SHIELD

SECURING AGAINST INTRUDERS AND OTHER THREATS THROUGH A NFV-ENABLED ENVIRONMENT

[H2020 - Grant Agreement No. 700199]

Deliverable D3.1

Specifications, design and architecture for the vNSF ecosystem

Editor	R. Preto (Ubiwhere)
Contributors	F. Ferreira, T Batista (Ubiwhere), C. Fernández, B. Gastón (i2CAT), M. De Benedictis, A. Lioy (POLITO), E. Trouva (NCSRD), O. Segou (ORION), H. Attak, L. Jacquin (HPE), A. Pastor (TID)

Version	1.0
Date	June 5th, 2017
Distribution	PUBLIC (PU)

Executive Summary

Following the work done in D2.1, where the requirements of the SHIELD platform were elicited and the high-level design and architecture of the platform was exposed, a detailed study of the different components has been done in order to obtain the low-level architecture and design (subcomponent granularity), the specifications (transformation of the user requirements into technical requirements/specifications) and the implementation guide (technologies to use). This work has been divided into the two technical development work packages of SHIELD namely WP3 and WP4. This deliverable covers the components developed for WP3, namely i) the vNSF Store, which holds a registry of NS and vNSF-related information; ii) the vNSF Orchestrator, which deploys and manages the lifecycle of the NSs and vNSFs; iii) the monitoring vNSFs, which produce the information to detect the threats; iv) the remediation vNSFs, which provide the means to actuate and mitigate detected threats sand and v) the Trust Monitor, which verifies that both NSs and vNSFs, as well as other nodes from the infrastructure, are trusted at all times.

One of the main aspects exposed in this deliverable is the transformation of the user requirements into specifications or technical requirements. The requirements identified in D2.1 were categorised into platform requirements, non-functional requirements and service requirements. Firstly, we have identified which of these requirements have implications in every phase and subcomponent (D2.1 already did this work but only at a component level). Secondly, we have translated the requirements from the business language used in D2.1 to the technical language needed for the developments.

This deliverable defines the vNSF architecture blueprint to be used in SHIELD, in which the common elements of a vNSF are defined and the available interfaces depicted. The case of Network Services is covered and we provide an example of a NS built from two vNSF with the intent to stipulate the mandatory internal elements when more than one vNSF is employed, that ensure control and configuration of the entire vNSF set. Both the architecture and the interfaces presented for vNSF comply with the ETSI NFV group recommendations and specifications [1].

This deliverable also details the vNSFs currently envisioned for implementation in SHIELD, specifically, an Intrusion Detection System (IDS), a TLS re-encryption gateway, a traffic analysis vNSF, a deep packet inspection vNSF, a packet filter vNSF acting at network layer, and a forward proxy vNSF acting at application layer.

From the detailed research conducted on previous projects, open source implementations related to NFV technology and state-of-the-art technologies embraced by the community, we have concluded that the Store shall incorporate the VNF and NS descriptors validation work done in the SONATA [2] project; the vNSFO shall build on the work carried out within the OSM project [3]; and the TM makes use of the work from the SECURED [4] project where it reuses the Third-party Verifier based on Open Attestation v1.7 [5], the Whitelist Database based on Apache Cassandra 2 [6] and the SDN-enabled switch attestation prototype [7].

From all the information presented in this deliverable, one can grasp the architecture and design envisaged for the WP3, how the components work internally and how they interact with each other and for which purposes, along with the vNSFs to implement for SHIELD. As a final remark, it is estimated that, during the course of the project, and as a result of the increased

knowledge gathered over time, additional vNSFs may be defined or components tailoring may be required to better frame the project scope and goals.

Table of Contents

1. INTRODUCTION

This document presents the detailed architecture, design and specifications of the components involved in the Virtual Network Security Function (vNSF) ecosystem, within WP3. It summarises the work done in the first iteration of T3.1. This deliverable starts from the high-level architecture, design and requirements presented in D2.1; and provides specific details of the components' design, definition and their adequateness regarding the SHIELD requirements.

SHIELD, as a Use-Case (UC) driven project, aims to cover the functionality required by the following three Use Cases (defined in D2.1 and briefly repeated here for the sake of completeness):

- *Use Case 1:* An Internet Service Provider (ISP) using SHIELD to secure its own infrastructure. This UC involves the ISPs deploying vNSFs in their network to detect security incidents and provide protection against those incidents (Figure 1).

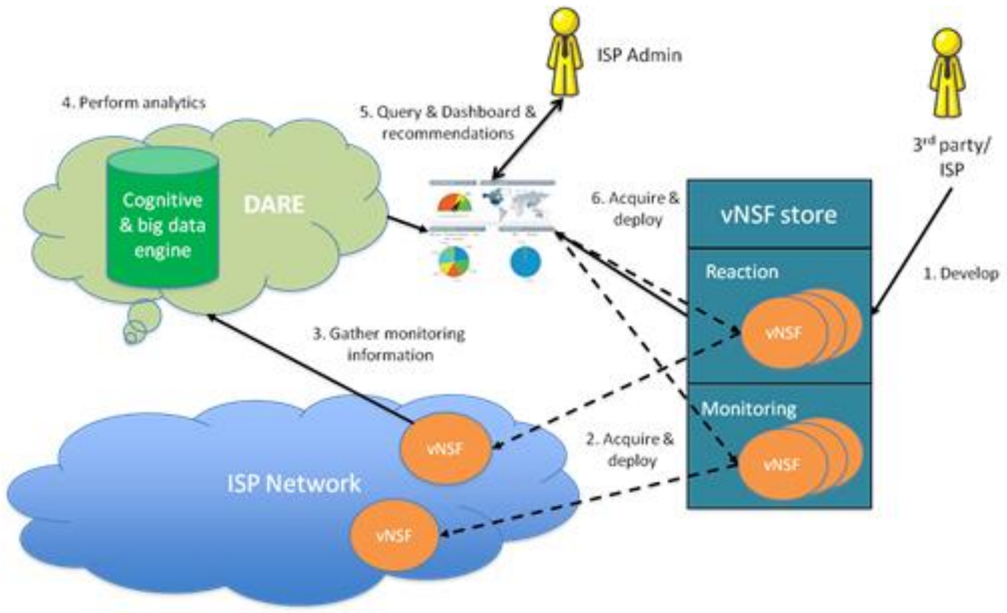


Figure 1: High-level picture of the use case 1

- *Use Case 2:* An ISP is leveraging SHIELD to provide advanced Security as a Service (SecaaS) services to its customers. This UC assumes that network security services (consisting of vNSFs), along with real-time incident detection and mitigation services, are offered as-a-Service to ISP clients, such as enterprises, public bodies, etc. (Figure 2).

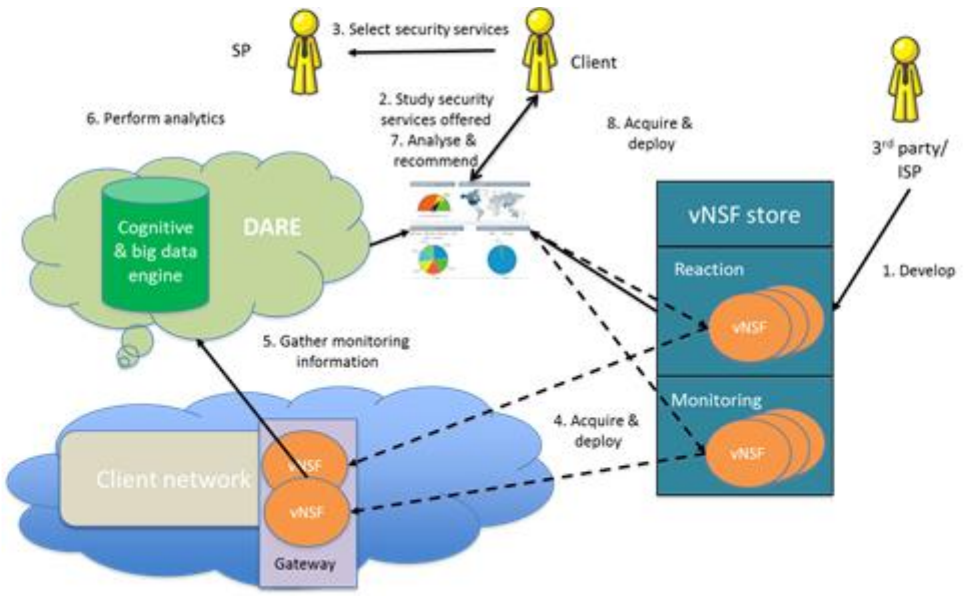


Figure 2: High-level picture of the use case 2

- *Use Case 3:* Contributing to national, European and global security. This UC assumes that incident information is exposed, in a secure and private manner, to public cybersecurity authorities (Figure 3).

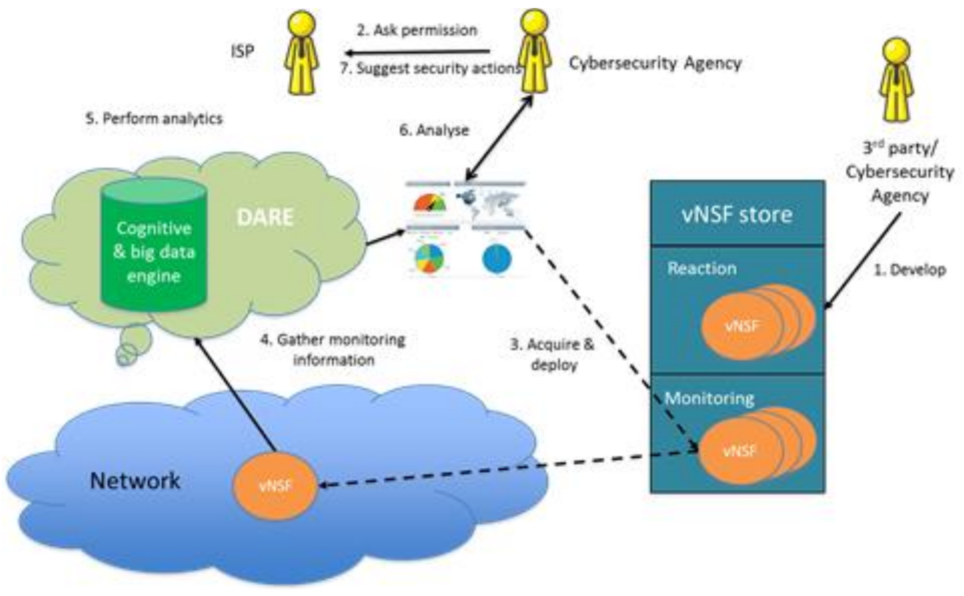


Figure 3: High-level picture of the use-case 3

The high level architecture defined in WP2 and reproduced on Figure 4 states that SHIELD consists of 6 main components; of these, WP3 deals with the vNSF Ecosystem, the vNSF Orchestrator (vNSFO), the Store and the Trust Monitor (TM).

Although the three use-cases act as the basis of the analysis, the resulting architecture, design, specifications and implementation have been elaborated to produce a unified and universal solution i.e. a single cybersecurity solution that can be used for multiple purposes. To this intent, the SHIELD platform provides the actors in the different use-cases with different views and roles on the network. For example, while an ISP (use-case 1) can view the big picture of the cybersecurity analysis and can deploy vNSFs in any location of the network, the ISP client (use-case 2) only has access to a limited vision of the cybersecurity picture (information that is offered by the ISP and/or paid by the client) and can deploy vNSFs in specific places of the network (i.e. to its gateways) to protect their own services. Cybersecurity agencies (use-case 3) have a country- or European-wide security view of the communication infrastructure and the security threats and incidents that take place over this infrastructure, without having access to sensitive information that belongs to ISPs and their clients, which could reveal potential business plans or data.

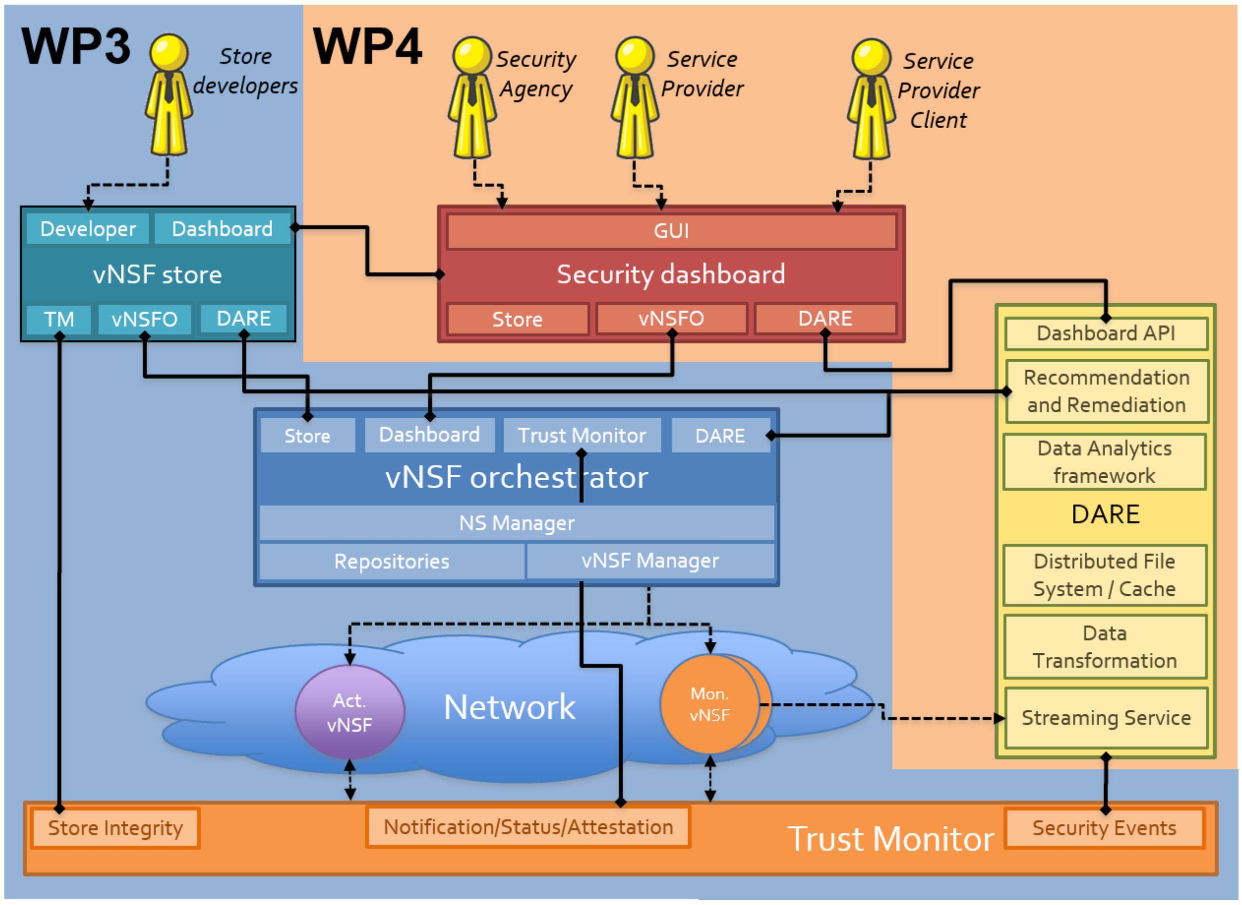


Figure 4: High-level architecture of SHIELD, with components per WP

Based on these use cases and the requirements highlighted in Deliverable D2.1, the designed high-level architecture for the SHIELD platform is articulated around different components, illustrated in Figure 4 and described in more detail in this deliverable. From the point of view of the vNSF environment; the vNSF Store holds a record of Network Services (NS) and vNSF-related information, whose data is used by the vNSF Orchestrator to deploy them into its managed infrastructure. Once deployed, vNSFs and NSs are managed by the vNSFO and verified by the Trust Monitor during the start and at runtime, along with other nodes from the infrastructure, assessing their trustworthiness at all times. These core components, as part of WP3, are complemented by those in WP4: i) the DARE, storing and analysing the security logs and events provided by the running NSs and vNSFs; and ii) the Security Dashboard, presenting the results from DARE to the operator. Both DARE and Security Dashboard components are detailed in deliverable D4.1 but for the sake of providing a self-explained deliverable a brief summary of their function will be presented here.

Monitoring vNSFs inspect captured data and provide valuable information to the DARE component. The network status is reported periodically and all this information is centralised in the DARE. Then, the data analytics framework (DARE subcomponent) analyses all the heterogeneous network information previously collected via monitoring vNSFs and Trust monitor. It features cognitive and analytical components capable of predicting specific vulnerabilities and attacks. Finally, the remediation engine (another DARE subcomponent) provides recommendations in the form of new network services (sets of vNSFs) or medium level policies (configurations of existing VNSFs) to remediate the detected threats. These recommendations and the attack information is given to the intuitive and appealing graphical user interface provided by the Security Dashboard component, which allows authenticated and authorized users to access SHIELD's functionalities. From this dashboard, operators have access to monitoring information showing an overview of the security status. Furthermore, Security Dashboard allows operators as well as tenants to visualized recommendation generate by DARE allowing to take actions and resolve detected vulnerabilities. Authorized users will therefore be able to react through the Security Dashboard, NFVO by deploying new services (NS, VNSFs) if required, or configuring the existing services (NS, VNSFs) to mitigate the attack.

2. Design and Architecture

Network function virtualisation (NFV) technology is one of the cornerstone technologies used within the SHIELD project and has ETSI as one of its main standardisation drivers. The ETSI NFV architecture is used as the starting point for SHIELD's architecture, aiming to place SHIELD in a position where it can contribute with these standardisation activities and align itself to the de-facto architecture. Thus, the software components envisioned in SHIELD's vNSF environment have been aligned wherever possible with the current vision of ETSI community. This vision/architecture may be extended as needed in order to accommodate components or features not yet considered or agreed by this standardisation body. The following figure (Figure 5) displays how SHIELD's architecture aligns with ETSI NFV architecture.

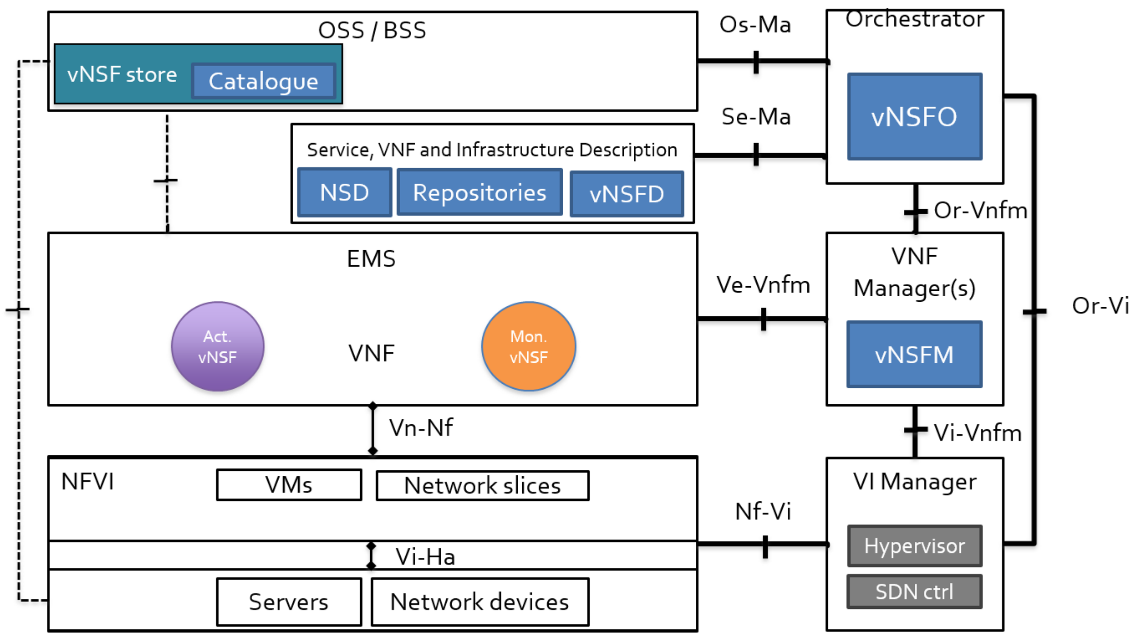


Figure 5: SHIELD vNSF environment's architecture mapped to ETSI NFV architecture

Summarising, the Store lies in the Operational and Business Support layer, the vNSFO directly fits into the role of the Orchestrator envisioned in the ETSI NFV architecture and the vNSFs also have a direct mapping within the VNF section. The subcomponents and even modules or elements were successfully mapped as well, e.g. the NS and vNSF information (descriptors, records, infrastructure-related data, etc.), as well as the vNSF Manager (vNSFM) that directly corresponds to the VNF Manager following what ETSI envisions for the mechanism used to control the vNSFs (EMS subcomponent and so on). The only remaining component present in SHIELD's architecture and in the scope of WP3 (Trust Monitor) performs attestation tasks, which are not contemplated in the ETSI NFV architecture and thus it has no direct mapping.

The following section describes the design and architecture for the SHIELD's WP3 components, i.e. the list of vNSFs to be deployed in the network, the vNSF Store, the vNSF Orchestrator and the Trust Monitor. This description is more detailed than its counterpart in D2.1, as it specifically addresses low-level details such as the subcomponents within the vNSF environment, their detailed workflows and relation between these and other components in the SHIELD platform.

2.1. Security network functions and services

The NFV concept achieves, through virtualisation, the reduction of the capital expenditures incurred by common specialised hardware devices and provides a broad spectrum of network functionalities that are deployed on top of common hardware. The Virtual Network Functions (VNFs) can be moved, restarted or erased rapidly, up to the order of seconds. VNFs implement common network functions such as gateways, proxies, firewalls and transcoders, traditionally carried out by specialised hardware devices and deployed on top of commodity IT infrastructure. The focus within SHIELD will be the development of VNFs implementing security functions (hereinafter called vNSFs). To ease their management, the developed vNSFs will conform to the ETSI NFV group recommendations. In the following subsections we describe a general architecture to be followed by the SHIELD vNFSs.

2.1.1. General vNSF architecture

Each vNSF is composed by one or more VNF Components (VNFCs) that are interconnected through Virtual Network Links (VLs). Security services offered in SHIELD will consist of one or more vNSFs. These services will be dynamically deployed to identify and mitigate security attacks, threatening conditions or anomalous behaviours. The vNSFO will be responsible for the orchestration of the vNSFs into services and the deployment, management and configuration of the resulting end-to-end network services. An example of a network service (NS) that consists of three different vNSFs (VNF1, VNF2 and VNF3) connected through virtual links is shown in Figure 6. As depicted, VNF2 is composed by three VNFCs connected through virtual links that are internal to the VNF.

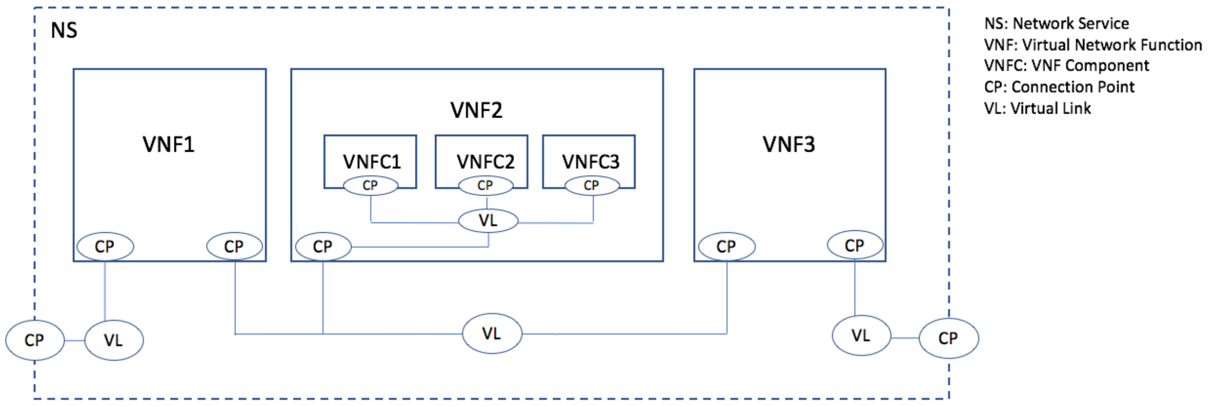


Figure 6: Network Service example

2.1.1.1. vNSF interfaces

According to ETSI NFV specifications [1], there are five types of interfaces identified relevant to a VNF. As illustrated in Figure 7:

- SWA-1 interface: This interface enables communication between various network functions within the same or different network services. The SWA-1 interface can be established between two VNFs, a VNF and a Physical Network Function (PNF), or between a VNF and an End Point. A VNF may support more than one SWA-1 interface.
- SWA-2 interface: This interface refers to VNF internal interfaces, for the communication between the different VNFCs of a VNF, i.e. for VNFC to VNFC communication. The type of information exchanged through this interface depends on the function of the VNF.
- SWA-3 interface: This interface interconnects the VNF with the NFV management and orchestration layer specifically with the VNF Manager (VNFM). Through this interface the lifecycle management of the VNF is performed (e.g. instantiation, termination, scaling, etc.). The SWA-3 interface corresponds to the Ve-Vnfm reference point.
- SWA-4 interface: This interface is used by the Elemental Management (EM) to communicate with a VNF. It is a management interface used for the runtime management of the VNF to perform functions related to Fulfilment, Assurance, and Billing (FAB) as well as Fault, Configuration, Accounting, Performance and Security (FCAPS). This interface will cover also the NSF-facing interface's functionality defined in the IETF I2NSF standard, within the task for defining policy recommendations.
- SWA-5 interface: The SWA-5 interface links the VNF with the NFVI and corresponds to the Vn-Nf reference point. This interface provides access to a virtualised slice of the NFVI resources allocated to the VNF, i.e. to all the virtual compute, storage and network resources allocated to the VNF depending on the VNF type and its special requirements for resources.

