrity Report Schema, Specification Version 2.0, Revision 5". Tech. rep. Aug. 2011. URL: https://trustedcomputinggroup.org/wp-content/uploads/IWG_Integrity_Report_Schema_v2.0.r5.pdf

[89] Trusted Computing Group. "TCG Architecture Overview, Version 1.4". Tech. rep. Aug. 2007. URL: https://trustedcomputinggroup.org/wp-content/uploads/TCG_1_4_Architecture_Overview.pdf

[90] Trusted Computing Group. "TCG FIPS 140-2 Guidance for TPM 2.0". Tech. rep. Feb. 2017. URL: https://trustedcomputinggroup.org/wp-content/uploads/TCG_FIPS_140_Guidance_for_TPM2_0_v1r1_20170202.pdf

[91] Trusted Computing Group. "Trusted Platform Module Library Part 1: Architecture Family 2.0 Revision 1.38". Tech. rep. Sept. 2016. URL: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf

[92] Trusted Computing Group. "Trusted Platform Module Library Part 2: Structures Family 2.0 Revision 1.38". Tech. rep. Sept. 2016. URL: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-2-Structures-01.38.pdf

[93] Trusted Computing Group. "Trusted Platform Module Library Part 3: Commands Family 2.0 Revision 1.38". Tech. rep. Sept. 2016. URL: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.38.pdf

[94] "VIM emulator project website". URL: https://osm.etsi.org/wikipub/index.php/VIM_emulator

[95] "VMWare project website". URL: https://www.vmware.com

[96] X. Wan, Z. Xiao, and Y. Ren. "Building Trust into Cloud Computing Using Virtualization of TPM". In: *2012 Fourth International Conference on Multimedia Information Networking and Security.* Nanjing (China), Nov. 2012, pp. 59–63. DOI: 10.1109/MINES.2012.82

[97] J. Wang, F. Xiao, J. Huang, et al. "A Security-Enhanced vTPM 2.0 for Cloud Computing". In: *ICICS: International Conference on Information and Communications Security.* Beijing, (China): Springer International Publishing, Dec. 2017, pp. 557–569. DOI: https://doi.org/10.1007/978-3-319-89500-0_48

[98] R. Wilkins and B. Richardson. "UEFI Secure Boot in Modern Computer Security Solutions". Tech. rep. Unified Extensible Firmware Interface Forum, Sept. 2013. URL: https://uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf

[99] "Xen project website". URL: https://xenproject.org

[100] Z. Yan, P. Zhang, and A. V. Vasilakos. "A security and trust framework for virtualized networks and software-defined networking". In: *Security and Communication Networks* 9.16 (Mar. 2015), pp. 3059–3069. DOI: 10.1002/sec.1243

[101] R. Yeluri and A. Gupta. "Trusted Docker Containers and Trusted VMs in OpenStack". 2015. URL: https://01.org/sites/default/files/openstacksummit_vancouver_trusteddockercontainers.pdf

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.