SHIELD

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

[H2020 - Grant Agreement No. 700199]

Deliverable D3.3

vNSF framework ready for experiments

Editor C. Fernandez (i2CAT)

Contributors G. Gardikis (SPH), C. Xilouris, E. O. Segou (ORION), C. Fernandez, E. Trouva (NCSRD), L. Jacquin, H. Attak (HPELB), M. De Benedictis, A. Lioy (POLITO), F. Ferreira, R. Preto (Ubiwhere), A. A. Pastor, J. N. Mendoza (TID).

Version 1.0

Date November 29th, 2018

Distribution PUBLIC (PU)










Executive Summary

This report documents the conclusions of the project activities related to the definition of the requirements, the high-level architectural design of the SHIELD platform, as well as its development and deployment.

Specifically, this deliverable provides details on the current status of the architecture, design, development and deployment related to the vNSF environment; that is, the attestation of the infrastructure via the Trust Monitor (TM), the Store and Orchestrator (vNSFO) interacting with each Network Service (NS) and Virtual Network Secure Function (vNSF), and the list of NSs themselves.
The final mapping of the requirements (Platform Functional, Non-Functional, Service Functional, Ethical & Regulatory Compliance) is also collected on this report.
Finally, instructions for the set-up of the environment and of each WP3-based component are also supplied.

Table of Contents

1. INTRODUCTION

1.1. SHIELD project overview

The SHIELD project aims to deliver virtualised security services to protect the flow of data across the infrastructure controlled by the SHIELD platform and also to ensure that such infrastructures are properly attested and secured.

To do so, several technologies are combined. The Network Function Virtualisation (NFV) and Software-Defined Networking (SDN) environments allow the deployment of virtualised services (NSs); whilst the Trusted Computing (TC) provides means to attest and verify whether the infrastructure is secured or trusted. Finally, the Big Data (BD) Analytics and Trusted Computing (TC) provide remediation suggestions to mitigate threats in any virtualised or physical node running on the SHIELD-managed infrastructure.

1.2. Scope of this document

The WP3 ("vNSFs ecosystem") is devoted to the realisation of the environment required to deploy the different NSs and vNSFs. That covers part of the infrastructure in the SHIELD architecture, the development of the SHIELD NSs and vNSFs themselves and the management of the lifecycle of the instances and of the packages through the vNSF Orchestrator and Store, respectively. The attestation of the infrastructure and services, to ensure all of these are trusted at all times, is also encompassed in this Work Package and thus described here.

This document (D3.3, "vNSF framework ready for experiments") provides an overview of the final release of the vNSF and trusted computing infrastructure. During M20-M27, SHIELD has continued the deployment, adjustment and configuration efforts on its NFVI and has finalised the definition, elicitation of requirements, development and testing of the suggested Network Services. Likewise, during these months there was substantial attainment on most of the features to be offered; affecting the vNSF Store, the Orchestrator and the Trust Monitor.

D3.3 extends the following deliverables:

- D2.2 "Updated requirements, KPIs, design and architecture", which gives the final, updated version of D2.1.
- D3.2 "Updated specifications, design and architecture for the vNSF ecosystem", which contains the design and specifications for the SHIELD vNSFs, Orchestrator, Store and Trust monitor provided in D3.1; after the second iteration of the requirements evaluation.

1.3. Organisation of this document

This document is organised as follows:

- Chapter 1 (present chapter) serves as a basic introduction to this document and its scope;
- Chapter 2 provides the overview of the final functional layout of the vNSF ecosystem;

- Chapter 3 details the updates at the different levels (architecture and design, development) per WP3 component since D3.2;
- Chapter 4 lists the instructions to perform the setup of the environment tools that support the operations of the vNSF ecosystem;
- Chapter 5 includes the guide for the installation of the WP3-related components;
- Chapter 6 concludes the document, summarising the WP3 work during the project and suggesting next steps for improvements;
- Annex A provides the Ethical Regulatory Compliance requirements for the WP3 components.