As the SHIELD framework is compliant to the ETSI MANO specifications, the SHIELD vNSFs will support the aforementioned interfaces.

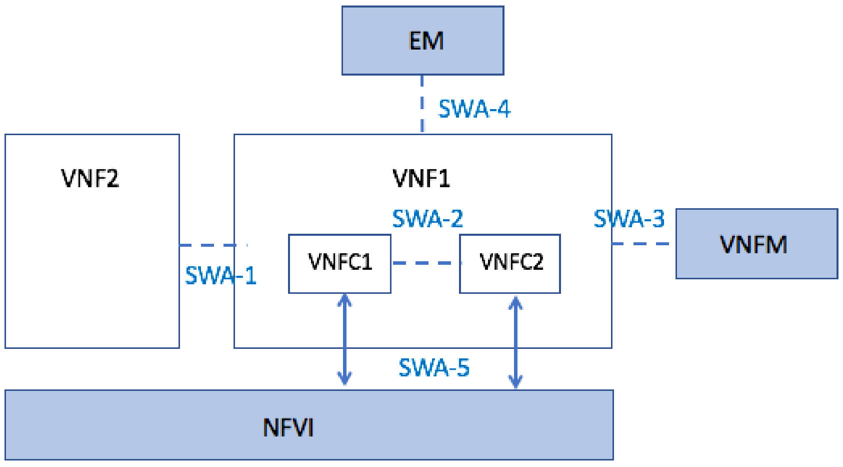


Figure 7: Types of VNF interfaces

2.1.1.2. vNSF common elements

The internal structure of a SHIELD vNSF is illustrated in Figure 8. Although the internal implementation of a vNSF concerning its functionality (vNSF functionality) is to be decided by each vNSF developer, there are some common elements that vNSFs should have to be compatible with the SHIELD framework. Specifically, these elements are:

- The **vNSF controller** is the internal element devoted to the support of the vNSF lifecycle through the vNSFM. The interaction between the vNSFM and the vNSF takes place through the SWA-3 interface.
- The **init configuration** element is responsible for the initialisation of the vNSF that happens at the beginning of the vNSF execution. This is an optional component that is present in the vNSFs for which an initial configuration should take place on the vNSF before its execution.
- The **data collector** element is the component responsible for gathering the output data from the vNSF. The format of the output data are in low-level application-dependent format.
- The **data transformation** element, whose role is to transform the output data of the vNSF from a low level, application-dependent format (Data Collector) to a high-level format that is understandable by DARE.
- The **configuration listener**, an element responsible to listen for new policy configurations recommended by the Remediation Engine of DARE and injected by the vNSFO into the vNSF.
- The **policy transformation** element, whose role is to transform the high-level format rules recommended by the Remediation Engine of DARE (policies) to low-level, application-dependent format rules that can be enforced to the vNSFs. This element will be part of each vNSF on which policy enforcement is expected to take place.
- The **streaming service** is the element responsible for transmitting application-level monitoring data, such as security logs or alerts produced by the vNSF, to the Streaming Service located at the DARE.
- The **vNSF Functionality** element represents the functionality performed by the vNSF.

It is important to note that the above elements are not what we refer to as vNSFCs (or vNFCs in ETSI terminology). It is possible that all the above elements reside in a single vNSFC. Additionally, apart from the vNSF Controller that allows the lifecycle management of each vNSF, none of the other described elements are mandatory for all vNSFs. The presence of the other components listed above is dependent of the type of each developed vNSF. The data collector, the data transformation and the streaming service will be present in all vNSFs that produce some output, which will be used by the Data Analytics Engine of DARE for the identification of security incidents and threats. Similarly, the policy transformation and the configuration listener will be present in all vNSFs that permit some application-level configuration for security purposes (threat identification or mitigation) through the Security Orchestrator.

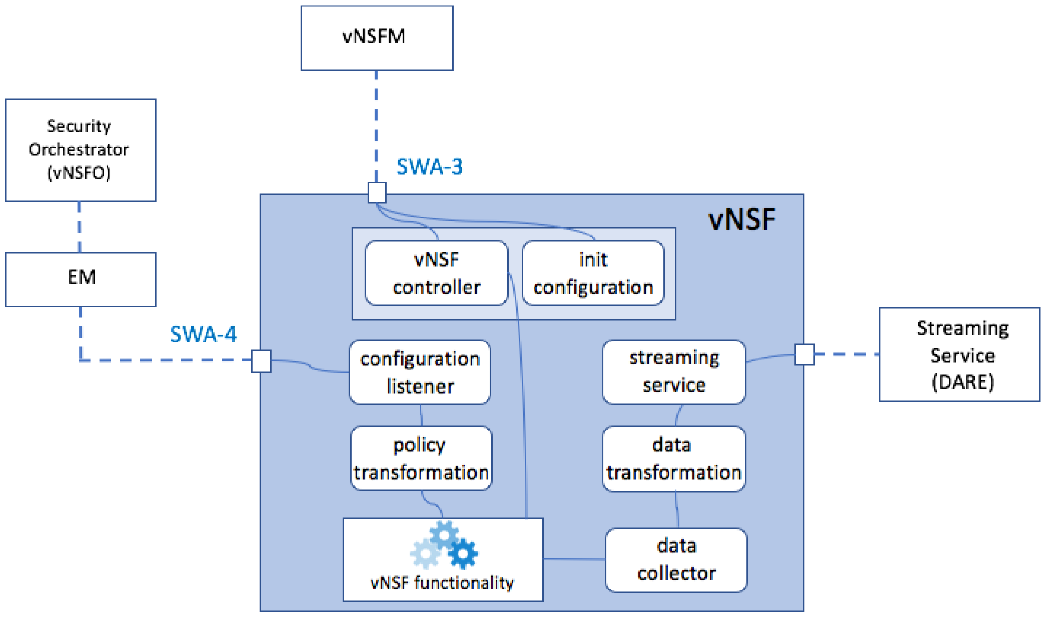


Figure 8: Internal structure of a SHIELD vNSF

Figure 9 depicts the internal elements of a vNSF comprised of two vNSFCs. In the case of having a vNSF comprised of more than one vNSFCs, the vNSF Controller element will be present in one of the available vNSFCs. Additionally, the vNSF Functionality element will be present in all vNSFCs composing the vNSF. The remaining elements of the common vNSF architecture can be freely allocated in the different vNSFCs, again taking into account the type and function of the vNSF (vNSF that provides output, vNSF that accepts configuration, etc.). In the specific example illustrated in Figure 9, the data collector, the LH data transformation and the streaming service components reside in the second vNSFC (vNSF-C2).

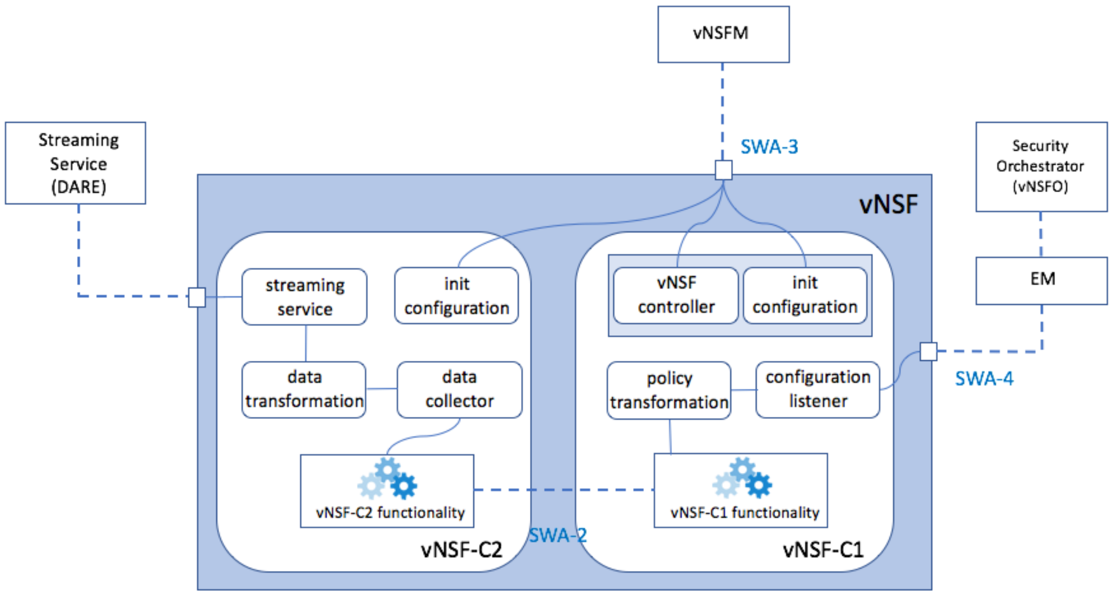


Figure 9: Internal elements of a vNSF composed by two vNSFCs

2.1.1.3. vNSF Descriptor (vNSFD) and NS Descriptor (NSD)

Each vNSF will have an associated descriptor document, whose role is to instruct the vNSFO on how to deploy it and how it should be connected to other virtual functions. This descriptor document, usually referred as vNSF Descriptor (vNSFD) is a deployment template which describes a vNSF in terms of deployment and operational behaviour requirements. Information typically detailed in the vNSFD contains deployment rules, scaling policies and monitoring parameters related to the function of the vNSF. Moreover, the vNSFD will contain connectivity, interface and KPIs requirements that can be used by the vNSFO to establish appropriate virtual links between vNSF components instances, or between a vNSF instance and the endpoint interface to other virtual functions. A similar descriptor file is accossiated with each network service (NSD), providing information on the vNSFs that comprise a particular network service, as well as connectivity information that specifies how these vNSFs are chained together to provide a given network service.

2.1.2. SHIELD vNSFs

SHIELD will implement several monitoring and remediation vNSFs. Monitoring security functions perform traffic monitoring and analysis to detect intrusions and report illegitimate traffic or malicious activity. On the contrary, the role of the remediation security functions is to mitigate security threats or risks by applying security policies and taking actions, such as dropping/rejecting specific packets or flows and blocking data coming from specific users. It must be noted that several vNSFs implemented in the project may assume both roles, i.e. monitoring and remediation.

2.1.2.1. Monitoring vNSFs

The monitoring vNSFs will probe the network in different ways to extract relevant low-level information from the NFVI. This network data is called “Network data collection” and its contents will vary depending on the purpose of each monitoring vNSF. After the network data collection is obtained, it is transformed from an application specific format into a high-level structure with a generic format via the “data transformation” process and then is sent to the DARE through the “Streaming Service” interface (as depicted in Figure 8). The rationale of converting the data to a generic format and provide the DARE’s Streaming Service with a generic format is to allow DARE’s compatibility with different implementations for a sigle vNSF type. For example, the definition of a generic format for monitoring data coming from intrusion detection systems would allow the compatibility with different IDS vNSFs implementations (e.g. Snort, Bro, Suricata).

2.1.2.2. Remediation vNSFs

The reacting vNSFs will be in charge of providing mitigation actions, as defined by the DARE. The rules or policies composing a mitigation action and expressed via an application-independent configuration's abstraction, will be proxied to the vNSFO by the Security Dashboard, when accepted by the final user. Each reacting vNSF involved in a particular mitigation action will receive the set of policies via the SWA-4 interface (Figure 8) and will be in charge of translating it to low level configuration, understandable by the implemented security network function. Thus, the translation process will be offloaded to the different reacting vNSFs in a specific module, named policy transformation in Figure 8. This is done in order to reduce the load on the centralised points of the architecture, as well as to ease any modification/update in the translation process by the vNSF developer.

2.1.2.3. List of vNSFs

The consortium has selected a number of candidate vNSFs that will allow to demonstrate SHIELD capabilities (detection and mitigation) in security attacks. Specifically, the following vNSFs are targeted for implementation: an Intrusion Detection System (IDS), a mcTLS gateway, a traffic analysis vNSF,a deep packet inspection vNSF, a packet filter vNSF acting at network layer, a forward proxy vNSF acting at application layer. The specific functionalities selected for the vNSFs implementation depends on the security requirement analysis as defined in WP2 and on the security threats to be addressed during the project's demonstrations.

The detail associated with the specification of each vNSF differs based on its maturity level. As a consequence, some vNSFs already provide a detailed low-level specification of its internal architecture/workflow while others are still in a preliminary stage, therefore presenting only its envisioned functionalities.

Virtual Intrusion Detection System (vIDS)

An IDS is equipped with advanced traffic analysis and monitoring capabilities for attack and vulnerability detection. It monitors and logs the network traffic for signs of malicious activity and generates an alert upon discovery of a suspicious event. Two different techniques are used to detect malicious traffic/activity, separating IDSs into two main categories: i) statistical anomaly-based IDS and ii) signature-based IDS. Anomaly detection IDSs have the advantage over signature based IDSs in detecting novel attacks for which signatures do not exist. However, anomaly detection IDS suffer from high false detection rate.

IDS deployment typically consists of one or more sensors placed strategically on the network. Additionally, the solution may contain an optional central console for easier management of all sensor nodes.

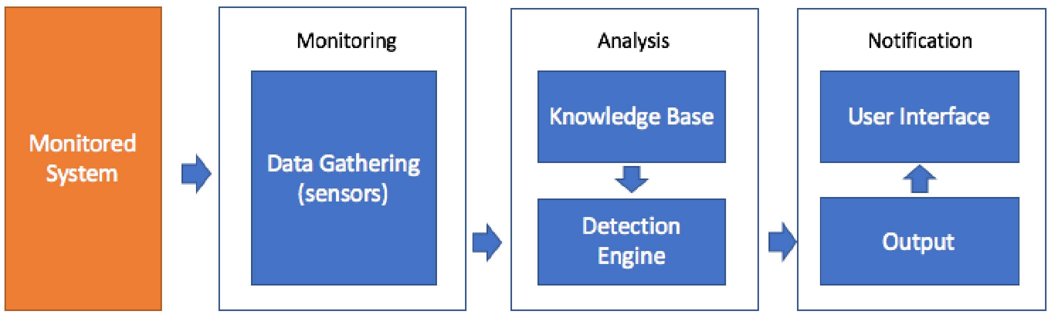


Figure 10: Typical IDS architecture

The functionality of an IDS involves three distinct phases: a) Monitoring, b) Analysis and c) Notification. A typical architecture of an IDS is illustrated in Figure 10. During the Monitoring phase, the IDS is collecting data from the monitored system, through the deployed sensors. At the Analysis phase, the IDS Detection Engine analyses the gathered data by using a Knowledge Base. The Knowledge Base includes information that allows the Detection Engine to classify the analysed data as threatening events. This information includes predefined rules (signatures), user defined rules or historical data. The historical data allows the modelling of the normal behaviour of the monitored system into a profile enabling the detection of deviations of the current status when compared to this considered normal profile. Finally, during the notification phase, the IDS will output notifications of the detected events by logging this information into specific files and user interfaces or trigger alerts that can be consumed by other components.

Deep Packet Inspection (DPI)

DPI is the practice of filtering and examining IP packets, across Layers 2 through 7. Although Stateful Packet Inspection (SPI, often employed by firewalls) is more restricted, DPI may extend to headers, protocol structure and payloads, thus allowing for the implementation of advanced cybersecurity measures. DPI can be an effective detection tool for a multitude of cyberattacks such as Denial of Service (DoS), buffer overflow, cross-site scripting exploits, injection attacks etc. DPI capabilities, however, can be limited as the payload structure becomes more complex (e.g. through obfuscation, encryption etc). SHIELD aims to implement a vNSF dedicated to DPI, as part of the trusted platform.

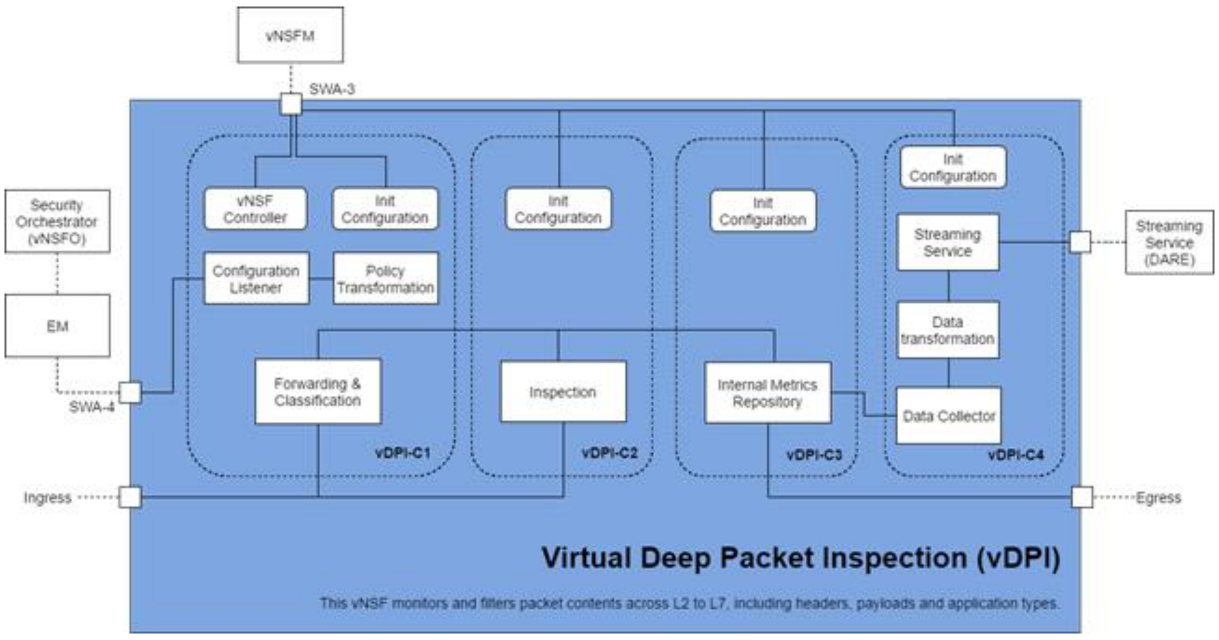


Figure 11: vDPI design and main components

SHIELD will implement a trusted vDPI encompassing several vNSF components (vNSFCs) as illustrated in Figure 11:

- **vDPI-C1 (Forwarding and Classification):** This vNFC handles routing and packet forwarding. It accepts incoming network traffic and consults the flow table for classification information for each incoming flow. Traffic is forwarded using default policies until it is properly classified and alternate policies are enforced. It is often unnecessary to mirror packet flow in its entirety in order to achieve proper identification. Since a smaller number of packets may be utilized, the expected response delay can therefore be close to negligible. In a case where the Inspection, Forwarding and Classification VNFCs are not deployed on the same compute node, traffic mirroring may introduce additional overhead. A classified packet can be redirected, marked/tagged, blocked, rate limited, and reported to a reporting agent or monitoring/logging system within the network.

- **vDPI-C2 (Inspection):** The traffic inspection vNFC implements the filtering and packet matching algorithms and is the necessary basis to support additional forwarding and classification capabilities. It is a key component for the successful implementation of the vDPI and the most computationally intensive. The component includes a flow table and an inspection engine. The flow table utilises hashing algorithms for fast indexing of flows, while the inspection engine serves as the basis for traffic classification.
- **vDPI-C3 (Internal Metrics Repository) & vDPI-C4 (Monitoring Dashboard):** The internal metrics repository acts as local storage, while the Monitoring Dashboard handles data sharing with DARE.

The vDPI lifecycle is managed by the vNSF Orchestrator, and specifically the vNSF Manager Engine. The vNSFO is in charge of starting, stopping, pausing, scaling and configuring the vDPI. Thus, the Forwarding and Classification component acts as a managing/controlling vNFC and is assigned a floating IP for management. Internal communication is implemented via vlinks (detailed in section “Specifications and Implementation”). Policies are relayed from the Security Orchestrator and translated within the managing vNFC.

mcTLS Middlebox and Gateway

Multi-Context TLS (mcTLS) [8] is a secure protocol that extends TLS to incorporate trusted middleboxes into a secure session. In the mcTLS negotiation, server and client decide the TLS policy (for example: allow trusted middleboxes to access the headers). Middleboxes have access only to information required for accomplishing their function (negotiated during the mcTLS policy definition).

The objective of this vNSF is to provide an environment able to monitor only the necessary payload of HTTPS traffic in order to identify security issues. The mcTLS Gateway can be used during the process of adoption of the mcTLS protocol and until it is widely adopted. The Gateway is deployed close to the legacy TLS Web server, which is subject to monitoring.

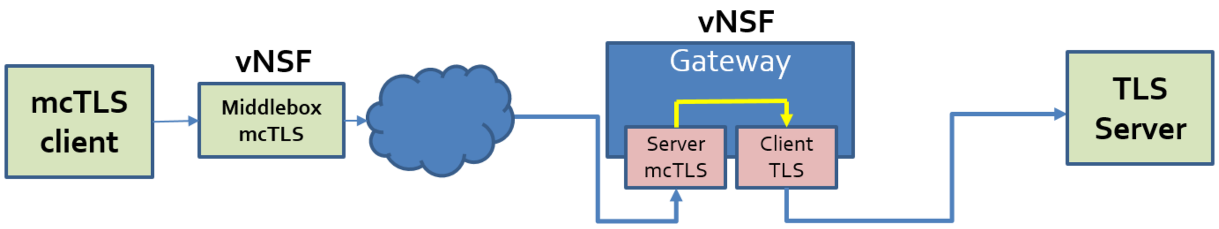


Figure 12: mcTLS Gateway elements

As shown in Figure 12, mcTLS will be deployed using two different elements:

- **Gateway**: Provides a mcTLS interface to the client (requires the use of a client with mcTLS) without modifying the server.
- **Middlebox**: Performs the monitoring of the payload negotiated with the client and the Gateway.

Functionalities provided:

- Gateway to translate between the mcTLS and TLS protocols providing a conversion tool for companies wishing to monitor their Web services in HTTPS traffic (controlling which parts of the data that can be read or written by trusted middleboxes) for security reasons or for CDN providers in order to monitor their clients TLS traffic.
- Middlebox for traffic monitoring: authorised data (e.g. selected HTTP headers, specific Content-Types…) over mcTLS traffic.
- Potential direct interaction with other vNSFs able to block traffic when a security threat is detected through SHIELD framework.
- Certificate delegation and control from origin servers through ACME (Automated Certificate Management Environment) [9].

HTTP/S Analyser

The objective of this vNSF is to provide the classification of HTTP and HTTPS traffic without analysing the payload content in a privacy-friendly way.

This vNSF will be trained through machine learning techniques to provide the HTTP traffic classification in order to be able to analyse the behaviour of a device or network. The vNSF will be able to work with the traffic mirror or with stored information in tstat [10] format.

Functionalities provided:

- Traffic capture and tstat format traffic generation.
- HTTP/S traffic and classification in several categories: BROWSING, VIDEO, DOWNLOAD by network flow. Traffic analysis is based only in L2-L4 (it is not a DPI vNSF).

L3 Filter

This vNSF will implement a filtering application acting at the network layer, or Layer 3 of the ISO/OSI stack. It will allow or deny traffic by specifying an Access Control List (ACL), in form of a whitelist or blacklist. The ACL will be configured by translating the high level configuration to a set of filtering rules for specific IP addresses.

Functionality provided:

- Allow or deny traffic identified by a certain IP address (source, destination).

Forward L7 Filter

This vNSF will implement a forward proxy that would offer the possibility to block all the traffic the user wants to block. To do so, it will inspect traffic at application layer (also named Layer 7 in the ISO/OSI stack) and filter it according to defined rules. The vNSF will behave as an agent that will receive requests from a client (e.g. a web browser) and forward them to the specified server, if it doesn't match a blacklist.

Functionalities provided:

- Traffic inspection for specific Layer 7 protocols and headers(e.g. HTTP, FTP);

- URL filtering;
- Access Control List (e.g. IP based, MAC based, domain based).

2.1.2.4. Functionality mapping

The following table describes, per vNSF, how these provide the specific monitoring or remediation capabilities.

vNSF	Monitoring (description)	Remediation (description)
vIDS	Real-time traffic analysis (L3-L4 and L7) for intrusion detection based on signatures. It can also be used as a simple packet sniffer or packet logger.	No
vDPI	Filtering and examining traffic (L2-L7), extending acquisition of headers, protocol structure, application types. Payload analysis might be available on a per-case basis.	Mirroring suspicious flow to DARE and limiting or blocking it
mcTLS Gateway	Monitoring the necessary payload of HTTP requests to identify threats.	No
Traffic analysis for HTTP/HTTPS	Classification of HTTP and HTTPS traffic using ML techniques, without analysing the payload content.	No
L3 Filter	No	Allow or deny traffic identified by a certain IP address (source, destination)
Forward L7 Filter	No	Traffic inspection for specific Layer 7 protocols and headers (e.g. HTTP, FTP), URL filtering, Access Control List (e.g. IP based, MAC based, domain based)

2.2. Store

SHIELD aims to set up a single, centralised digital store for vNSFs and NSs. This approach allows SPs to offer new security features for protecting the network or extend already existing functionalities without the need of modifying core elements of the framework. The store acts as a repository for vNSFs and NSs that have been previously published.

The main novelty in the Store is the onboarding of vNSFs/NSs in a secure and trusted way. The onboarding process will ensure the provenance is from a trusted source and that the contents integrity can be assured. Once this is achieved the security information is stored for safekeeping and provided upon request so other components can check that the vNSF/NS has not been tampered with since it was onboarded.

Another relevant feature provided by the Store is the verification done on the vNSF and NS associated descriptors to ensure the instantiation process by an Orchestrator is performed without hassle. Building on the descriptors syntax check concept from the SONATA project [2], the submission process shall check all descriptors for inconsistencies as well as implement a network topology validation. This last check will prevent issues such as unwanted loops in the forwarding graphs or reference to undefined networks or missing ports.

Figure 13 presents all the Store sub-components, along with their relations depicted.

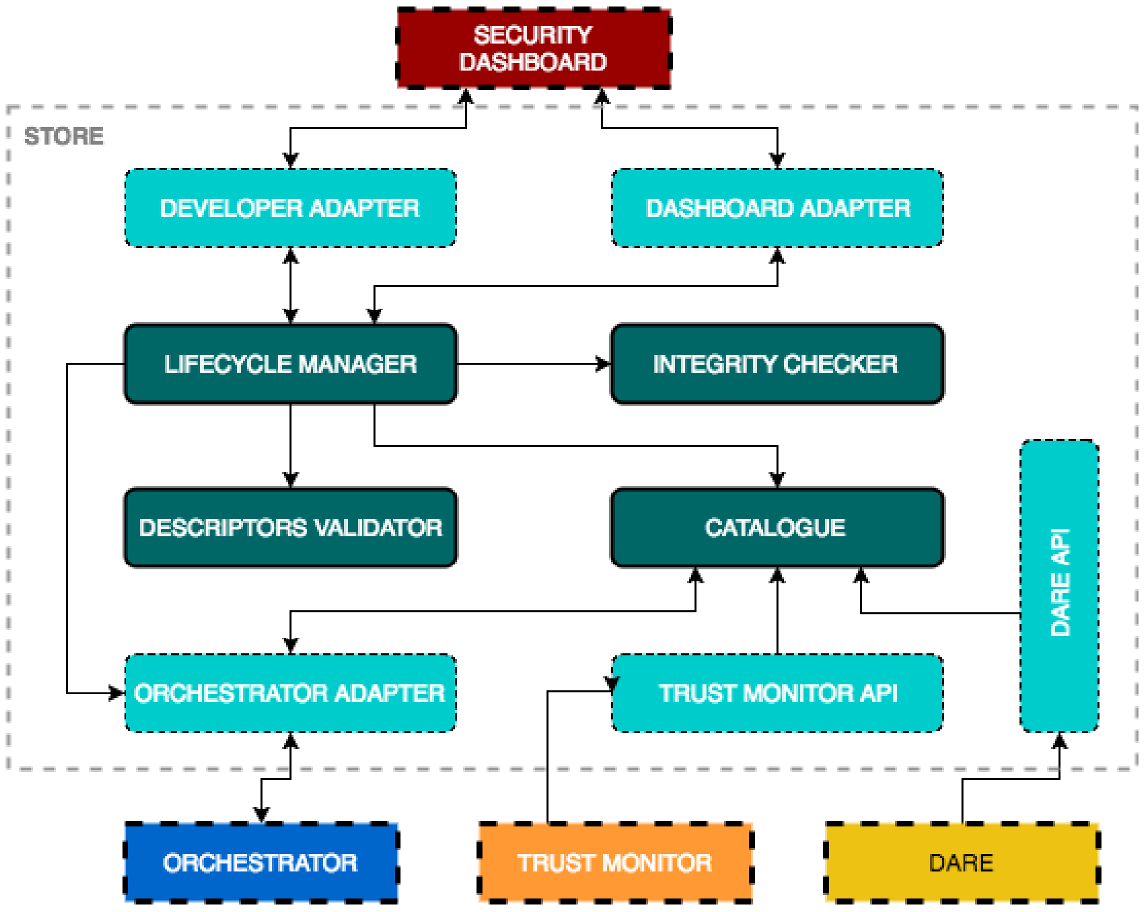


Figure 13 : vNSF Store subcomponents

2.2.1. Subcomponents

The current section will present each subcomponent depicted in Figure 13 mentioning its main role. STORE component encloses four main subcomponents (LIFECYCLE MANAGER, INTEGRITY CHECKER, DESCRIPTOR VALIDATOR and CATALOGUE) as well as four subcomponents aiming to provide connectivity with other SHIELD components. These subcomponents (DEVELOPER ADAPTER, DASHBOARD ADAPTER, ORCHESTRATOR ADAPTER, TRUST MONITOR API and DARE API) will be translated to either: APIs (providing a connection point to external components);

Connectors (using the features of external components); Adapters (enclosing both API and Connectors features.

Lifecycle Manager

This subcomponent manages the vNSF/NS onboarding lifecycle. From the moment a NS/vNSF is submitted to the Store this sub-component takes over the entire process and ensures the proper steps are performed for a successful onboarding. In the event of a failure it notifies the Developer of the situation and performs all the necessary housekeeping steps.

Descriptors Validator

Successful vNSF/NS onboarding comprises parsing its descriptor and validate the specified deployment and operational behaviour requirements. This job is performed by the Descriptors Parser sub-component.

The two main tasks assigned to this component are syntax validation to prevent incorrect vNSF/NS descriptors from being processed for instantiation, and topology validation to assure the integrity of the vNSF/NS topology and avoid inconsistencies such as potential loops in the forwarding graphs or referenced to an undefined network or missing ports.

Integrity Checker

When submitting a vNSF/NS to the Store the Developer must provide a manifest of the files used (or referenced) by the vNSF/NS. This manifest must contain hashes of each referenced file, and must be digitally-signed so its contents can be trusted.

It is paramount to a secure environment to ensure that the vNSF/NS content is trusted and wasn't tampered with in any way once onboarded. The goal of the Integrity Checker sub-component is to verify the integrity and provenance of the submitted data. This process encompasses validating the manifest which holds the hashes for the all files, as well as the ones regarding the descriptors. This information is provided later on to any component assessing that the vNSF/NS wasn't tampered with.

Catalogue

All the onboarded and sandboxed vNSFs/NS are kept in a repository. The Catalogue sub-component manages the records-keeping activities. Any additional metadata associated with the onboarding process or the vNSF/NS itself is managed here as well.

Developer Adapter

Ensures integration features with the developer.

Dashboard Adapter

Ensures integration features with the Security Dashboard.

Orchestrator Adapter

Ensures integration features with the Orchestrator.

Trust Monitor API

For vNSF security-related queries an API is provided to the Trust Monitor. Any interaction with the Store is done through this interface.

DARE API

For DARE-related queries an API is provided to DARE. Any interaction with DARE is done through this interface.

2.2.2. Update since D2.1

As a result of the specification activities, it was decided to keep a single NS/vNSF catalogue instance for use in the SHIELD platform, placing this catalogue as a subcomponent of the Store. This approach helps to reduce information replication throughout SHIELD’s components and define more clearly the responsibilities of both Store and vNSFO components. The Store will be responsible for managing and providing the information of all the onboarded NSs and vNSFs.

Additionally, after further analysis of the requirements in D2.1, two additional platform requirements were mapped to the Store component: “PF02 - vNSF lifecycle management” was included since the Store is responsible for managing partially the lifecycle of vNSFs, being responsible for the onboarding process; “PF11 - vNSF attestation” is now mapped to the Store, since the Store will validate the digital-signature of each artefact onboarded, thus assessing the validity of its provenance.

2.2.3. General workflow

The Store interacts with multiple components, both in the vNSF environment (vNSFO, Trust Monitor) to provide information of NSs and vNSFs available at the catalogue; and with other components of the SHIELD platform (DARE and Security Dashboard) for analytics and visualisation purposes. Besides this, the Store exposes endpoints to the NS/vNSF developers to onboard new NSs. The data flow diagram of the Store (Figure 14) depicts these interactions.

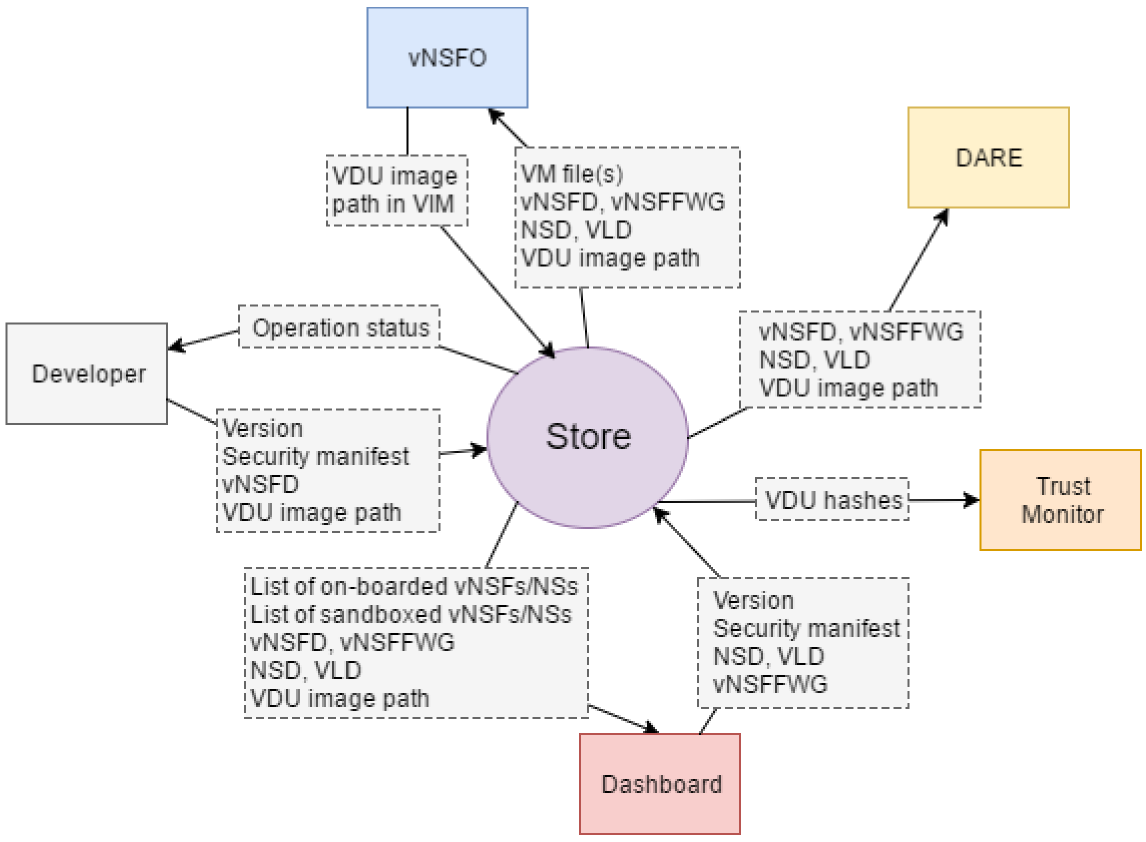

vNSFO
VDU image path in VIM
VM file(s) vNSFD, vNSFFWG NSD, VLD VDU image path
DARE
Developer
Operation status
Store
vNSFD, vNSFFWG NSD, VLD VDU image path
Version Security manifest vNSFD VDU image path
VDU hashes
Trust Monitor
List of on-boarded vNSFs/NSs List of sandboxed vNSFs/NSs vNSFD, vNSFFWG NSD, VLD VDU image path
Version Security manifest NSD, VLD vNSFFWG
Dashboard

Figure 14: Data flow diagram of Store

2.2.4. Internal operation

The subcomponents of the Store work together to perform operations related to the onboarding process, such as the validation of the vNSF and the registration of the VDU image(s); as well as the decommissioning of NSs and vNSFs. The specific workflows for such operations are described in the "Appendix A Intra-component interactions".

2.2.5. Interactions with other components

The Store interacts with other components, namely the vNSFO, the Trust Monitor, the DARE and the Security Dashboard; as well as with the end users. Specific details are provided in the "Appendix B Inter-component interactions".

2.3. Orchestrator

The vNSF orchestrator (vNSFO) used in SHIELD is an implementation following the NFV MANO (Network Functions Virtualisation Management and Orchestration) WG specifications. This orchestration component deploys the vNSFs and the NSs (made up of vNSFs) and manages their lifecycle; while also performing the global resource management, monitoring, validation and the authorisation of the NFVI resource requests. The specific functionality is delegated to specific subcomponents (Figure 15).

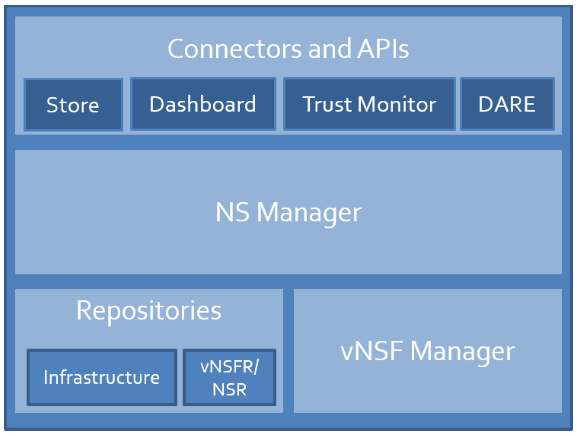


Figure 15: vNSFO subcomponents

2.3.1. Subcomponents

The functionality of the vNSFO is distributed among the following subcomponents: NS Manager, vNSF Manager, Repositories and APIs. These are explained below.

NS Manager

This manager controls the lifecycle of any given NS. It can instantiate (deploy) or terminate (destroy) a given service into/from the NFVI. Modifying the configuration of the constituent vNSFs is supported as well. Finally, monitoring and scaling capabilities are permitted to inspect and adapt the service to varying conditions of the network capabilities, namely those produced under attack.

vNSF Manager

This subcomponent manages the lifecycle of one or more vNSFs. Therefore, it is able to interact with the vNSFs in order to deploy (instantiate) or terminate these interacting with the NFVI to ensure these features. Configuration (or modification of their configuration) is supported as well, typically done at boot but also allowed for passing policies to be translated within the vNSFs. Other operations such as monitoring and scaling features are also envisioned to be provided by this component.

Repositories

Different repositories and registries offer helper functionality to the vNSF and NS Managers, acting as records for run-time information on the operation of the deployed NSs/vNSFs or on the status of the resources managed by the VIM. Such data is required for multiple operations; from attestation to analytics or mitigation, and also for visualisation purposes.

Connectors and APIs

The vNSFO exposes data through APIs and implements connectors to consume other components' APIs:

- **Store Connector:** The vNSFO queries the vNSFO Connector from the Store to retrieve all data related to a given NS or vNSF, given its ID in the Store. This step is required to initiate the deployment of a specific NS.
- **Dashboard Connector:** A user may select a recommendation from the Security Dashboard in order to deploy a specific NS in the network infrastructure and mitigate a specific threat. The Security Dashboard initiates then the communication with the vNSFO by pushing specific instances to deploy on specific locations and the Medium-level Security Policy Language (MSPL) policies that allow the configuration the vNSFs.
- **Dashboard API:** The vNSFO provides the Security Dashboard with data on the NFVI and running instances for visualisation purposes; namely the network topology and the running instances deployed per tenant.
- **Trust Monitor Connector**: The vNSFO contacts the Trust Monitor in order to perform the attestation on any given virtual or physical nodes. In case the attestation fails, the node shall be excluded from the NFVI.
- **Trust Monitor API**: The vNSFO provides the Trust Monitor with information on the network, the flow tables and the list of active physical nodes and running virtual instances.
- **DARE API:** The vNSFO provides the DARE component with the topology of the network, the list of instances per tenant and the active deployed instances. This provides DARE with enough information on the network view to adequately provide mitigation recommendation.
- Other connectors will certainly be used in order to perform orchestration-related operations; for instance the connector to the VIM will allow provisioning resources in the NFVI and retrieving metrics.

2.3.2. Update since D2.1

The subcomponents of the vNSFO have been further analysed against the requirements, and consequently adapted. This analysis resulted in some subcomponents being renamed to better state their functionality (the "Orchestrator Engine" has been renamed to "NS Manager", the "Infrastructure Repository" was renamed to "Repositories", "Northbound" and "Data engine" APIs were renamed to "Store", "Dashboard", "DARE" Connectors and APIs). Some were added to cover extra functionalities (the connection with the Trust Monitor) and others were removed to define the architecture more clearly and avoid duplicities (the "Catalogue", introduced as a separate subcomponent, is kept in the Store; and the "Monitoring" is a feature provided by the NS and vNSF Managers).

The mapping of the functionality of the vNSFO with the requirements from SHIELD is revised according to the specifications proposed. "PF10 - vNSF validation" is now a responsibility of the Store. On the other hand, other Platform Functional (PF) requirements and Non-Functional (NF) requirements have been now mapped to vNSFO to cover related, though not direct, responsibilities.

2.3.3. General workflow

The vNSFO communicates with other components in the vNSF environment to retrieve vNSF and NS-related information (Store), support the attestation of the security state of the running vNSFs, receive notifications (Trust Monitor) and inject policies (vNSFs). The orchestrator communicates as well with other components in the SHIELD platform in order to receive policies for the vNSFs (Security Dashboard) and to provide up-to-date status on the network and vNSF status (DARE). The data flow diagram of the vNSFO (Figure 16) depicts these interactions.

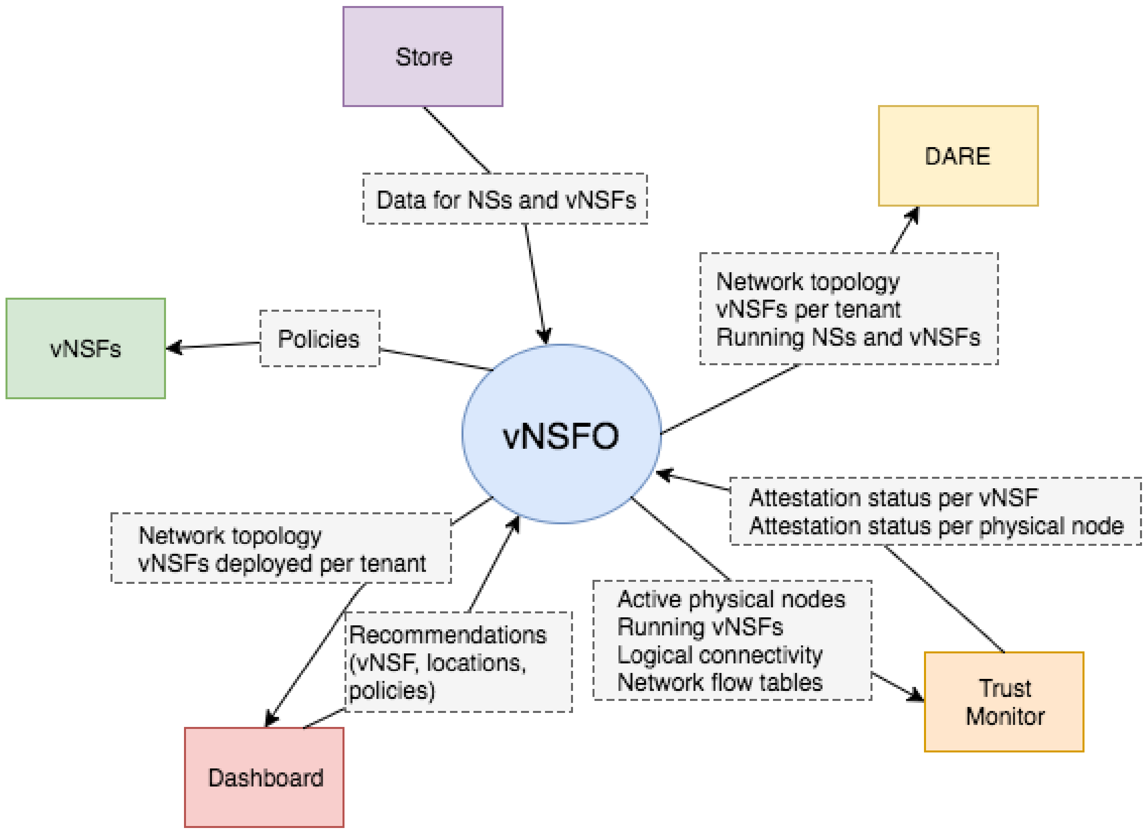


Figure 16: Data flow diagram of vNSFO

2.3.4. Internal operation

Both the NS Manager and vNSF Manager work closely, along with other subcomponents of the vNSFO, to manage the lifecycle of the NSs and vNSFs. The operations are described in the “Appendix A Intra-component interactions”.

2.3.5. Interactions with other components

The vNSFO interacts with the Store, NFVI, Trust Monitor, DARE and Security Dashboard in order to obtain information on NSs, deploy their resources, attest them and gather information to support analytics and visualisation. Specific details are provided in the “Appendix B Inter-component interactions”. Trust Monitor

2.4. Trust Monitor

The Trust Monitor (TM) assesses the trust in the network infrastructure bearing the deployed vNSFs, namely the NFVI Points of Presence (PoP) and the hardware network devices (e.g. switches). 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 detecting intrusion in the network infrastructure, considering the control and management plane components (vNSF store, orchestrator, DARE, Security Dashboard) implicitly trusted. From a technical standpoint, extending the TM security concepts to assess the control and management plane, is feasible since they are based on the same kind of computer architecture (in term of operating system, virtualisation technology, application packaging).

SHIELD's 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 plan;
  - T4 - flashing of firmware/software of the network infrastructure nodes;
- Software threats:
  - T5 - zero-day vulnerability exploitation;
  - T6 - malicious (or accidental) administration: configuration modification, crafting SDN rules update;
  - T7 - installation and execution of arbitrary firmware/software;

SHIELD 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 SHIELD does not provide any physical perimeter protection.

Using Trusted Platform Module (TPM), remote attestation and other Trusted Computing mechanisms, the TM protects SHIELD's network infrastructure against T3, T4, T6 and T7. Particularly, the TPM protected log of all binary executed on a node allows the TM 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 TM does not detect it directly but instead would detect any behaviour's modification of the computer or network nodes since it would not be correct compared to the genuine control plane components, mainly the vNSFO. The TM verifies each node against their expected state, as configured by the vNSFO; if an attacker introduces a new control plane entity and change – even slightly - the configuration of one node, the TM will detect it since it will not match the vNSFO's view.

T5 is not detected by the TM or regular Trusted Computing mechanisms. Zero-day vulnerability can be reduced by using code analysis tools and/or prevent their consequences by reducing the ability of the attackers by using mechanisms such as control-flow protection. Nevertheless, these kind of attacks are usually the initial attack vector used to install additional software: execution of this software is detected by the TM.

Each physical node must be successfully authenticated - using hardware-based cryptographic identities - and verified by the Trust Monitor before joining the SHIELD infrastructure. The Trust Monitor leverages the Remote Attestation workflow, as defined by the Trusted Computing (TC) [11] paradigm (see "Appendix C Definition of technologies"), to verify the integrity of the code being executed (e.g., running instances of vNSFs, software directly managing virtualisation processes, etc) on each physical node, as well as its configuration, both at boot and run-time. The TM acts as a continual verification engine for the physical infrastructure hosting the NSs, capable of interacting with the rest of the vNSF ecosystem (vNSFO, vNSF 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, and to report them 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 and valid configurations. The networking-related configuration, including the dynamic Software-Defined Network forwarding rules, are verified by the Trust Monitor as well, using the overall view available in the vNSFO.

Trust Monitor subcomponents are identified in the figure 17.

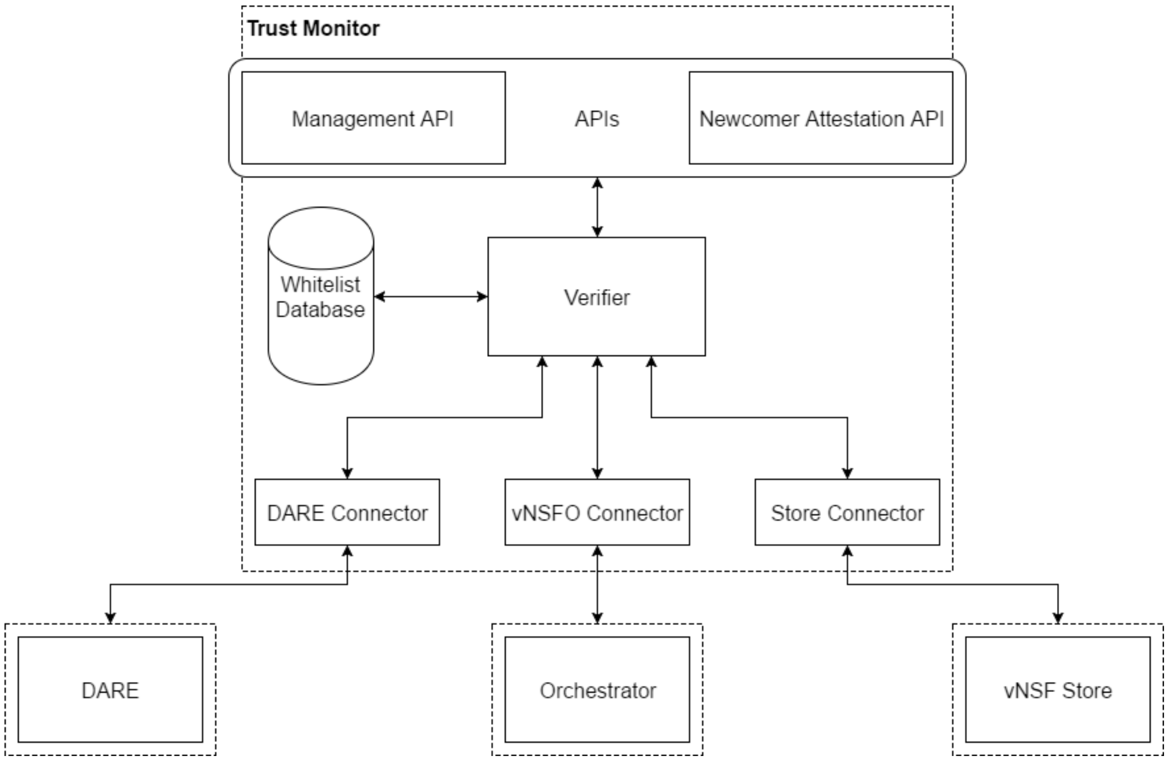


Figure 17: Trust Monitor subcomponents

2.4.1. Subcomponents

A description of the TM's subcomponents depicted in Figure 17 is provided below:

Verifier

It performs the TC-compliant Remote Attestation operations on each component that has been pre-registered with it. It performs both initial attestation of newcomers, periodic attestation tasks and notification of security events to both the DARE and the vNSF Orchestrator. Each target must run specific software to gather the integrity measurements and send back this information to the Verifier.

Whitelist Database

It contains the list of measurements of known software - for both the platform and the vNSFs - and valid configuration. The list of known measurements for each vNSF is gathered from its security manifest in the vNSF Store. It should be noted that vNSFs are versioned in the Store, which allow detection of changes in a vNSF (and hence the need to update the Whitelist Database) or simultaneous use of different versions of the same vNSF.

vNSF Store Connector

This connector is used to receive requests for integrity information from the store for each vNSF to be attested. This subcomponent is responsible for querying the vNSF Store via a client API and for retrieving the data required for the attestation of the vNSF: code in execution, with a special emphasis on custom applications that are not available from the standard software repositories, and configuration files required by the integrated security function, deployment and runtime. This information is required to keep the Whitelist Database up to date with the measurements of the software components needed for the execution of the vNSF.

The TM updates the Whitelist Database only when it detects that a new vNSF, or an updated version of it, is deployed in the NFVI. This could be achieved by keeping a version for each vNSF and check the version of running vNSFs against the already measured ones.

DARE Connector

This connector sends security events to the DARE if one physical or virtual instance is detected as compromised by periodic attestation, or in case a newcomer fails during authentication or initial integrity validation. The subcomponent's workflow is triggered by the Verifier.

vNSFO Connector

The vNSFO connector notifies the vNSFO about the need to terminate a compromised vNSF or to exclude a physical node from the NFVI. This workflow is triggered by the Verifier upon a failed attestation. In addition, it is used to request the configuration of the network at a given time from the vNSF Orchestrator. The configuration consists of the description of active physical nodes, running virtual instances, logical connectivity and network flow tables.

Newcomer Attestation API

It exposes an API that receives requests from the vNSFO for remote attestation of a node of the NFVI. The attested node must be pre-registered with the TM before performing the attestation procedure.

Management API

This is a read-only interface for retrieving status information about the attestation of the infrastructure.

2.4.2. Update since D2.1

After further analysis, the interface between the Trust Monitor and the vNSFO should be extended to support enrolment of a newcoming node on the NVFI PoP in the TM.

In addition, the mapping between the component and the Platform Requirements (PF), as envisioned in D2.1, has been reconsidered with regards to the capabilities of the other components of the platform. More specifically, "PF04 - Security data monitoring and analytics" is addressed by the data acquisition and analysis capabilities provided by the DARE, as the Trust Monitor does not receive logs straight from the vNSFs to detect occurring security incidents. The "PF13 - Mitigation" requirement is addressed by the recommendation and remediation capabilities of the DARE; the corresponding requirement for the Trust Monitor is “PF19 - Network infrastructure attestation”. The "PF18 - Service composition" requirement is addressed by the Security Dashboard and the DARE, as they are the components involved in the selection and deployment of vNSFs. Differently from D2.1, the "PF11 - vNSF attestation" requirement is fulfilled by the Trust Monitor, which is collecting attestation's data from the hosts running the vNSFs and check their integrity information against the known values retrieved from the Store.

2.4.3. General workflow

The purpose of the TM is to assess the trustworthiness of the nodes composing the NFVI, in order to act on compromised nodes (e.g. exclusion from the NFVI) and attest the integrity of newcomers. To do so, the TM should be able to interact and cooperate with several other components of the SHIELD infrastructure, such as the vNSFO, vNSF Store, etc. An overall description of the flows between the TM and the other component of the infrastructure is depicted in Figure 17.

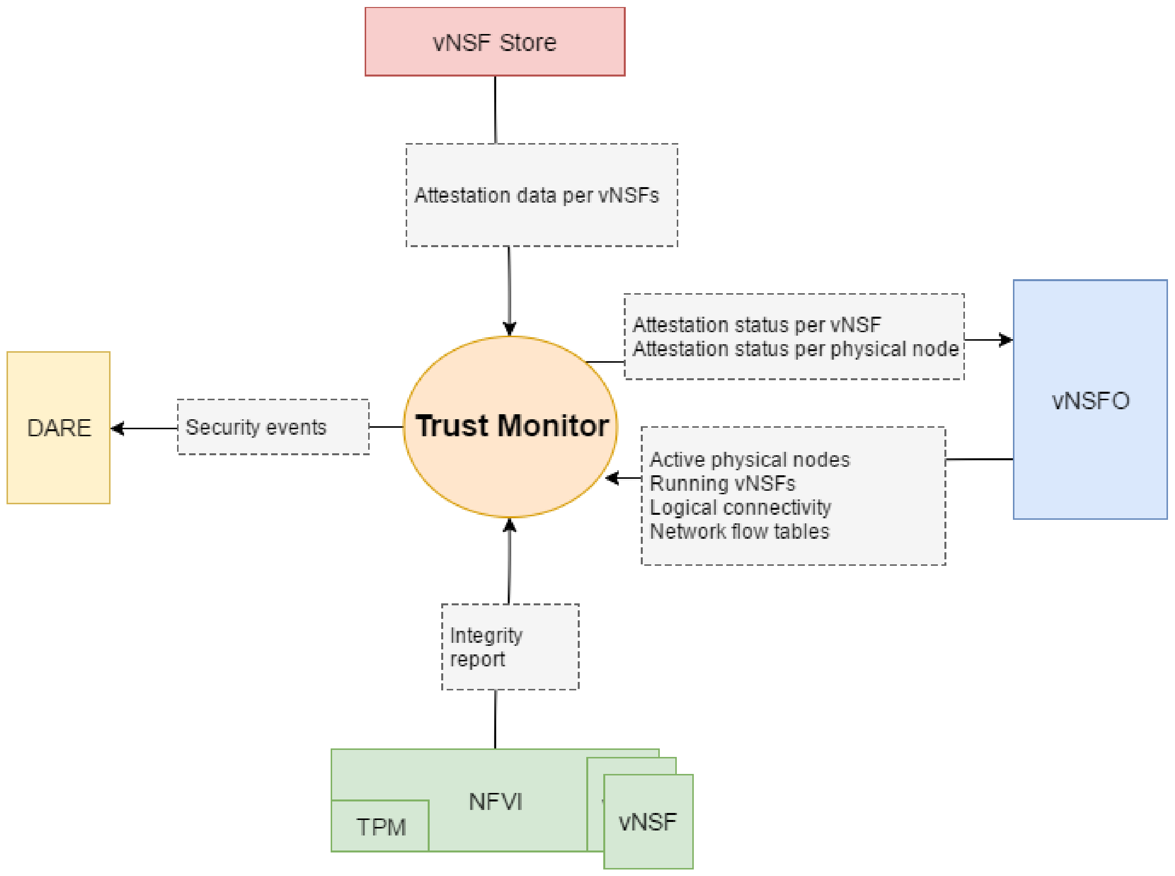


Figure 18: Data flow diagram of Trust Monitor

2.4.4. Internal operation

To assess the trust of the NFVI (both physical nodes and virtual instances), the Trust Monitor needs to keep an updated list of known measurements about software packages and valid configurations. To do so, it interacts with the vNSF Store to retrieve the information needed for performing attestation of vNSFs, packaged within the security manifest of each network function's instance. Additionally, the Trust Monitor can download and measure packages of various Linux distributions from the official repositories, and can also keep internal knowledge of the software updates for each of them. This particular data is used by the TM to attest the infrastructure nodes and rate them with different trust levels (e.g. by considering untrusted a node with a known software vulnerability).

The Trust Monitor is also able to keep an updated view of the network infrastructure at a given time by a specific interaction with the vNSFO, which in turn updates the Trust Monitor with status changes of the NFVI. This information can then be utilised by the Trust Monitor to periodically attest the NFVI, to detect any compromised node. In addition, the vNSFO could directly ask the Trust Monitor to attest a node joining the NFVI, referred as "newcomer".

The whitelist of known measurements can be used for checking the integrity report provided by each physical node of the NFVI during the Remote Attestation workflow. If any of the verification steps fail, the Trust Monitor is in charge of notifying the failure to the vNSFO and also log the event in the DARE.

2.4.5. Interaction with other components

The Trust Monitor interacts with the Store, vNSFO and DARE components of the SHIELD's infrastructure to request attestation-related information or as a response of an external attestation request. A detailed description of each workflow is presented in the "Appendix B Inter-component interactions".

3. Specifications and Implementation

The information conveyed in this section decreases the abstraction level for the software solution provided. Based on the components and sub-components defined in the architecture section it presents additional insight on the inner details of said sub-components by defining implementation-oriented behaviours, operations and interactions. Such behaviours may be supported by software design elements such as data flows, state machines, decision flows or API/interfaces descriptions.

Targeting an implementation-oriented approach this section references possible technologies or features from existing technologies to use, reused outcomes or extensions to develop based on other projects or even specify features to create from scratch. To assist the reader in understanding how the selected technologies fits within SHIELD rationale, the requirements fulfilment is also included.

3.1. Security network functions and services

This section describes the vNSFs identified so far to perform monitoring and remediation within the scope of the SHIELD platform. For each of them a mapping of its functionality against a subset of the SHIELD requirements is provided, as well as low level specification and implementation details when available.

3.1.1. Virtual Intrusion Detection System (vIDS)

3.1.1.1. Implementation details

For the implementation of a virtualised Intrusion Detection System in SHIELD it is planned to adopt the IDS VNSF [12] that was developed in the frame of CHARISMA project. Several modifications and extensions will be made to support full compatibility with the SHIELD platform.

The vIDS vNSF, as used in CHARISMA, includes the following components:

- Snort IDS: An open-source intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks.
- Barnyard2: An open-source software tool that takes Snort output and writes it to a SQL database to reduce load on the system.
- PulledPork: An open-source tool that automatically downloads the latest Snort rules (threat signatures).
- Snorby: An open-source web-based graphical interface for viewing and clearing events logged by Snort.
- **Rule Configuration Service:** A service that accepts requests for creating, deleting and modifying rules that can be applied in Snort detection engine.
- Event Publisher Service: A service responsible for publishing the alerts produced by Snort detection engine.

The current CHARISMA IDS vNSF implementation is based on Ubuntu 14.04 operating system, which was selected as the guest operating system in CHARISMA project. Incoming traffic to the IDS vNSF is being analysed in real time and analysis decisions are being communicated to external interfaces as HTTP requests. This vNSF consists of one virtual machine which requires to have one virtual network interface where all traffic that need to be monitored must be routed (or mirrored). Thus, the CHARISMA IDS has a single vNSFC. Additionally, the vNSF is accompanied by an ETSI compliant descriptor that allowed its life-cycle management through the TeNOR (T-NOVA) orchestrator.

The IDS implementation is based on Snort open source IDS. Snort [13] is an open-source intrusion detection system that is developed by Sourcefire. It is capable of performing real-time traffic analysis and packet logging on IP networks. Snort architecture is composed by the packet capture library, the packet decoder, the preprocessor, the Snort detection engine which is configured with detection rules and the alert output components plug-ins.

Rule Configuration Service

To provide intrusion detection functionalities based on policy defined by external modules to the vIDS, this VNSF implements a RESTful API which accepts requests for creating, deleting and modifying rules that can be applied in Snort detection engine. This offers an easy way of external configuration of the VNSF without requiring knowledge of its inner workings.

Event Publisher Service

The IDS VNSF provides another functionality, necessary for the utilisation of the results produced by Snort packet analysis, the Event Publisher Service. This service translates, curates, and publishes events in readable format to external interfaces for further analysis. Once traffic enters the IDS vNSF, Snort software analyses all packets. Snort detection engine, described above, can contain rules which consist of conditions. When the conditions of a rule are met, the detection engine produces an event and saves it in a log file. Snort event logs are saved in Unified2 format so the Event Publisher Service translates them to JSON format, assesses their timestamp to avoid publishing redundant information and publishes the events.

A number of modifications to the CHARISMA IDS vNSF to make it compatible with the SHIELD platform are foreseen. More specifically:

- **Virtualisation enabler:** A CentOS 7.X will be used as the guest operating system to provide a virtual machine-based IDS for SHIELD. Additionally, a second version of the IDS will be provided bundled in a Docker container or -if required- multiple Docker containers.
- **vNSF descriptor:** The vNSF descriptor of the vIDS will have to be implemented from scratch to allow life-cycle management through the OSM orchestrator.
- **Rule Configuration Service**: This component matches the configuration listener element included in all vNSFs that accept configuration through the Security orchestrator. Modifications to the current implementation are expected to allow compatibility with the Security Orchestrator and the exact format of the policies sent.
- **Event Publisher Service**: This component matches the streaming service element included in all vNSFs that provide monitoring information data to the DARE. Modifications to the current implementation are expected to allow compatibility with the data format expected from the DARE Streaming Service.

- **User interface and output**: As SHIELD platform features a User Dashboard for displaying output and threat alerting and notifications to the user, it is unlikely that the Snorby GUI component will be required for the SHIELD vIDS implementation.

3.1.1.2. Requirements mapping

Requirement	Requirement name	Requirement description
SF08	DoS Protection	A security service SHALL protect against volumetric Denial of Service attacks. Detect the DoS attack and divert the traffic for filtering. Forwarding the good traffic flows to the destination.
VI_SPEC_01	vIDS will perform traffic analysis against its signatures database to detect a DoS attack and notify DARE about it; which will in turn instruct specific mitigation procedures.	
SF09	Intrusion Detection/Prevention System	A security service SHALL detect attacks with a wide range of techniques such as network flow or behaviour analysis and deep packet inspection. Allow traffic flows according to IPS rules. Monitor traffic network traffic at OSI layer 7 and generate alerts for security policy violations, infections, information leakage, configuration errors and unauthorised clients.
VI_SPEC_02	vIDS will analyse the traffic in L3-L4 and L7, generating appropriate alerts upon any detected intrusion and notify DARE regarding identified security threats or incidents. After internal analysis and correlation, DARE will instruct specific mitigation procedures.	
NF05	Impact on perceived performance	When network traffic is proxied or analysed, the user experience SHALL not be degraded.
VI_SPEC_03	The traffic analysis carried out by this IDS should not seriously delay or degrade the detection and mitigation operations.	

3.1.2. Virtual Deep Packet Inspection (vDPI)

3.1.2.1. Implementation details

The implementation of the vDPI components is based on a variety of technologies allowing to perform traffic inspection as well as packet capturing. The following technologies are currently envisioned to be used in the implementation of this vNSF:

- **nDPI** [14]: is an open source alternative to the OpenDPI [15] library, maintained by ntop. Its goal is to extend the original library and add new protocols that are otherwise available only on the paid version of OpenDPI. Furthermore, nDPI is modified to be more suitable for traffic monitoring applications, by optimising the DPI engine. One of its major advantages is that nDPI can support application-layer detection of protocols, regardless of the port being used.
- **PF_RING** [16]: is a set of library drivers and kernel modules, which enable high-throughput packet capture and sampling. The PF_RING kernel module library polls packets through the Linux NAPI. Packets are copied from the kernel to the PF_RING buffer for analysis with the nDPI library.
- **DPDK (Data Plane Development Kit)** [17]: comprises of a set of libraries that support efficient implementations of network functions through access to the system's network interface card (NIC). DPDK offers to network function developers a set of tools to build high speed data plane applications. DPDK operates in polling mode for packet processing, instead of the default interrupt mode. The polling mode operation adopts the busy-wait technique, continuously checking for state changes in the network interface and libraries for packet manipulation across different cores.

A PF_RING implementation has the capacity of maintaining uninterrupted connectivity with the OpenStack network. DPDK has the capacity to bypass the Linux kernel, leading to high-performance packet capture but less robust and fault-tolerant that PF_RING.

3.1.2.2. Requirements mapping

Requirement	Requirement name	Requirement description
SF02	Detect/Block access to malicious websites	The vDPI can block access to known malicious websites including (but not limited to) phishing websites, known malware Command and Control servers, Ransomware Command and Control servers and payment sites. vDPI can also block access depending on application types (e.g. SMB connections, IRC, RDP etc. that can indicate the potential presence of a backdoor). This does not include deep content inspection.

VD_SPEC_01	vDPI will be able to redirect, limit or block suspicious traffic based on already established rules. Other suspicious traffic can be redirected to DARE for analysis and thus the vDPI should be able to receive new policy configurations.	
SF08	DoS Protection	A security service SHALL protect against volumetric Denial of Service attacks. Detect the DoS attack and divert the traffic for filtering. Forwarding the good traffic flows to the destination.
VD_SPEC_02	vDPI will be able to indicate suspicious traffic and redirect to DARE for inspection. Some known (D)DoS attack types (e.g. Ping of death, application level flooding etc.) can be blocked by common rules and policies. vDPI will not be performing behavioural analysis; this will be performed in DARE to assess legitimacy of other examined traffic flows.	
SF09	Intrusion Detection/Prevention System	A security service SHALL detect attacks with a wide range of techniques such as network flow or behaviour analysis and deep packet inspection. Allow traffic flows according to IPS rules. Monitor traffic network traffic at OSI layer 7 and generate alerts for security policy violations, infections, information leakage, configuration errors and unauthorised clients.
VD_SPEC_03	vDPI will offer deep packet inspection capabilities based on the nDPI library. Capabilities include inspection of packet headers, applications types etc., but not deep content inspection (which requires reassembly and inspection of an entire message)	
NF05	Impact on perceived performance	When network traffic is proxied or analysed, the user experience SHALL not be degraded.
VD_SPEC_04	The traffic inspection performed by vDPI should not seriously degrade the user's quality of experience on the NS. vDPI engine will be based on open source high-throughput tools (nDPI, PF_RING etc) and will be able to parse small subsets of mirrored traffic.	

3.1.3. mcTLS Middlebox and Gateway

3.1.3.1. Implementation details

This vNSF is going to be available in Y2, so implementation details are not yet available. It will be based in the mcTLS Open Source project [18] over Ubuntu.

3.1.3.2. Requirements mapping

Requirement	Requirement name	Requirement description
SF09	Intrusion Detection/Prevention System	A security service SHALL detect attacks with a wide range of techniques such as network flow or behaviour analysis and deep packet inspection. Allow traffic flows according to IPS rules. Monitor traffic network traffic at OSI layer 7 and generate alerts for security policy violations, infections, information leakage, configuration errors and unauthorized clients.
VM_SPEC_01	The vNSF (middlebox) allows the monitoring of ciphered traffic directed at a specific server (HTTPs server with a mcTLS Gateway vNSF) in order to identify attacks.	
NF05	Impact on perceived performance	When network traffic is proxied or analysed, the user experience SHALL not be degraded.
VM_SPEC_02	When network traffic is proxied or analysed, the user experience SHALL not be degraded.	

3.1.4. HTTP/S Analyser

3.1.4.1. Implementation details

This vNSF it is going to be available in Y2, so implementation details are not yet available. It is going to be based in machine learning techniques to provide the HTTP/s traffic classification.

3.1.4.2. Requirements mapping

Requirement	Requirement name	Requirement description

SF09	Intrusion Detection/Prevention System	A security service SHALL detect attacks with a wide range of techniques such as network flow or behaviour analysis and deep packet inspection. Allow traffic flows according to IPS rules. Monitor traffic network traffic at OSI layer 7 and generate alerts for security policy violations, infections, information leakage, configuration errors and unauthorized clients.
VH_SPEC_01	Traffic classification will allow the classification of traffic traversing the network and therefore enable its correlation with potential attacks therefore improving its detection/mitigation mechanisms.	
SF08	DoS Protection	A security service SHALL protect against volumetric Denial of Service attacks. Detect the DoS attack and divert the traffic for filtering. Forwarding the good traffic flows to the destination.
VH_SPEC_02	Traffic classification will allow the classification of traffic traversing the network and therefore enable its correlation with potential attacks therefore improving its detection/mitigation mechanisms.	
NF05	Impact on perceived performance	When network traffic is proxied or analysed, the user experience SHALL not be degraded.
VH_SPEC_03	When network traffic is proxied or analysed, the user experience SHALL not be degraded.	

3.1.5. L3 Filter

3.1.5.1. Implementation details

The implementation of this vNSF will be based on the packet filtering framework included within the Linux kernel, starting from the 2.4 version. The framework, maintained by the netfilter.org project, consists of different subsystems, such as iptables [19]. This userspace program can be used to configure the filtering ruleset, composed of rules consisting of classifiers (e.g. the source IP address) and one connected action (e.g. deny).

The vNSF will provide an Access Control List in a standard format, such as XML, containing a list of IP addresses to be allowed or denied, depending on the kind of list (whitelist, blacklist). The vNSF will manage the low-level translation of the ACL to iptables rules.

An implementation for a L3 packet filter, based on iptables, has been developed in the scope of the SECURED project [4], and will be considered as base point for development of this vNSF.

3.1.5.2. Requirements mapping

Requirement	Requirement name	Requirement description
SF09	Intrusion Detection/Prevention System	A security service SHALL detect attacks with a wide range of techniques such as network flow or behaviour analysis and deep packet inspection. Allow traffic flows according to IPS rules. Monitor traffic network traffic at OSI layer 7 and generate alerts for security policy violations, infections, information leakage, configuration errors and unauthorised clients.
VL3_SPEC_01	The L3 Filter vNSF is expected to be integrated with the IDS/DPI as a remediation vNSF, blocking any malicious traffic detected by the IDS.	
NF05	Impact on perceived performance	When network traffic is proxied or analysed, the user experience SHALL not be degraded.
VL3_SPEC_02	The filtering operation performed by the vNSF should not seriously degrade the user's quality of experience on the NS.	

3.1.6. Forward L7 Filter

3.1.6.1. Implementation details

This vNSF will be implemented by leveraging the functionalities offered by the Squid web cache [20] for its internal logic. This software can inspect traffic at application layer (e.g. HTTP, FTP, Gopher, WAIS) to filter specific URLs and provide ACL management. Squid is also able of acting as a web cache, even though this functionality is considered out of scope for the proposed vNSF. In addition, Squid may be configured as a Transparent Proxy, which would also require the redirection of incoming HTTP traffic to the port Squid is running on (e.g. via an iptables rule). By interacting with certain plugins, such as DansGuardian web content filter, the vNSF would be able to grant or deny access to a web page depending on its actual content (e.g. phrase matching) other than URL. The filtering capabilities managed by Squid are the most relevant ones for implementing this reacting vNSF.

An implementation for a L7 filter, based on Squid, has been developed in the scope of the SECURED project [4], and will be considered as base point for development of this vNSF.

3.1.6.2. Requirements mapping

Requirement	Requirement name	Requirement description
SF02	Detect/Block access to malicious websites	A security service SHALL control access to malicious websites, such as phishing servers, malware spreading, C&C servers, etc. The user must be alerted and the access to the site could be blocked/allowed depending on the configured policy rule.
VL7_SPEC_01	The Forward L7 Filter vNSF will be able of blocking L7 traffic depending on different criteria in order to control access to malicious websites (such as by filtering HTTP data according to blacklists of URLs).	
NF05	Impact on perceived performance	When network traffic is proxied or analysed, the user experience SHALL not be degraded.
VL7_SPEC_02	The filtering operation performed by the vNSF should not seriously degrade the user's quality of experience on the NS.	

3.2. Store

Based on the general architecture of the Store component provided in previous sections, the present section aims providing a preliminary specification of its low-level functionalities.

3.2.1. Specifications

For each subcomponent of the Store component, the low-level specifications are provided below.

Lifecycle Manager

Lifecycle Manager subcomponent is responsible for implementing a set of features that enable the envisioned onboarding lifecycle of either vNSFs and NSs. The vNSF/NS onboarding lifecycle comprises the following steps:

- **Submitted** A vNSF has been submitted to the Store for onboarding by a Developer. Due to the nature of the process, as it comprises time-consuming operations such as validations and considerable-sized downloads, the submission request is promptly acknowledged and the process continues in the background. Later on, the Developer will be notified whether the operation succeeded or failed.

- **Sandboxed:** A vNSF is registered in the Catalogue but is not yet ready for production. It is undergoing a validation process to determine whether it is deemed fit for service.
- **Onboarded:** A vNSF has successfully undergone all the required checks to be considered able to integrate the ecosystem and is fit for attestation tests.
- **Decommissioned**: A vNSF has been taken out of service and can no longer be instantiated.

Descriptors Validator

To ensure a vNSF/NS can be onboarded, the descriptors provided in the package need to be validated. These descriptors are checked for:

- Syntax errors to prevent incorrect vNSF descriptors from being processed.
- vNSF topology integrity to avoid potential loops or errors such as references to undefined network interfaces.

Every onboarded vNSF descriptor will be checked for syntax, correctness and completeness issues. With no issues found the next step is to check the defined network topology and ensure inconsistencies such as no unconnected interfaces are present and all virtual links are properly defined. Upon successful validation, the vNSF may proceed with the onboarding process. Any error results in a notification to the Developer stating what is not compliant with the SHIELD requirements. As for Network Services, onboarding the descriptors provided in the package need to be validated. These descriptors are checked for:

- Syntax errors to prevent incorrect NS descriptors from being processed.
- vNSF/NS topology integrity to avoid potential loops or errors such as references to undefined network interfaces
- Decommissioned vNSF usage to avoid service instantiation issues

Again, every NS descriptor will be checked for syntax, correctness and completeness issues. With no issues found the next step is to check whether any usage of decommissioned vNSF is present. Upon successful validation, the NS may proceed with the onboarding process. Any error results in a notification to the Security Dashboard stating what isn’t compliant with the SHIELD requirements.

Integrity Checker

The vNSF onboarding security check is performed by:

- Verifying the package digital signature against the stored one to prove provenance.
- Checking the hashes for the vNSF-related files against the ones provided in the manifest to ensure integrity.

The security manifest format is defined by SHIELD and all submitted vNSFs, regardless of intended target vNSFO, shall comply with it (no tailoring is allowed). Upon successful checks the vNSF may proceed with the onboarding process. Any error results in a notification to the Developer stating what is not compliant with the SHIELD requirements.

Catalogue

The Catalogue handles the records for the entire Store component. It stores data of all the onboarded vNSFs and NSs and can convey it to the other components upon request through the adapters provided for such purpose. The specific data is defined below:

- vNSF Catalogue
  - Version: an identifier for the submitted vNSF package which defines a unique set of specific functionalities and dependencies provided within the vNSF-related descriptors.
  - Status: the current status of the vNSF. It can be “submitted”, “sandboxed”, “onboarded” or “decommissioned”.
  - Security manifest: holds the hashes for all the vNSF-related files as well as information needed for attestation.
  - vNSF Descriptor (vNSFD): description for the vNSF, containing the vNSFCs that conform the vNSF, the available flavours to deploy and the description of the virtual links interconnecting the different vNSFCs.
- NS Catalogue
  - Version: an identifier for the submitted NS package which defines a unique set of specific functionalities and dependencies provided within the NS-related descriptors.
  - Status: the current status of the NS be it submitted, sandboxed, onboarded or decommissioned.
  - Security manifest: holds the hashes for all the NS-related files as well as information needed for attestation.
  - NS Descriptor (NSD): description for the service, containing the vNSFs that conform the service and their forwarding graphs, the virtual link description interconnecting the vNSFs, the preferred flavour (instance configuration) per vNSF to use and any SLA to be met by the NS.
  - Virtual Link Descriptor (vLD): definition of the virtual network links that interconnect the vNSFs.
  - vNSF Forwarding Graph Descriptor (vNSFFGD): definition of the network deployment for the vNSFs contained in the NS.

Developer Adapter

This module provides connectivity with the Developer either in the form of an API for the Developer to use Store’s features as well as a connector allowing Store to push information to the Developer.

Dashboard Adapter

This module provides connectivity with the Security Dashboard component either in the form of an API for the Dashboard to use Store’s features as well as a connector allowing Store to use Security Dashboard’s functionalities.

Orchestrator Adapter

This module provides connectivity with the Orchestrator component either in the form of an API for the Orchestrator to use Store's features as well as a connector allowing Store to use Orchestrator's functionalities.

Trust Monitor API

This module provides connectivity to Trust Monitor component in the form of an API.

DARE API

This module provides connectivity to DARE component in the form of an API.

3.2.2. Implementation details

The Store component will leverage existing technologies that already address some of the features intended for its implementation. Currently the following implementations are being analysed to be used as a basis or as an extension of SHIELD's store component:

- SONATA catalogue
- SONATA NS/VNF syntax validation features
- SONATA NS/VNF topology validation features
- TeNOR catalogue
- OSM catalogue
- OSM NS/VNF descriptors

REST API Services will be used to expose an interface to access Store's internal features. Further specifications comprising the envisioned APIs can be found in "Appendix D Application Programming Interfaces (APIs)".

3.2.3. Requirements mapping

Requirement	Requirement name	Requirement description
PF02	vNSF lifecycle management	The platform SHALL be able to manage the full lifecycle of vNSFs (on boarding, instantiation, chaining, configuration, monitoring and termination).
S_SPEC_01	The Store provides the Developer with an interface to onboard a vNSF and the Security Dashboard with another interface to onboard NSs. It also provides an interface to vNSFO to query vNSF and NS information during instantiation. The remaining states for the vNSF lifecycle management are outside the scope of the Store.	
PF10	vNSF validation	The store SHALL validate that the image of a vNSF is not manipulated, faked or invalid.

S_SPEC_02	At the time of a vNSF submission by the Developer the Integrity Checker ensures that the vNSF content is trusted and stores (amongst other data) the hash(es) for the vNSF image(s) which can be provided upon request for integrity checks by other components.	
PF11	vNSF attestation	The platform SHALL check the provenance and integrity of a vNSF and associated policies, before it starts to operate.
S_SPEC_03	When the Developer submits a vNSF the Integrity Checker validates the digital-signature associated with it to verify the provenance of the submitted data and analyses its integrity to ensure it wasn't tampered with in any way. This data is stored and can be provided upon request for attestation purposes to other components.	
PF15	Service store	The store SHALL allow selecting security services from the catalogue.
S_SPEC_04	A record of the successfully onboarded Network Services is kept by the Catalogue. These security services are provided upon request through the Store's interfaces.	
PF17	Interoperability	The platform SHALL expose openly-defined APIs for information exchange with third parties.
S_SPEC_05	The Store provides the interoperability features through APIs and connectors. The vNSF onboarding is accomplished by the Develope's API, the NS onboarding and store-related GUI interaction is done by the Dashboard API, the vNSF and NS data concerning orchestration is provided by the Orchestrator API and the attestation-related data is conveyed by the Trust Monitor API.	

3.3. Orchestrator

When analysing the Platform Requirements (described in D2.1), four well-known NFV MANO solutions were identified and analysed. These are OSM [3], TeNOR [21], SONATA [2] and OpenBaton [22]. OSM stems from industrial community, whilst TeNOR, SONATA and OpenBaton have grown in the R&D environment.

To carry out the analysis, the accordance with the Platform Requirements (as defined in D2.1) was examined, along with several extra indicators; from more subjective, like the extensibility and complexity degree in terms of development, to others such as its ongoing and future roadmap as well as its community. When considering how appropriate are the provided features to the SHIELD's Platform Requirements, OSM and TeNOR provide mostly the same capabilities; with more support by the former to extra VIMs and SDN controllers, and on monitoring and operational capabilities on the latter. SONATA and OpenBaton show a focus on specific aspects (the former focusing on identity management, the latter in service operations). These are at different stages of development, being SONATA under development and OpenBaton a more consolidated orchestrator. Both provide advanced features on their field of focus, and provide extensive documentation. A detailed analysis can be found in the "Appendix E Technology Selection". After evaluating the aforementioned indicators and prioritising the Platform Requirements, the community and available support, the consortium decided to use OSM as the base vNSFO for SHIELD.

3.3.1. Specifications

The low-level specifications of the subcomponents of the Orchestrator are provided below.

NS Manager

The NS Manager supports issuing the following operations on the NS:

- **NS instantiation:** initial validation and deployment of the vNSF and NS instances, according to the lifecycle events defined in the vNSF and NSD and triggered from the latter. The operations on the vNSFs are delegated to the vNSF Manager
- **NS configuration:** changes in the configuration of any given NS through its descriptor, whether these are done prior to starting the NS or as an active update while the NS is running. One of the possible changes encompassed is the distribution of policies to be applied within vNSFs
- **NS monitoring:** monitoring of the NS performance. The metrics from the network links and the compute instances and service-specific data are retrieved from the NFVI and the VNFM. The operations on the vNSFs are delegated to the vNSF Manager
- **NS scaling:** increase or decrease of the NS capacity according to the auto-scaling policies defined per vNSF and NS in their descriptors. The scaling can result in increasing/decreasing capacity per vNSF, creating or terminating vNSF instances and adjusting the number of links between vNSFs
- **NS termination**: release any given NS instance and its associated resources (vNSF instances, NFVI-related resources, connecting links between vNSFs)

vNSF Manager

The vNSF Manager supports issuing the following operations on the vNSFs:

- **vNSF instantiation:** initial validation and deployment of the vNSF instances in the NFVI
- **vNSF configuration:** changes in the configuration of the deployed vNSF, whether these are done prior to starting the vNSF or during runtime. Some possible changes are the introduction of user-specific attributes
- **vNSF monitoring:** monitoring of the performance information of the instance, obtained from the NFVI; and of the vNSF itself, providing service-specific metrics within the vNSF instance

- **vNSF scaling:** increase or decrease of the vNSF capacity through adding (scale-out) or removing (scale-in) compute nodes associated to the vNSF
- **vNSF termination:** release any given resource from the NFVI that is related to the vNSF

Repositories

Two repositories are expected to persist information on the running vNSFs and NSs, along with physical and virtual nodes from the NFVI.

- **vNSF and NS instance repository:** A registry of the running virtual instances for both NSs and vNSFs, the status per vNSF and any data related to them, such as low-level information of the NFVI (IDs of each record/running instance of NSs and vNSFs, IPs per vNSF, etc). Such records are consulted by different processes, such as the monitoring carried out in the NS manager. This role is covered by the VIM, abstracting the NFVI details.
- **Infrastructure repository:** This repository keeps a list of which resources from the NFVI are available, reserved or allocated. The NS Manager consults this to perform operations such as the validation prior to the deployment of a requested NS and the mapping of its resources into the physical infrastructure.

Dashboard API

This exposes a read-only API that provides necessary information on the resources in the NFVI. This is used by the Security Dashboard to present its graphical view to the user.

Trust Monitor API

This exposes an interface for the Trust Monitor to retrieve information on the NFVI, which can be used to perform the periodic attestation task.

DARE API

This interface provides DARE with a global view on the infrastructure. This is done by offering information on the physical nodes, the running and available vNSFs, etc. The analytics leverage on this and external monitoring data sources.

3.3.2. Implementation details

The vNSF Orchestrator will be based on the OSM solution. The following software modules and technologies will be used to fulfil the orchestration:

- Service Orchestrator (SO)
    - Acting as the NS Manager, RIFT.ware provides end-to-end network service orchestration, abstracting from computing resources, and provisioning lifecycle management and interconnection of VLs
- Resource Orchestrator (RO)

    - OpenMano enables operations from both the Infrastructure and vNSF Managers. It provisions resources as needed, interacting with multiple VIMs and SDN controllers. Along with SO, these conform the NFVO entity in the ETSI NFV architecture
- vNSF Configuration and Abstraction (VCA)
    - Generic vNSF Manager allowing the initial vNSF configuration (pre-boot). It relies on Canonical's Juju charms and cloud-init to provide instructions to the vNSFs to be deployed

These modules can be mapped of the ETSI NFV architecture as depicted in Figure 19

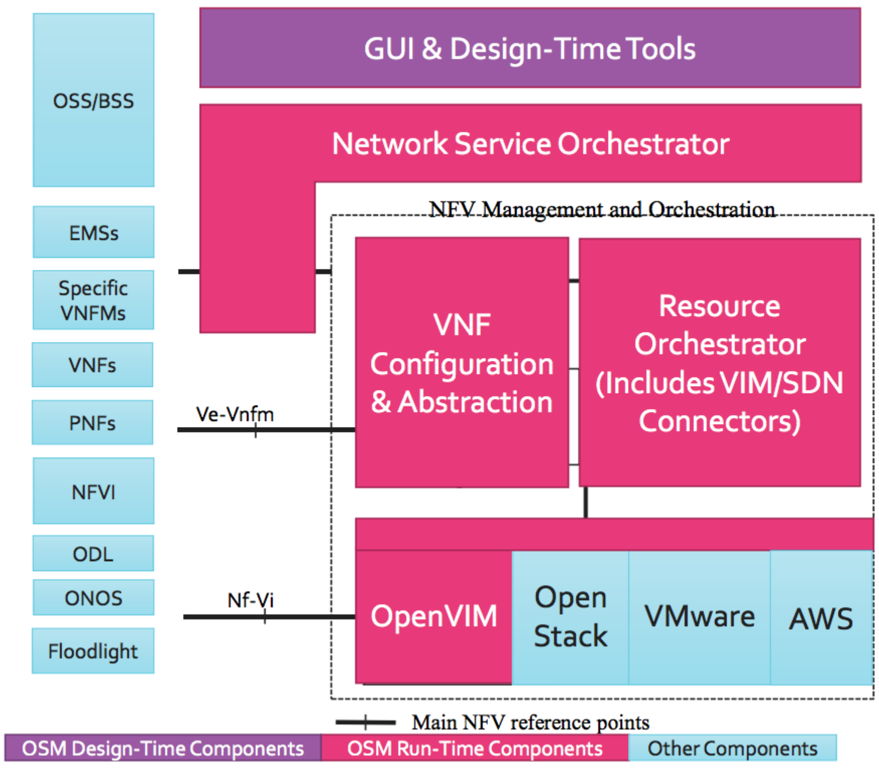

GUI & Design-Time Tools
Network Service Orchestrator
NFV Management and Orchestration
OSS/BSS
EMSs
Specific VNFMs
VNFs
PNFs
NFVI
ODL
ONOS
Floodlight
Ve-Vnfm
Nf-Vi
VNF Configuration & Abstraction
Resource Orchestrator (Includes VIM/SDN Connectors)
OpenVIM
Open Stack
VMware
AWS
Main NFV reference points
OSM Design-Time Components
OSM Run-Time Components
Other Components

Figure 19: OSM mapped to ETSI NFV architecture

More details on the implementation and deployment details can be found in the documentation and whitepaper for the latest release (R2) [23][24].

The details on the specific development for the interfaces and connectors described earlier are provided in the "Appendix D Application Programming Interfaces (APIs)".

3.3.3. Requirements mapping

Requirement	Requirement name	Requirement description
PF01	vNSF and Network Service (NS) deployment	The platform SHALL be able to deploy the vNSFs in different PoPs and domains. The deployment can occur within internal or external premises.

O_SPEC_01	The NS and vNSF Managers can initiate the deployment of the vNSFs in the different PoPs; as these are previously registered into the vNSFO.	
PF02	vNSF lifecycle management	The platform SHALL be able to manage the full lifecycle of vNSFs (on boarding, instantiation, chaining, configuration, monitoring and termination).
O_SPEC_02	The NS and vNSF Managers can control the different stages in the lifecycle of the vNSFs.	
PF03	vNSF status management	The operator SHALL be able to control the lifecycle via a graphical user interface. The vNSF lifecycle should support events like DEPLOY, START, STOP, MODIFY, DELETE.
O_SPEC_03	The NS and vNSF Managers will receive lifecycle events from Security Dashboard to deploy or instantiate, run, stop, configure or delete the vNSFs.	
PF07	Service elasticity	The platform COULD provide the mechanism to allow scalability of the vNSFs.
O_SPEC_04	The NS and vNSF Managers provide the capability to request a specific NS or vNSF to adapt (scale) to its operational conditions.	
PF11	vNSF attestation	The platform SHALL check the provenance and integrity of a vNSF and associated policies, before it starts to operate.
O_SPEC_05	The vNSFO checks the associated policies before configuration.	
PF13	Mitigation	The platform SHALL be able to trigger, in the case of an event, proper actions to mitigate the threat.
O_SPEC_06	As the result of an accepted suggestion by a user in the Dashboard, the NS and vNSF Managers receive and distribute requests to deploy specific mitigation NSs.	
PF17	Interoperability	The platform SHALL expose openly-defined APIs for information exchange with third parties.
O_SPEC_07	The orchestrator will provide different APIs to interact with the DARE, the Trust Monitor and the Security Dashboard.	

PF19	Network infrastructure attestation	The platform SHALL verify that the network infrastructure that executes the vNSF is in a trusted state (network elements and server identity, software, configuration).
O_SPEC_08	The vNSFO provides information on newcomer nodes on the NFVI; so that the Trust Monitor can periodically attest those.	
NF03	Scalability	The platform SHALL be expandable by adding nodes in the network infrastructure, to increase capacity.
O_SPEC_09	The vNSFO interacts with the VIM and is aware of the existing and newcomer nodes in the NFVI, which will be later on provided to the Trust Monitor.	

3.4. Trust monitor

The general architecture and design of the Trust Monitor has been defined according to the Platform Requirements, as defined in D2.1. This section aims to describe the specifications of the low-level functionalities that will be developed within the Trust Monitor sub-components.

3.4.1. Specifications

The low-level specifications of each subcomponent of the Trust Monitor are reported as follows, as well as the mapping of the specifications to the PFRs. The specifications may be subject to minor modifications during the development stage.

Verifier

The Verifier is the central sub-component of the Trust Monitor. It manages different functionalities:

- Registration of a node
- On-demand attestation of a node
- Periodic attestation of the nodes in the NFVI
- Notification of attestation failure to both the DARE and the vNSFO

The registration phase is needed to setup the attestation process with each NFVI PoP composing the network infrastructure. Each node of the NFVI should be properly configured to enable its interaction with the TPM and to start measuring the software running into it. The remote attestation procedure is performed both in the initial attestation of newcomers and periodic attestation tasks. It requires the Verifier to perform the following operations:

1. Send an attestation request to the node, including a nonce for freshness of the response
2. Validate the response
3. Extract the software measurements from the integrity report, consisting of the software and configuration utilised by both the host and the vNSFs running into it

4. Verify the integrity measurements of the host against the reference values contained in the Whitelist Database
5. For each vNSF, verify the integrity measurements against the known digests contained in the vNSF security manifest
6. For SDN-controlled switches, verify the SDN forwarding rules with regards to the expected one (in the SDN controller)

The Verifier can verify the measurements of the host by leveraging the Whitelist Database functionality. The known measurements of each vNSF can be retrieved via the API exposed by the vNSF Store. In case of failure during attestation, the Verifier leverages the APIs provided by both the vNSFO and the DARE. Periodic attestation should be performed by an internal task that leverages the API offered by the vNSFO to retrieve the "map" of the current status of running nodes in the NFVI.

DARE Connector

The Trust Monitor should be able to collect relevant information from the NFVI in real time to verify the nodes' software integrity. This information is used to detect security incidents regarding misuse of a node. In case of failure upon attestation, a security event is sent by the Trust Monitor to the DARE. This information is logged by the DARE and it could also be shared with a third entity.

Whitelist Database

The database contains the complete data of the executables allowed on the attested platforms. More specifically, for example on Linux-based platforms, it contains the following information for each file to be measured:

- The digest
- The full path name
- The packages in which it is contained (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 database should store the history of each package, reporting the information about its updates (e.g. the type of update). Given the packages' history, the Verifier verifies the IMA log at one of the following trust levels:

- **Level 1:** TPM and IMA measurement in the node is running correctly
- **Level 2:** In addition to Level 1, all the software is found in the reference database but there is at least one with a known security vulnerability
- **Level 3:** In addition to Level 2, at least one binary has a known functional bug
- **Level 4:** In addition to Level 3, no known security vulnerabilities or functional bugs are found in the measured software

vNSF Store Connector

This subcomponent allows the retrieval of the security manifest for each vNSF to be attested.

vNSFO Connector

The vNSF Orchestrator is in charge of terminating nodes of the NFVI if their execution cannot be trusted. Therefore, the Trust Monitor is in charge of notifying both the vNSFO and the DARE in case of remote attestation failure. In addition, the Trust Monitor should have a clear view of the vNSFs and NFVI PoPs deployed in the SHIELD infrastructure, in order to perform the periodic attestation of running nodes. To do so, it will leverage a specific functionality offered by the vNSFO API.

Newcomer Attestation API

The sub-component exposes an API for on-demand registration and attestation of newcomers in the NFVI.

Management API

The sub-component exposes a read-only API for checking the status of the Trust Monitor, and retrieving relevant information about the attestation of the infrastructure.

3.4.2. Implementation details

The Trust Monitor implementation starts from components that have been developed in the EC-funded project SECURED [4]. More specifically, the following technologies could be reused:

- Third-party Verifier based on Open Attestation v1.7 [5]
- Whitelist Database based on Apache Cassandra 2 [6]
- SDN-enabled switch attestation prototype [7]

These technologies, representing the starting point for the development stage, are bound to a Linux CentOS 7 environment equipped with TPM 1.2 device. The development efforts in the project are aimed to enrich the already available software with the SHIELD-specific APIs.

Additionally, the Trust Monitor should be able to support TPM 2.0-enabled hardware, meaning that the attestation framework needs further improvements. Regarding this point, the OpenCIT [25] framework, developed by Intel, will be exploited as an evolution to the Open Attestation framework.

As stated in the official website of the project, Open Attestation no longer receives any update and it does not provide support for the TPM 2.0 devices. On the opposite side, OpenCIT does not support the integrity report workflow at the moment of writing, meaning that further improvements are needed over the mainstream version.

The details on the specific development for the interfaces and connectors described earlier are provided in the “Appendix D Application Programming Interfaces (APIs)”.

3.4.3. Requirements mapping

Requirement	Requirement name	Requirement description
PF08	Platform expandability	The platform SHALL be easily extended to support new security services.
T_SPEC_01	The Trust Monitor provides documented APIs and interfaces to enable the interaction with the different components. In addition, the component provides a generic client-service workflow to attest the nodes in the NFVI.	
PF11	vNSF attestation	The Trust Monitor attests deployed vNSFs.
T_SPEC_02	The Trust Monitor should attest the vNSFs deployed on top of a host in the NFVI and provides notifications to both DARE and vNSFO.	
PF12	Log sharing	Sharing logs with a third entity SHALL be allowed. The granularity of the data provided by the logs depends on the severity and type of each attack.
T_SPEC_03	The Trust Monitor provides other components of the infrastructure with APIs to retrieve information about its status. In addition, the Trust Monitor notifies events about attestation failures to both the DARE and vNSFO, which could enrich their logs as well.	
PF16	History reports	The platform SHALL generate reports of past incidents based on historic data.
T_SPEC_04	The Trust Monitor contributes to the definition of reports of past incidents, as it will provide notifications to both the DARE and the vNSFO to enrich the logs of occurring incidents.	
PF17	Interoperability	The platform SHALL expose openly-defined APIs for information exchange with third parties.
T_SPEC_05	The Trust Monitor provides different APIs to interact with the DARE, the vNSFO and the vNSF Store. In addition, the component provides a management API to allow third parties to retrieve status information about the infrastructure's attestation.	
PF19	Network infrastructure attestation	The platform SHALL verify that the network infrastructure that executes the vNSF is in a trusted

		state (network elements and server identity, software, configuration)
T_SPEC_06	The Trust Monitor attests the software integrity of the network infrastructure and provides notifications to both DARE and vNSFO.	
NF01	Response time	The platform SHALL report the incident within a relatively short time (in the order of seconds)
T_SPEC_07	The Trust Monitor periodically attests the nodes in the NFVI (in the order of seconds) to identify any occurring incidents and report them to both DARE and vNSFO. The bottleneck for minimising the latency between two subsequent attestations is the latency introduced by the usage of TPM, as it registers the measurements in the node.	

4. CONCLUSIONS

This document presents the technical details of the vNSF ecosystem, starting with the high-level architecture and design to the specifications and implementation. The former section deals with the high-level picture of the SHIELD vNSF platform, its purpose and interconnections between components, whilst the latter presents low-level details, such as the specifications to cover, its mapping with the requirements defined in D2.1 and the decisions regarding implementation aspects.

As depicted in the document, SHIELD's vNSF ecosystem is composed by the vNSFs, Store, Orchestrator and Trust Monitor components. High-level architecture is provided per each of these components, taking into consideration the requirement specification as well as SHIELD's use cases. On the other hand, the high-level specifications, especially for the vNSFs and vNSFO, have been defined by following the recommendations and specifications of ETSI, considering it as the main standardisation body in the area. This alignment is one of the main goals of the consortium since it greatly promotes and eases the dissemination and exploitation of SHIELD's results into this standardisation body or other reference ecosystems. An example of the envisioned collaboration deals with the contribution of extensions developed within the project into some of the current standardisation bodies.

The low-level details specified for each of the scoped components result in an important asset for the next phase of the project, concerned with the implementation of such components. Specifically, the definition of the intra and inter-connectivity workflows makes it easier to agree on the responsibilities and behaviour of each component, how these will be implemented and which features will be provided by each one of them ensuring its integration at a later stage of the project. The specification of these connections took into consideration the full set of components involved in the architecture, including some from the analytics and visualisation part; whose definition is addressed in D4.1.

The details on the implementation per component indicate the intention to reuse the results of previous projects and other open-source solutions as much a possible; covering a fair amount of functionality and thus allowing to better focus on innovative aspects not yet covered by the community.

With all these aspects in mind, we conclude the first iteration of the project's design phase and we enter into the first iteration of the development phase.

REFERENCES

[1] ETSI, “Network Functions Virtualisation (NFV); Virtual Network Functions Architecture” - http://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_NFV-SWA001v010101p.pdf (Accessed May 2017)
[2] Sonata - http://sonata-nfv.eu/ (Accessed May 2017)
[3] Open Source Mano - https://osm.etsi.org/ (Accessed May 2017)
[4] The SECURED project - http://www.secured-fp7.eu/ (Accessed May 2017)
[5] Open Attestation project - https://github.com/OpenAttestation/OpenAttestation/tree/v1.7 (Accessed May 2017)
[6] Apache Cassandra 2 project - https://cassandra.apache.org/ (Accessed May 2017)
[7] Towards trusted SDN - http://ieeexplore.ieee.org/document/7116186/ (Accessed May 2017)
[8] mcTLS - http://mctls.org/ (Accessed May 2017)
[9] Automated Certificate Management Environment, https://datatracker.ietf.org/wg/acme/charter/ (Accessed May 2017)
[10] Tstat, “TCP STatistic and Analysis Tool” - http://tstat.polito.it
[11] Trusted Computing - https://trustedcomputinggroup.org/trusted-computing/ (Accessed May 2017)
[12] Charisma D3.2 - http://www.charisma5g.eu/wp-content/uploads/2015/08/CHARISMA-D3.2_v1.0.pdf (Accessed May 2017)
[13] Snort IPS - https://www.snort.org/ (Accessed May 2017)
[14] ntop nDPI - http://www.ntop.org/products/deep-packet-inspection/ndpi/ (Accessed May 2017)
[15] OpenDPI code repository - http://code.google.com/p/opendpi/ (Accessed May 2017)
[16] PF_RING - http://www.ntop.org/products/packet-capture/pf_ring/ (Accessed May 2017)
[17] Data Plane Development Kit - http://dpdk.org/ (Accessed May 2017)
[18] mcTLS code repository - https://github.com/scoky/mctls (Accessed May 2017)
[19] The netfilter.org “iptables” project - https://www.netfilter.org/projects/iptables/index.html (Accessed May 2017)
[20] Squid - http://www.squid-cache.org/ (Accessed May 2017)
[21] TeNOR - https://github.com/T-NOVA/TeNOR (Accessed May 2017)
[22] OpenBaton: https://openbaton.github.io (Accessed May 2017)
[23] Open Source MANO, release 2 - https://osm.etsi.org/wikipub/index.php/OSM_Release_TWO (Accessed May 2017)
[24] Open Source MANO, release 2 whitepaper - https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseTWO-FINAL.pdf (Accessed May 2017)
[25] OpenCIT project - https://github.com/opencit/opencit/wiki/Open-CIT-2.2-Product-Guide (Accessed May 2017)
[26] Trusted Computing Group - https://trustedcomputinggroup.org/trusted-computing/ (Accessed May 2017)
[27] Integrity Measurement Architecture - https://sourceforge.net/p/linux-ima/wiki/Home/ (Accessed May 2017)

LIST OF ACRONYMS

Acronym	Meaning
ACL	Access Control List
ACME	Automated Certificate Management Environment
API	Application Programming Interface
BSS	Business-Support System
C&C	Command and Control
CDN	Content Delivery Network
CoT	Chain of Trust
CRTM	Core Root of Trust for Measurement
DARE	Data Analysis and Remediation Engine
DoS	Denial of Service
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
EM	Element Management
EMS	EM System
FAB	Fulfilment, Assurance and Billing
FCAPS	Fault, Configuration, Accounting, Performance and Security
ETSI	European Telecommunications Standards Institute
FTP	File Transfer Protocol
GUI	Graphical User Interface

HTTP	Hyper Text Transfer Protocol
HTTPS	HTTP Secure
I2NSF	Interface to Network Security Functions
ID	Identifier
IDPS	Intrusion Detection and Prevention System
IDS	Intrusion Detection System
IMA	Integrity Measurement Architecture
IP	Internet Protocol
IPS	Intrusion Prevention System
IRC	Internet Relay Chat
ISP	Internet Service Provider
JSON	JavaScript Object Notation
LoC	Lines of Code
MAC	Media Access Control
MANO	MANagement and Orchestration
mcTLS	Multi-Context TLS
ML	Machine Learning
MSPL	Medium-level Security Policy Language
NFV	Network Function Virtualisation
NFVI	NFV Infrastructure
NS	Network Service
NSD	NS Descriptor
NSM	NS Manager

ODL	Open Day Light
ONOS	Open Network Operating System
OSI	Open Systems Interconnection
OSS	Operations Support System
PCR	Platform Configuration Registers
PF	Platform Functional
PFR	PF Requirement
PNF	Physical Network Function
PoP	Point of Presence
R&D	Research and Development
RDP	Remote Desktop Protocol
REST	REpresentational State Transfer
RO	Resource Orchestrator
SDN	Software-Defined Networking
SecaaS	Security as a Service
SMB	Server Message Block
SO	Service Orchestrator
SP	Service Provider
SPI	Stateful Packet Inspection
SQL	Structured Query Language
TC	Trusted Computing
TCG	TC Group
TLS	Transport Layer Security

TM	Trust Monitor
TPM	Trusted Platform Module
TSTAT	TCP STatistic and Analysis Tool
UC	Use Case
VCA	vNSF Configuration and Abstraction
VDU	Virtual Deployment Unit
VIM	Virtual Infrastructure Manager
VL	Virtual Link
VLD	VL Descriptor
VM	Virtual Machine
VNF	Virtual Network Function
VNFC	VNF Component
vNSF	Virtual Network Security Function
vNSFC	vNSF Component
vNSFD	vNSF Descriptor
vNSFFG	vNSF Forwarding Graph
vNSFFGD	vNSFFG Descriptor
vNSFM	vNSF Manager
vNSFO	vNSF Orchestrator
WAIS	Wide Area Information Server
XML	Extensible Markup Language

APPENDIX A. INTRA-COMPONENT INTERACTIONS

This section provides a detailed description of the internal processes carried out within the different components, along with explanations on each step of the process.

Store

vNSF Onboarding

Onboarding a vNSF (Figure 20) comprises several steps to ensure the data provided complies with the SHIELD constraints and policies. To avoid potential vNSF misbehaviour or malfunction the onboarding process encompasses an approval stage. In this stage the vNSF is registered but kept on a sandboxed state which makes it only visible to the Service Provider. Once this Service Provider deems the vNSF approved, it will be available in the Store for all the other users. Whilst the vNSF is sandboxed the Service Provider can perform any kind of validations to ensure the vNSF delivers as expected. To perform such validations a special kind of tenant may be used to provide a self-contained environment where the vNSF runs and allows the Service Provider to perform the validation in any way, shape or form, be it only the vNSF lifecycle (start/stop/etc.), any additional traffic or behaviour analysis, or operating as integrated in a NS (instantiated for the approval stage).

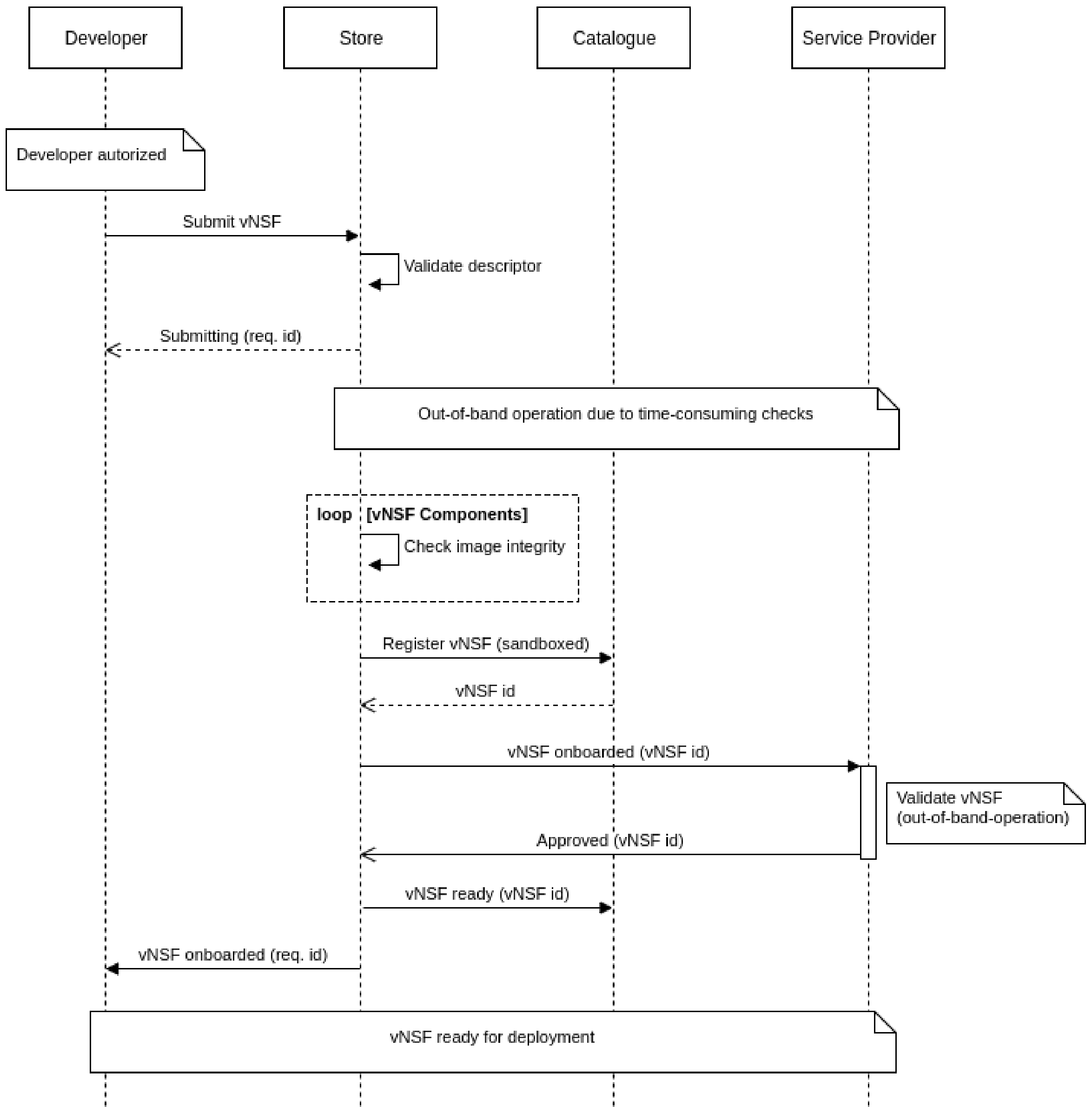


Figure 20: vNSF onboarding

NS Onboarding

The Network Service onboarding (Figure 21) is very much like the one for vNSF with the difference being the Service Provider is the one who builds a service through chaining one or more vNSFs.

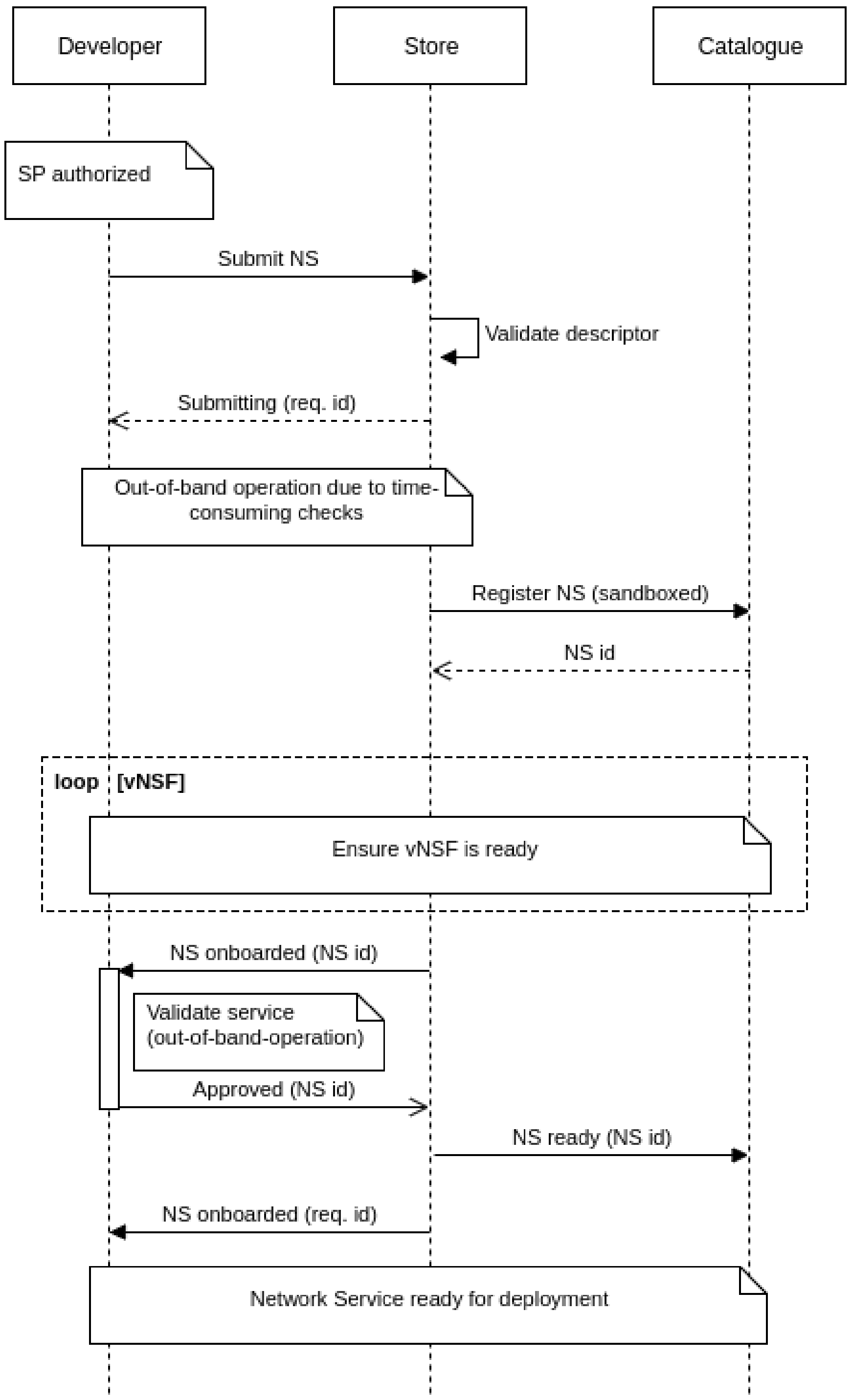


Figure 21: NS Onboarding

vNSF/NS Onboarding Failure

The onboarding may fail (Figure 22) due to errors in the descriptors, integrity checks or final approval by the Service Provider. An example of a workflow of an onboard failure of a vNSF is provided below.

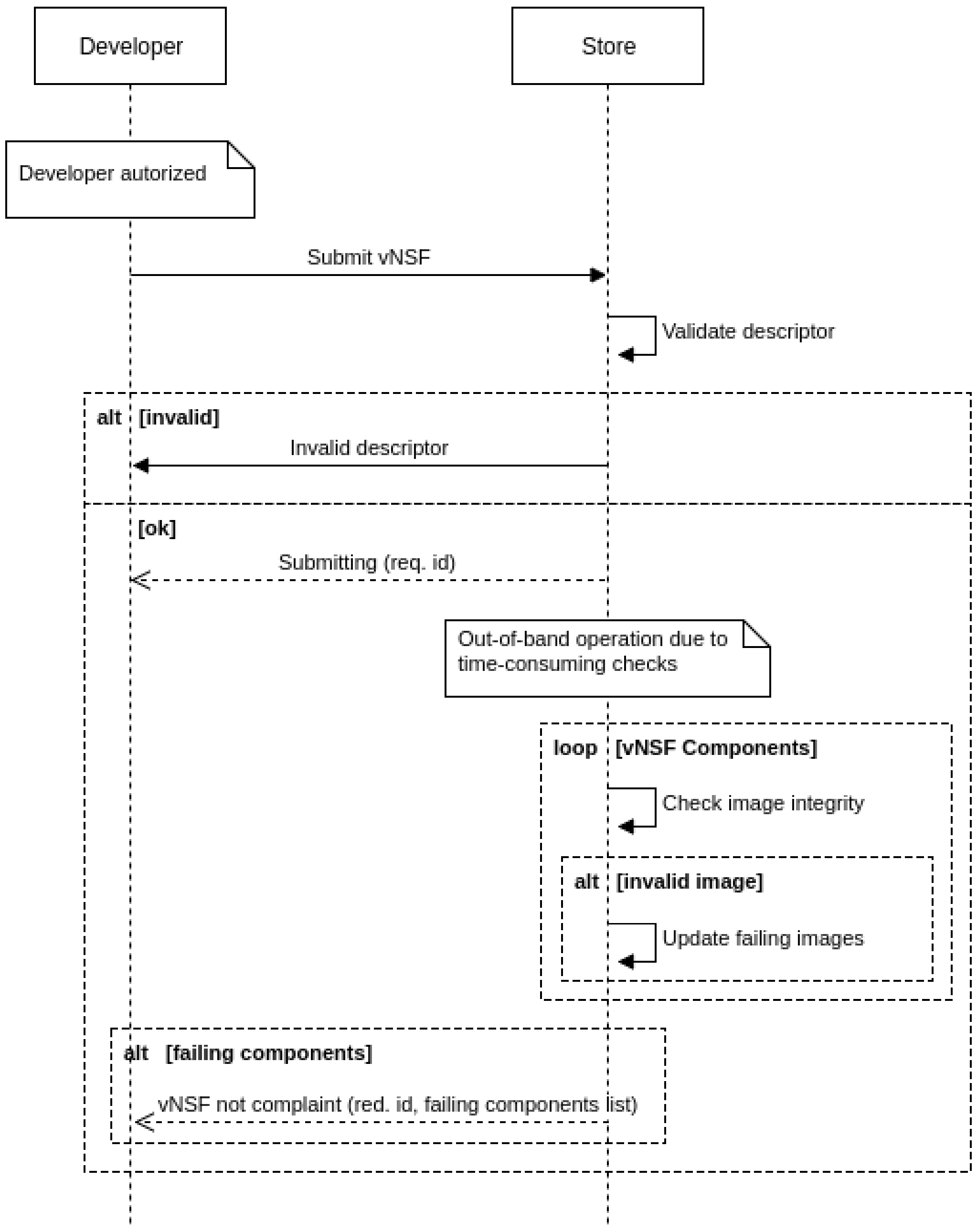


Figure 22: vNSF onboarding failure

vNSFO Orchestrator

NS instantiation

The vNSFO exerts the instantiation workflow (Figure 23) upon deployment of a given NS, which in turn deploys the constituent vNSFs and interconnect appropriately. As part of deployment, the configuration process can occur as well in order to perform pre-boot configuration on vNSFs.

- NS deployment
    1. The vNSFO retrieves the NS descriptor from the Store

2. The NS descriptor is parsed to identify the constituent vNSFs and virtual links
3. The NS Manager requests the VIM on each operation, which delegates the execution to the NFVI
4. The virtual links are defined for the vNSFs contained in the NS
5. Upon termination of the process, the resulting status is sent to the vNSFO

- vNSF deployment
  6. For each vNSF, the vNSFO retrieves the vNSF descriptor from the Store
  7. The VIM downloads the image corresponding to the specific vNSF to be deployed
  8. The request is forwarded to the NS Manager, then to the vNSF Manager
  9. The compute nodes are allocated by the VIM, and interconnected afterwards with the virtual links defined during the first stages of the NS deployment
  10. Upon termination of the process, the resulting status is sent to the vNSFO

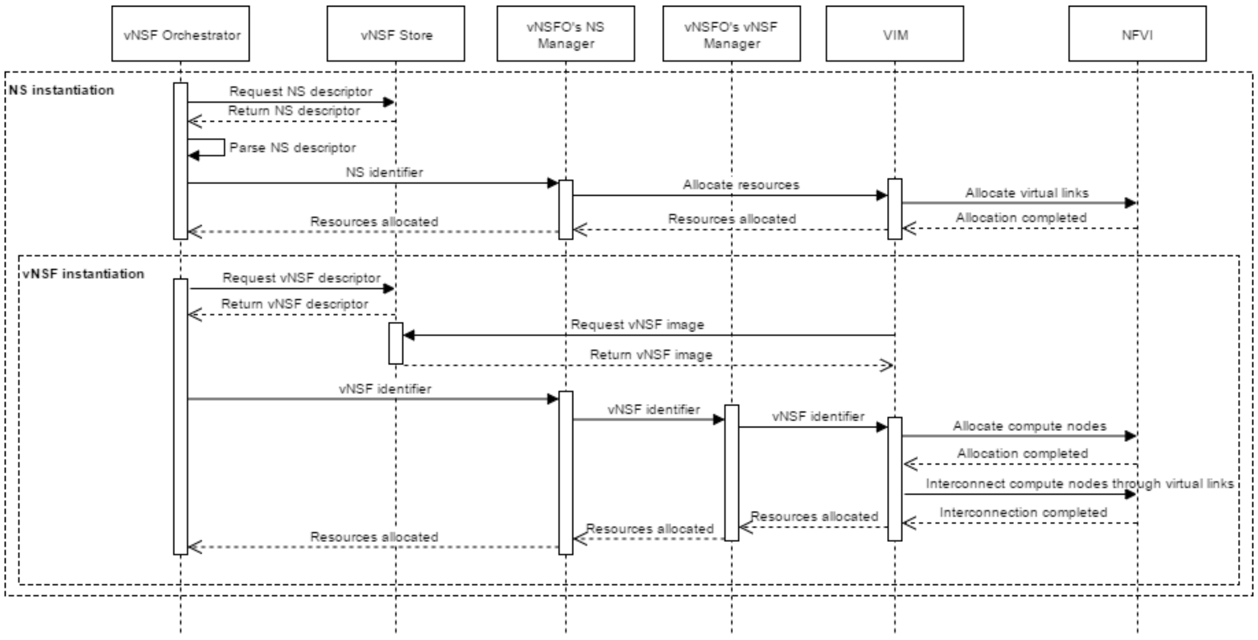


Figure 23: NS instantiation

NS configuration

The workflow is triggered when the vNSFO receives a request for configuring a deployed NS; for instance after a user selects a recommendation from the Security Dashboard, which will provide the vNSFO with policies to apply on specific vNSFs of a given NS. Then, the vNSFO calls upon the configuration on a given vNSF (Figure 24), deploying if needed the constituent vNSFs of the service and interconnecting them.

- NS configuration
  1. The request is forwarded to the NS Manager
  2. According to the configuration requested, the NS may be required to perform a change on the virtual links interconnecting the vNSFs within the service (updating, adding or deleting them) or address configurations on vNSFs only

3. The NS Manager requests the VIM on each operation, which delegates the execution to the NFVI
4. Upon termination of the process, the resulting status is sent to the vNSFO

- vNSF configuration

5. For each vNSF, the request is forwarded to the NS manager, then to the vNSF Manager
6. The vNSF Manager ensures that the provided configuration policies are valid
7. If the policies are valid; the vNSF Manager makes use of specific EMs to introduce configuration into the vNSFs. The vNSFs provide endpoints to listen for configuration changes
8. Upon termination of the process, the resulting status is sent to the vNSFO

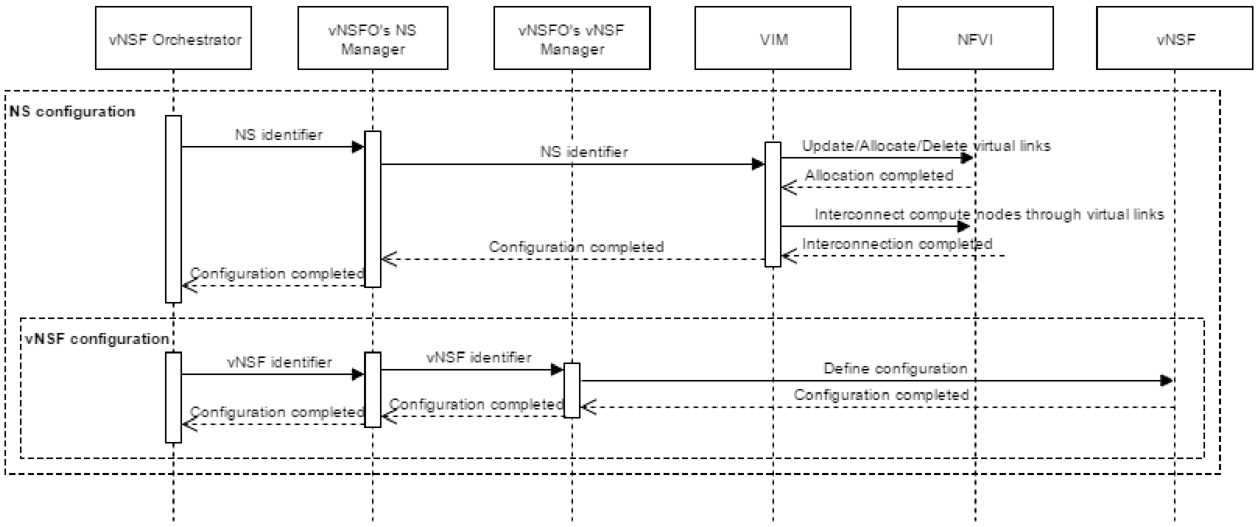


Figure 24: NS configuration

NS monitoring

The workflow (Figure 25) is triggered when the vNSFO receives a request for monitoring a running/deployed NS.

- NS monitoring

1. The request is forwarded to the NS Manager
2. Using the metrics retrieved from the constituent vNSFs, the metrics are aggregated to provide information on the status of the different monitoring values. These values are described in the NSD during its registration in the Store
3. Upon termination of the process, the resulting status is sent to the vNSFO

- vNSF monitoring

4. For each vNSF, the request is forwarded to the NS Manager, then to the vNSF Manager
5. The vNSF Manager asks the NFVI for metrics on the vNSF running instance (operation data on the compute nodes themselves) and requests the vNSFs for any metric on the processes running within them (such as load within specific services, etc)
6. Upon termination of the process, the resulting status is sent to the vNSFO

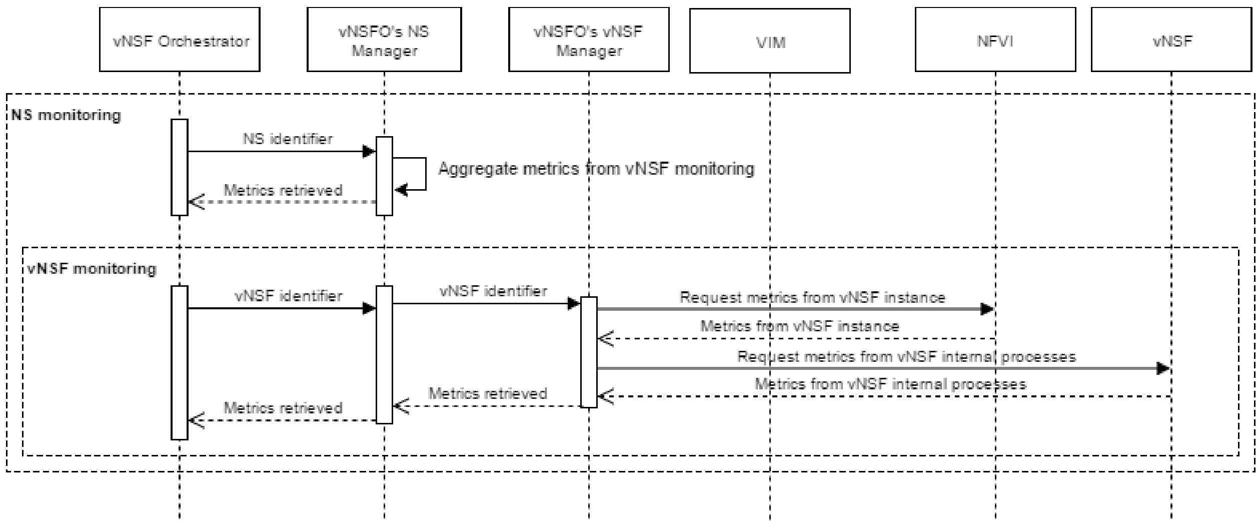


Figure 25: NS monitoring

NS scaling

The workflow (Figure 26) is triggered when the vNSFO receives a request for scaling (reduce, increase or extend resources) an existing NS.

- NS scaling
  1. The request is forwarded to the NS Manager, then to the VIM
  2. According to the operation requested, the NS may be required to update, add or delete virtual links interconnecting the vNSFs within the service
  3. The VIM interacts with the NFVI to update the definition of the links and their interconnection with the vNSFs
  4. Upon termination of the process, the resulting status is sent to the vNSFO
- vNSF scaling
  5. For each vNSF, the request is forwarded to the NS Manager, then to the vNSF Manager
  6. The vNSF Manager forwards the request to the VIM
  7. The VIM interacts with the NFVI to remove or extend the capacity of the vNSF with additional resources
  8. Upon termination of the process, the resulting status is sent to the vNSFO

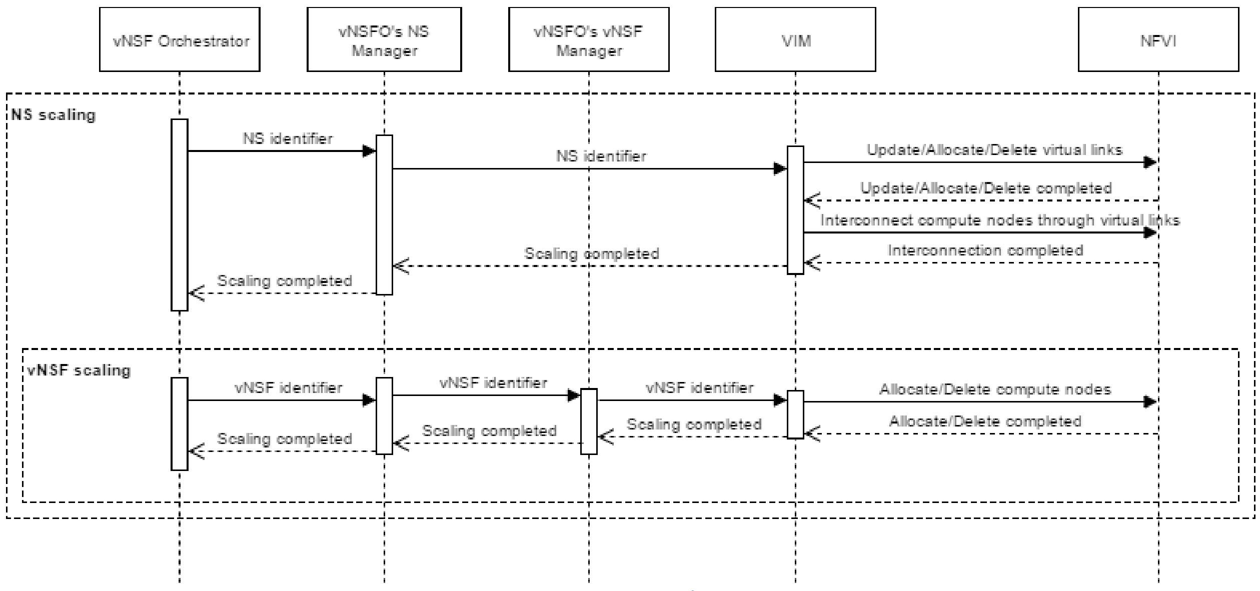


Figure 26: NS scaling

NS termination

The workflow (Figure 27) is triggered when the vNSFO receives a request for terminating a running/deployed NS.

- NS termination
  1. The request is forwarded to the NS Manager, then to the VIM
  2. The VIM interacts with the NFVI to remove the virtual links between the constituent vNSFs
  3. Upon termination of the process, the resulting status is sent to the vNSFO
- vNSF termination
  4. For each vNSF, the request is forwarded to the NS Manager, then to the vNSF Manager
  5. The vNSF Manager forwards the request to the VIM
  6. The VIM interacts with the NFVI to terminate the vNSF and release additional physical resources associated to these
  7. Upon termination of the process, the resulting status is sent to the vNSFO

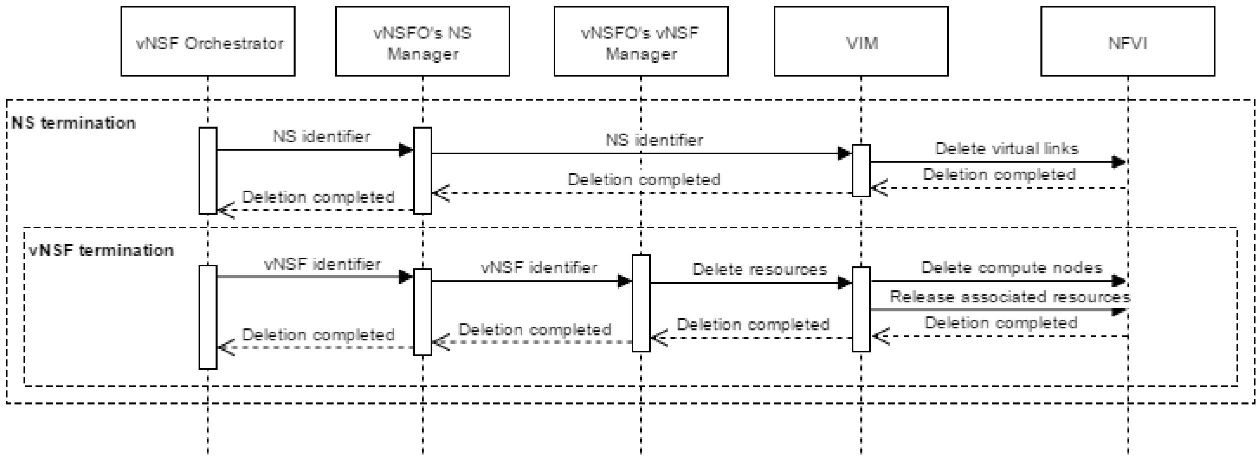


Figure 27: NS termination

APPENDIX B. INTER-COMPONENT INTERACTIONS

This section will include the description of the processes carried out between the components of the infrastructure. Each subsection will be focused on the processes initiated by a specific component.

Store

VDU Image Storage

Upon successful vNSF validation all referenced VDU images must be stored locally to allow faster instantiations. The Store provides the vNSFO with the VDU image(s) associated with the vNSF and receives a path to the image(s) storage location (Figure 28). Even though the VDU image(s) are downloaded to a Store-controlled storage location for integrity checks, these will only live in the storage controlled by the VIM. Once the images are stored by the VIM the Store do not need these anymore, so it deletes the local copy and records the final location in the Catalogue.

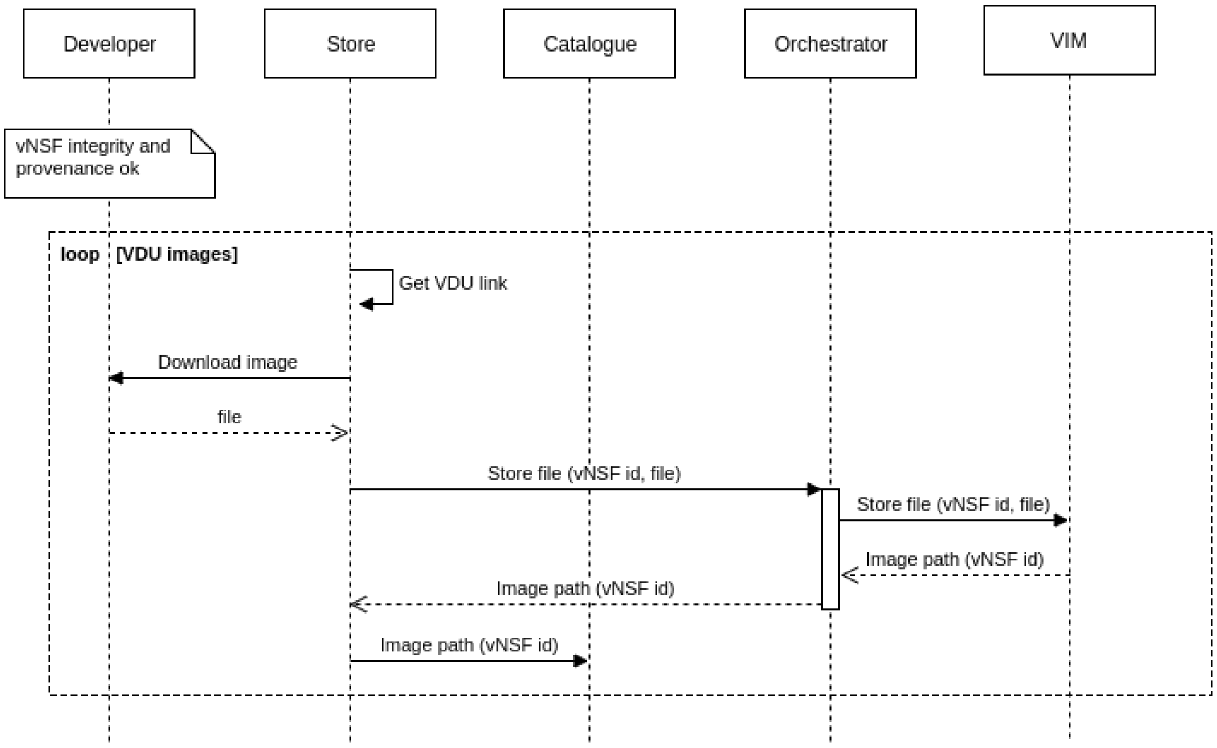
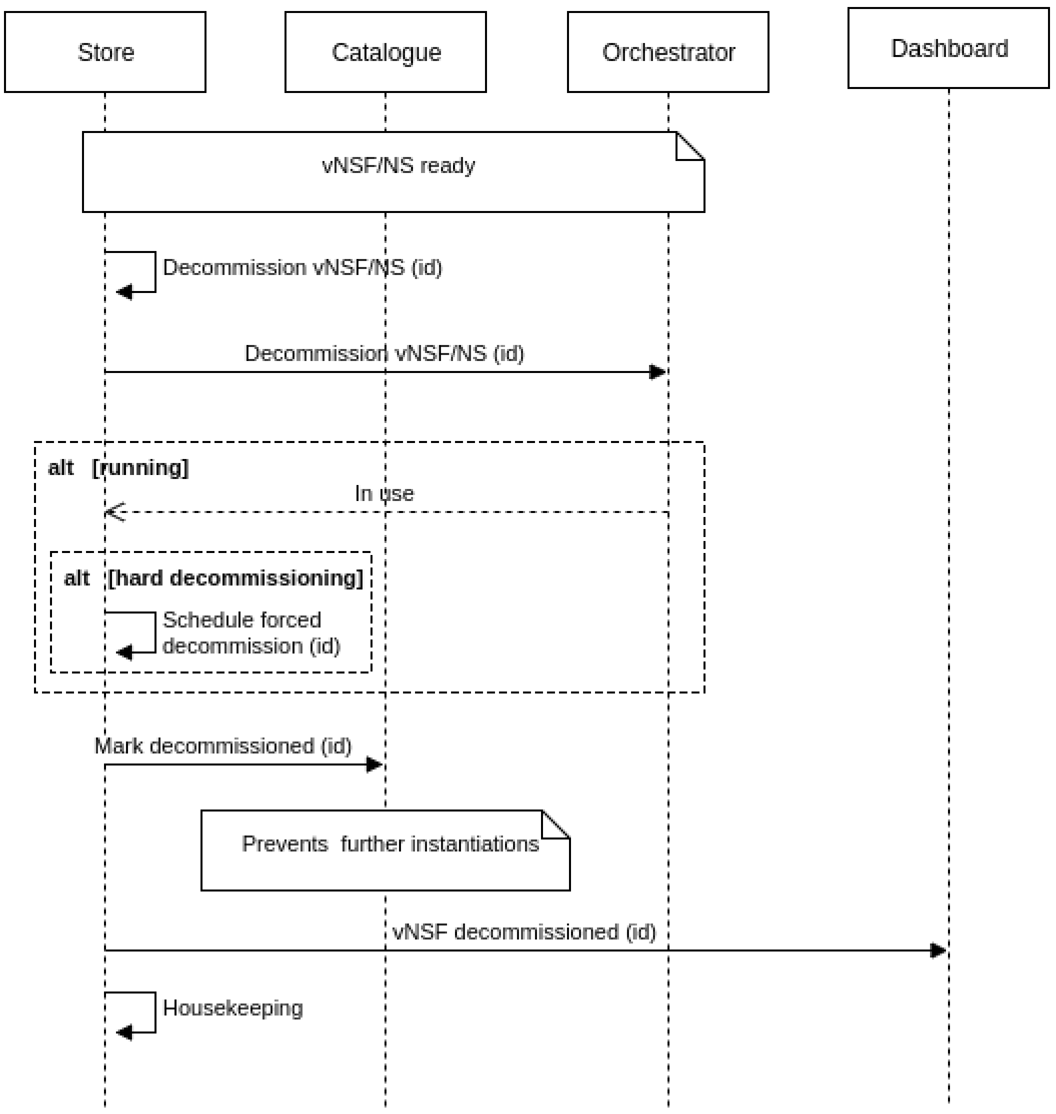


Figure 28: VDU image store

NS/vNSF Decommissioning

When a NS or vNSF reaches the end of life it must be removed from the Store. This operation (Figure 29) is triggered by the Store which marks the NS or vNSF as decommissioned to prevent further instantiations. For a running NS or vNSF a graceful decommission is provided through the schedule of the operation to a later date.


Store
Catalogue
Orchestrator
Dashboard
vNSF/NS ready
Decommission vNSF/NS (id)
Decommission vNSF/NS (id)
alt [running]
In use
alt [hard decommissioning]
Schedule forced decommission (id)
Mark decommissioned (id)
Prevents further instantiations
vNSF decommissioned (id)
Housekeeping
Figure 29: vNSF decommissioning

vNSF Orchestrator

Interaction with Store

The interaction between the Orchestrator and Store is effective during the deployment or instantiation. The vNSFO requests the NSD or vNSFD from the Store, as a first step to gather all resources for the NS instantiation, as depicted in Figure 23.

Interaction with Network infrastructure

The vNSFO talks with the NFVI on every operation defined for the vNSF and NS Managers. It accounts for two type of operations: creating, updating or removing virtual links and fetching metrics from the infrastructure. The different interactions can be observed from Figure 23 to Figure 27.

Interaction with Trust Monitor

The vNSFO will interact with the Trust Monitor at two points: first, when adding a physical node to the NFVI, so as to attest its software integrity before allowing it the access to the NFVI; and second, during the periodic attestation of the infrastructure. The process for the initial attestation is initiated by the vNSFO and is defined below, whereas the periodic attestation is depicted within the Trust Monitor section (Figure 34).

This part of the process is depicted in Figure 30, and it is described as follows:

1. The vNSFO queries the Trust Monitor to attest a newcomer and provides information about the target
2. The Trust Monitor registers the node internally if not already there
3. For each node in the NFVI, the Trust Monitor establishes a Remote Attestation process
4. Each node of the NFVI sends back its integrity report to the Trust Monitor
5. The Trust Monitor assesses each integrity report by leveraging the list of known measurements in the whitelist, as well as expected dynamic configuration such as SDN forwarding rules
6. The Trust Monitor replies to the vNSFO with the attestation result (failure or success)

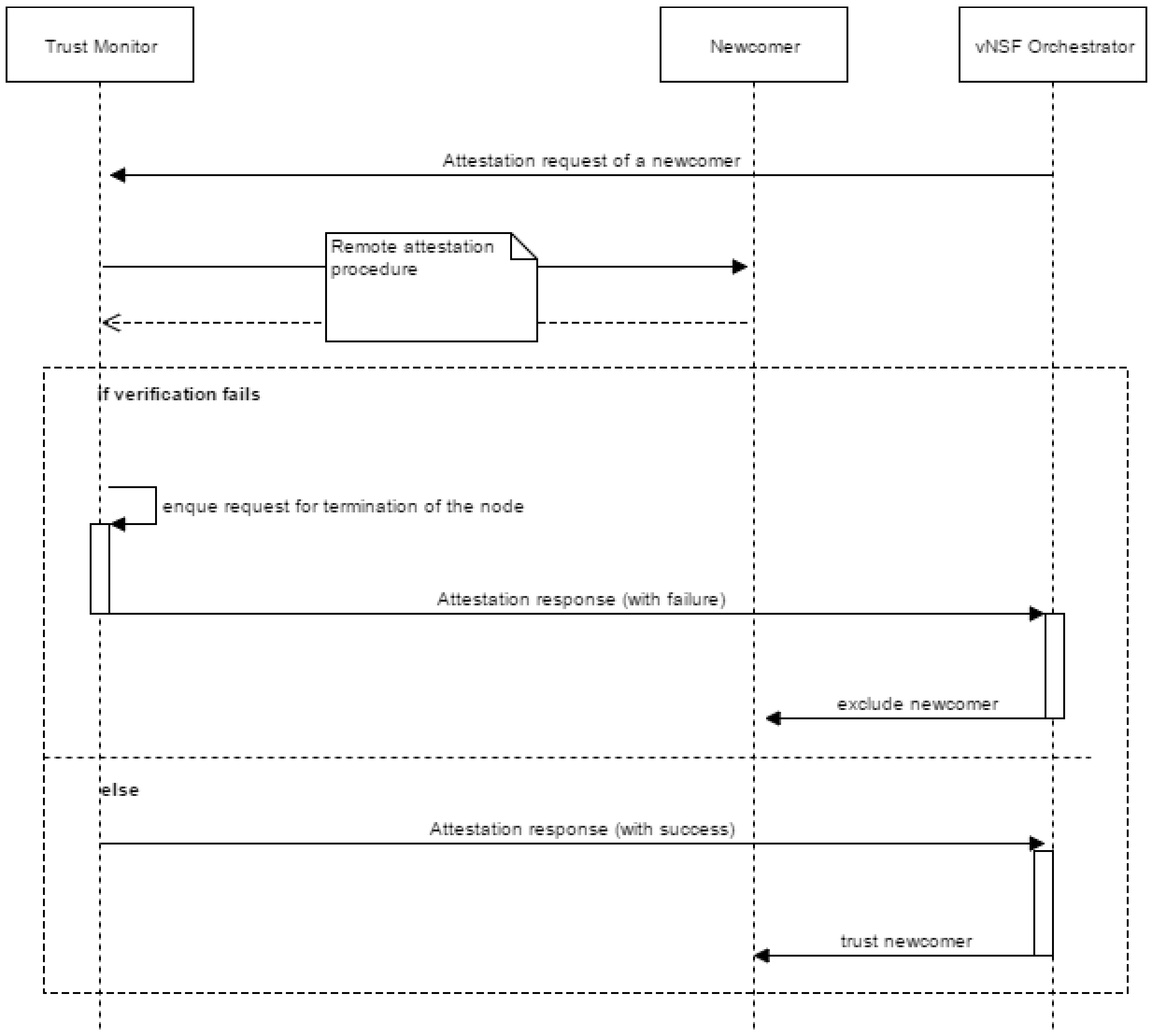

Trust Monitor
Newcomer
vNSF Orchestrator
Attestation request of a newcomer
Remote attestation procedure
if verification fails
enque request for termination of the node
Attestation response (with failure)
exclude newcomer
else
Attestation response (with success)
trust newcomer

Figure 30: Interaction between Trust Monitor and vNSFO in the initial attestation of a newcomer

Interaction with DARE

The vNSFO provides the DARE with information on the network topology, the list of vNSFs per tenant and the running NSs and vNSFs. Such information is used by the subcomponents within DARE to analyse the most appropriate deployment to mitigate an active threat (Figure 31).

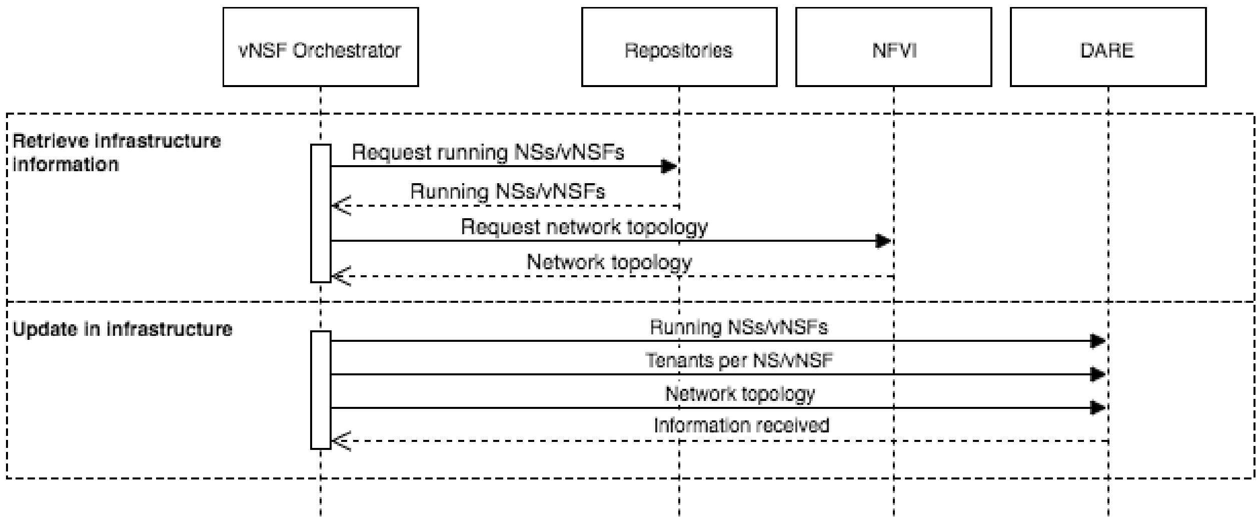


Figure 31: Interaction between the vNSFO and the DARE

Interaction with Security Dashboard

The end-user will access the Security Dashboard to obtain relevant information about the infrastructure and possible suggestions to exert in order to mitigate a given threat. The interaction between the Security Dashboard and the vNSFO occurs at this point; where the suggestions are provided to the vNSF Orchestrator as a set of NSs to deploy, as well as the policies to provide to the specific constituent vNSFs at the deployed NSs (Figure 32).

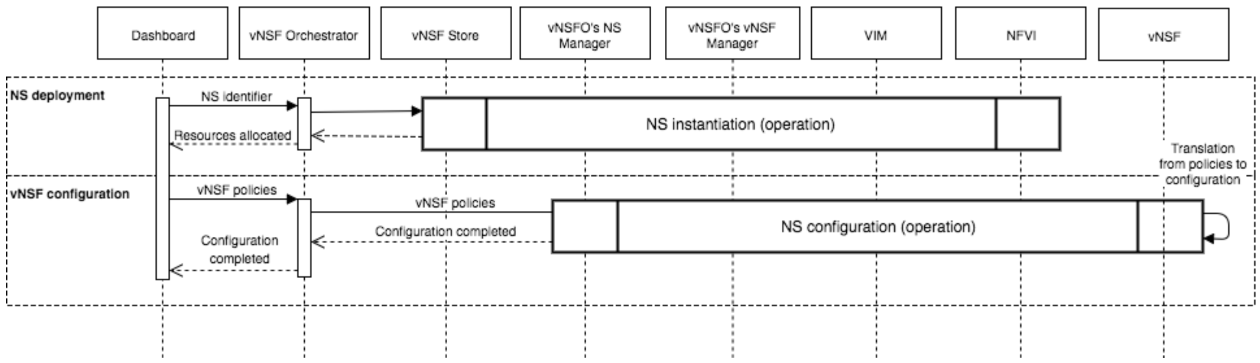


Figure 32: Interaction between the vNSFO and the Security Dashboard

Trust Monitor

Interaction with Store

The Trust Monitor interacts with the Store to retrieve attestation-specific information needed to verify the integrity of the vNSFs running in the NFVI.

The process, pictured in Figure 33, is described as follows:

1. The Trust Monitor sends a request to the vNSF Store containing a specific vNSF identifier
2. The vNSF Store sends back a response with the requested vNSF's security manifest

3. The Trust Monitor extracts the measurements (digests) of the software executed by the vNSF
4. The Trust Monitor checks if the digests are included in the Whitelist Database
5. If no matching digest is present, the TM updates the whitelist with the new measurements and links them to the correct vNSF identifier

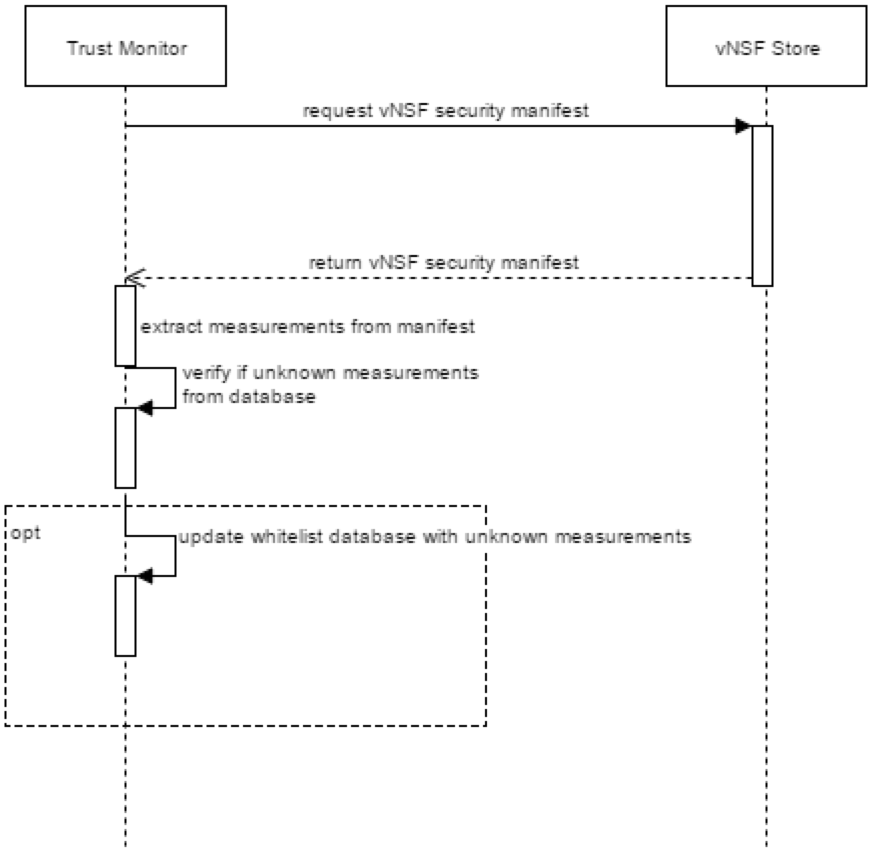

Trust Monitor
vNSF Store
request vNSF security manifest
return vNSF security manifest
extract measurements from manifest
verify if unknown measurements from database
opt
update whitelist database with unknown measurements

Figure 33: Interaction between Trust Monitor and vNSF Store

Interaction with vNSF Orchestrator

The Trust Monitor interacts with the vNSFO when performing attestation of the NFVI; either on the initial attestation of a newcomer of the NFVI or during the periodic attestation task. The former process is described in the vNSF Orchestrator section (Figure 30), whereas the process for the periodic attestation is described below (Figure 34).

1. The Trust Monitor retrieves the NFVI state from the vNSFO
2. The Trust Monitor extracts the list of nodes to be attested from the NFVI
3. For each node in the NFVI, the Trust Monitor initiates a Remote Attestation procedure
4. Each node of the NFVI sends back its integrity report to the Trust Monitor
5. The Trust Monitor assesses each integrity report by leveraging the list of known measurements in the whitelist, as well as the expected dynamic configuration such as SDN forwarding rules
6. If any of the verifications fails:
    a. The Trust Monitor sends a notification about the failure to the vNSFO
    b. The vNSFO excludes the node from the NFVI

7. In the other case, the process successfully terminates

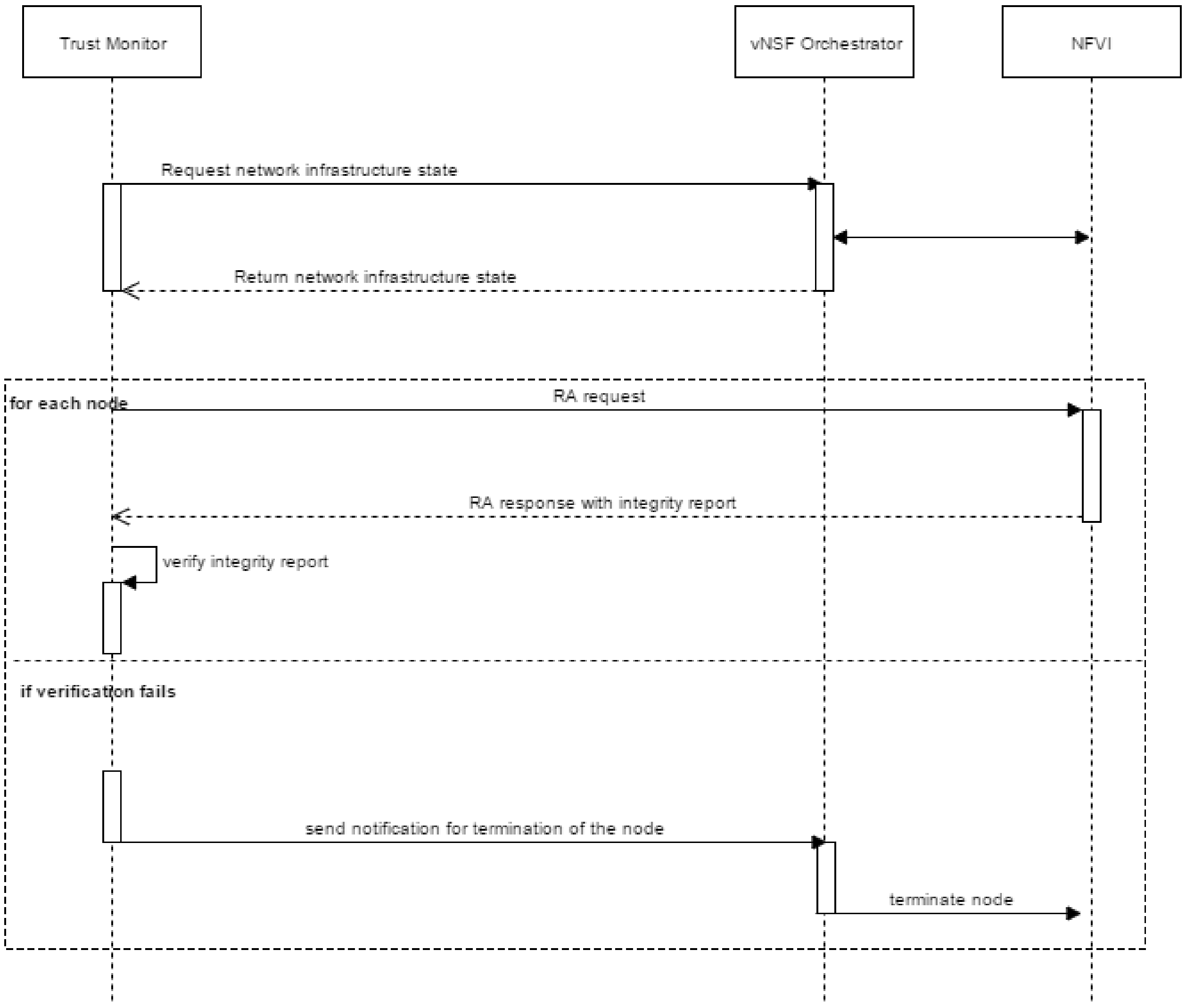

Trust Monitor
vNSF Orchestrator
NFVI
Request network infrastructure state
Return network infrastructure state
for each node
RA request
RA response with integrity report
verify integrity report
if verification fails
send notification for termination of the node
terminate node

Figure 34: Interaction between Trust Monitor and vNSF Orchestrator in the periodic attestation task

Interaction with DARE

The Trust Monitor sends by sending security event information to the DARE (i.e., a node is found to be compromised during initial or periodic attestation tasks); this can then be processed by the Big Data engine for logging and further sense extraction thanks to its security modules. The workflow is depicted in Figure 35 and goes as follows:

1. The TM detects a security event that should be logged in the DARE, such as an attestation failure of a NFVI node or vNSF (either during initial or periodic attestation)
2. The TM sends the alarm to the DARE with the detailed information about the failure.

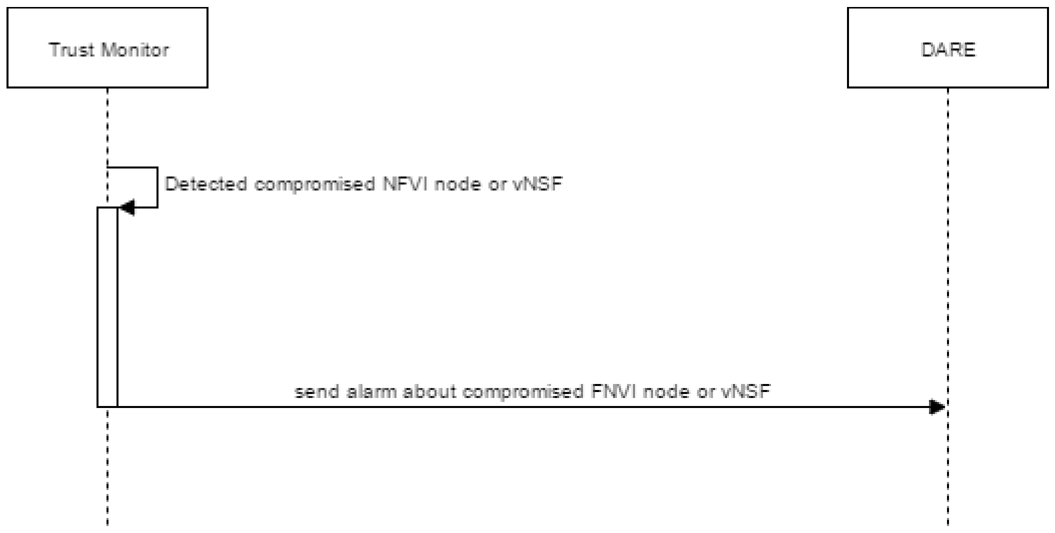

Trust Monitor
DARE
Detected compromised NFVI node or vNSF
send alarm about compromised FNVI node or vNSF

Figure 35: Interaction between Trust Monitor and DARE

APPENDIX C. TRUSTED COMPUTING TECHNOLOGIES

Trusted Computing aims at providing specific technologies and mechanisms to establish a hardware-based assessment of the integrity of a computing system. The Trusted Computing Group (TCG) [26] is the major company-backed TC consortium, which mainly focuses on the development of solutions for enabling TC in computing platform from mobile and embedded devices to data-centre class servers.

One of the fundamental principle of TC is the Chain of Trust (CoT), a transitive mechanism that ensures the trustworthiness of a computing system via a step-by-step extension process. The process requires the definition of a minimal combination of hardware and software - called Core Root of Trust for Measurement (CRTM) - that initiate the CoT measuring - and storing the measurement - the next software to be executed; it is implicitly trusted by a remote verifier. Each element of the CoT is responsible for measuring and storing the integrity of the next element, so that the whole chain can be verified by a third party. The starting point of the verification process is the CRTM, whose establishment requires a dedicated hardware security chip, called Trusted Platform Module (TPM).

The TPM is a device, standardised by the TCG, acting as a secure cryptoprocessor capable of storing keys, secrets, identities and measurements of the platform integrity. The standard has undergone different revisions, reaching the 2.0 version at the time of writing. Integrity measurements are protected by the TPM’s Platform Configuration Registers (PCR). PCRs can only be updated by the TPM itself, using an internal secure hash function, via the “extend” operation: at each step, the current value of a PCR is concatenated with the new measurement and the digest of the resulting message is stored in the PCR. This mechanism ensures that unless the platform is rebooted, no PCR-stored measurement can be erased - thus software-based attacks cannot hide execution of untrusted binaries.

The TCG also defines a specific workflow to attest the trustworthiness of TPM-equipped and measured boot enabled entities by a remote third party, called Remote Attestation. The PCRs’ value can be accessed by a remote entity by challenging the TPM with a nonce; using a hardware-protected key (i.e. only the TPM can use the private key for signing), the TPM protects the integrity of the PCRs’ with a signature which include the challenge nonce for freshness. Using the prior knowledge of all the platform’s TPM public key used for attestation, the remote entity can verify the genuineness of the signature - which also validates the hardware identity, as well as the content of the logged software events.

The TPM specification does not specify the measurement strategy to be adopted by the computing system for logged software events. The Integrity Measurement Architecture (IMA) [27] in Linux is a specific implementation that maintains a log of measured software events (e.g. the execution of a binary, using a configuration file) at runtime and, if enabled with a TPM, an aggregate integrity value is stored in one of the static PCRs. Although the log file might be manipulated by an attacker, the hardware register can’t be directly altered, meaning that a verifier could detect any unexpected tampering to the log file.

APPENDIX D. APPLICATION PROGRAMMING INTERFACES (APIS)

This appendix presents a first definition of the methods (and arguments) to be supported by the APIs exposed by each component.

Orchestrator

This section includes the low-level specifications of the operations offered by each API exposed by the vNSFO.

Dashboard API

The vNSFO will provide an interface so that the Security Dashboard can retrieve the necessary information to provide the visualisation to the end-user.

Operation	Arguments	Description
get_network_topology	-	Provides the topology of the network as provided by the VIM
get_deployed_vnsfs	-	Provides the running vNSFs
get_deployed_vnsfs	tenant_id	Provides the running vNSFs, filtered by tenant

Trust Monitor API

The vNSFO will provide an interface so that the Trust Monitor can obtain the information to perform the periodic attestation task.

Operation	Arguments	Description
get_physical_nodes	-	Provides the list of active physical nodes in the NFVI
get_deployed_vnsfs	-	Provides the running vNSFs
get_network_topology	-	Provides the topology of the network as provided by the VIM

get_network_flowtable	-	Provides the contents of the flow tables of the SDN controller

DARE API

The vNSFO will provide an interface for the DARE to obtain a global view on the NFVI and thus be able to perform the analytics and provide the recommendations.

Operation	Arguments	Description
get_network_topology	-	Provides the topology of the network as provided by the VIM
get_deployed_vnsfs	tenant_id	Provides the running vNSFs, filtered by tenant
get_deployed_vnsfs	-	Provides the running vNSFs
get_deployed_nss	-	Provides the running NSs

Trust Monitor

This section includes the low-level specifications of the operations offered by each API exposed by the Trust Monitor.

Management API

The Trust Monitor will provide a Management API with operations that would allow other components to check the status of the infrastructure's attestation.

Operation	Arguments	Description
get_status_info		Retrieves status information about the Trust Monitor
get_vnsf_attestation_info	node_id	Retrieves attestation-specific information for a single vNSF
get_nfvi_attestation_info		Retrieves attestation-specific information for the whole NFVI

get_nfvi_pop_attestation_info	node_id	Retrieves attestation-specific information for a specific NFVI PoP

Newcomer Attestation API

The Trust Monitor will provide an interface for receiving attestation requests for a newcomer in the NFVI. Note that the interface should be specific for the newcomer's attestation, as the Trust Monitor will later on perform periodic attestation tasks over the different nodes that have been pre-registered to it. The API may be used for both physical nodes, during the initial authentication phase, or for vNSFs, during their instantiation phase.

Operation	Arguments	Description
register_node	node_id, address, distribution, ...	Registers the node to the Verifier, given a unique identifier (which will be used for further attestation procedure), the address of the node, the distribution of the OS running in it
attest_node	node_id, analysis_type	Remote Attestation request to the node, identified by a unique ID. The client to be attested will provide the integrity measurements according to the type of requested analysis (e.g. load-time analysis with a certain trust level for the measurements)

Store

The Store will provide an interface to obtain a the information it persists as well as accessing features it provides.

Operation	Arguments	Description
onboard_vnsf	security_manifest, vnsf_descriptor	Onboards a vNSF
onboard_ns	security_manifest, ns_descriptor	Onboards a NS
get_vnsf_onboarding_status	id	Provides the status for the vNSF onboarding operation

get_ns_onboarding_status	id	Provides the status for the NS onboarding operation
list_vnsfs	-	Provides a list of all the onboarded vNSFs along with a brief description for each one
list_nss	-	Provides a list of all the onboarded NSs along with a brief description for each one
get_vnsf_info	id	Provides all the information on the onboarded vNSF
get_ns_info	id	Provides all the information on the onboarded NS
decommission_vnsf	id	Retire a vNSF
decommission_ns	id	Retire a NS
get_vnsf_security_info	id	Provides all the security information concerning a vNSF
get_ns_security_info	id	Provides all the security information concerning a NS

APPENDIX E. TECHNOLOGY SELECTION

Orchestrator

The analysis of different vNSFOs has been carried out to choose which one to use in SHIELD. To do this we selected a subset of some well-known open-source NFV MANO (TeNOR, OSM, SONATA, OpenBaton) and considered the adequateness depending on the mapping to the SHIELD's Platform Functional Requirements, the support of some relevant key features within the project and the status of its community and development.

Platform Functional mapping

PF Requirement	TeNOR	OSM	SONATA	OpenBaton
PF01 - vNSF and Network Service (NS) deployment	Y	Y	Y- (No external cloud deployment)	Y
PF02 - vNSF lifecycle management (on boarding, instantiation, chaining, configuration, monitoring and termination)	Y	Y- (Monitoring based on VIM implementation integrated in R3)	Y	Y
PF03 - vNSF status management (DEPLOY, START, STOP, MODIFY, DELETE)	Y	Y	Y- (Ongoing for: adding restart, stop, pause)	Y- (Some may be missing)
PF04 - Security data monitoring and analytics	Y- (Delegated to NSM and vNSFM)	Y- (Delegated to the EM)	Y- (Custom metrics allowed, infra metrics)	Y-
PF05 - Analytics visualisation	N/A	N/A	N/A	N/A

			(son-gui shows monitoring metrics)	
PF06 - Ability to offer different management roles to several users (multi-user with possibility of configuring different roles)	N	N	Y (Static dev/customer roles; new roles will be customised and dynamic)	N
PF07 - Service elasticity [optional]	Y	Y (Experimental NS scaling. manual GUI, support for adding/removing full VNFs to/from a running NS)	Y- (Will allow scale-out)	Y
PF08 - Platform expandability	Y	Y	Y	Y
PF09 - Access control	Y (Tokens)	Y (Certificates)	Y (User/sw, sw-sw using tokens)	Y
PF10 - vNSF validation	N/A	N/A	N (Signed packages in store, control mangling)	N/A
PF11 - vNSF attestation	N	N	N	N
PF12 - Log sharing	N/A	N/A	N/A	N/A
PF13 - Mitigation	Y	Y	N	Y
PF14 - Multi-tenancy	Y	Y-	N	N

	(1 tenant : 1 running NS)			(1 tenant for all)
PF15 - Service store	N/A	N/A	Y	N/A
PF16 - History reports	N/A	N/A	N (alerts aggregated)	N/A
PF17 - Interoperability	N/A	Y (Multiple VIMs)	N	N/A
PF17 - Interoperability	Y	Y	Y	Y
PF19 - Network infrastructure attestation	N/A	N/A	N	N/A
PF20 - Billing framework	N (Delegated to BSS)	N (Delegated to BSS)	N/A (License concept in NS related to billing)	N

Feature-focused analysis

Feature	TeNOR	OSM	SONATA	OpenBaton
Type of virtualisation	VMs	VMS (Containers may be possible)	VMs	VMs
VIM supported	OpenStack	OpenVIM (R1/R2), OpenStack (R2), VMWare (R2)	OpenStack	OpenStack
SDN controller supported	ODL (through netfloc)	ODL (R1/R2), Floodlight (R1/R2), ONOS (R2)	ODL	ODL, ONOS (ongoing)

Service Function Chaining	Y (Using netfloc plug-in and ODL)	Y (Direct, no plug-in)	Y (ODL SFC)	Y
Lifecycle management	Y (Start, stop)	Y (Available for VNF & NS)	Y	Y
Event management	Y (Custom)	Y (Provide messages about the deploy of vNSF & NS)	Y- (ongoing)	Y (Generic and specific)
Elasticity	Y (Scale-in, scale-out)	Y (Scale-in, scale-out; experimental support to modify running NSs)	Y- (Will allow scale-out)	Y (Auto-scaling)
Monitoring	Y (Per NS, per vNSF instance and inner service)	Y (Based on VIM, delegated to EM)	Y (Prometheus, log aggregation per component)	Y (Zabbix for NFVI and VNFs)
Dynamic vNSF placement	Y- (Algorithms in place, not tested)	N	N (Ongoing design for auto-location and distributed NSs)	Y

Maintenance-focused analysis

Key	TeNOR	OSM	SONATA	OpenBaton
LoC	441217	340861	6596	118322
Development language	Ruby	Python	Ruby, Python	Java

Community	i2CAT	ETSI and 60 orgs (8 net operators)	ATOS, i2CAT, etc	Fraunhofer/FOKUS, TUB
Projects in use	EU and national R&D ongoing projects	EU R&D projects, Telefónica VNF cert Lab, RIFT.ware	EU R&D projects	EU R&D project, 5GBerlin testbed