2. FINAL FUNCTION LAYOUT

2.1. vNSF environment architecture

The group of components and tools required for the lifecycle management of the SHIELD NSs, vNSFs and attestation on them and on infrastructure nodes -- grouped under the term of "vNSF environment" and covered by the work of WP3. This comprehends the Trust Monitor, the vNSF Store, the vNSF Orchestrator and all the NSs and vNSFs used in the project.

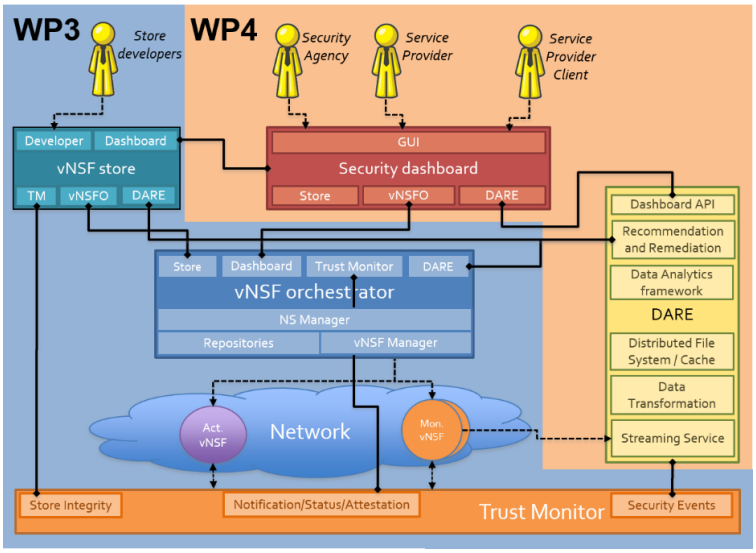

Figure 1: High-level architecture of SHIELD, with components per WP.

The Trust Monitor maintains an overview of all nodes available in the infrastructure: either virtual or physical, whether network devices or computing nodes. With this data, the TM can periodically check the trustworthiness of the SHIELD infrastructure by leveraging Trusted Computing (TC) mechanisms. This process consists of authentication and integrity verification techniques applied to each node joining the infrastructure. The TM interacts with almost all other WP3 components in order to get metadata from the vNSFs' security manifest (from the vNSF Store), to register nodes for attestation and to perform periodic attestation of nodes running on the NFVI itself (vNSF Orchestrator); as well as contacting the NFVI nodes themselves for attestation. Besides this, it also interacts with the DARE, enabling auditing capability and the Dashboard, for security notifications and recommendations for remediation.

The vNSF Store allows the onboarding of SHIELD vNSF and NS packages to some specific type of users (namely, developers and administrators). Such packages are onboarded in the NFVO (via the vNSFO API) and its metadata is stored internally; for instance, for attestation purposes (the security manifest used by the Trust Monitor). It also interacts with the Dashboard.

The vNSF Orchestrator provides a logic layer on top of the NFV Orchestrator (NFVO), so that the interaction with the NFVO, the VIMs and elements on the NFVI (from ancillary nodes to running vNSFs) is simpler and low-level details are hidden to other SHIELD components. Contacting this component results in the fetching of information from the VIM and some nodes

in the NFVI; the deployment, termination and configuration for NSs and vNSFs; as well as registration, isolation, termination and such for nodes registered by the NFVI.

Then, the NSs and vNSFs are dynamically deployed on the network infrastructure and perform two main type of actions: monitoring (identifying specific information from the network) and reacting (ability to carry out specific remediation action).

The network infrastructure and NFVI nodes do interact with the Trust Monitor so that the virtual services inside and its subcomponents can be authenticated; as well as with the vNSFO so that the latter can control the deployment, lifecycle management and the collection of monitoring data. Finally, the monitoring NSs send captured data to the DARE, which is valuable for further ingestion and determination of specific remediation actions.

3. UPDATES SINCE D3.2

This section iterates through each WP3 component to document any update on its design, architecture or development that has occurred during the last period here reported.

3.1. Trust Monitor

The Trust Monitor architecture has not been altered since D3.2, hence its definition and the definition of its subcomponents are left mostly intact.

The scope of the "vNSFO Connector" has been limited to the retrieval of network configuration at a given time for attestation of the NFVI. Originally, the Trust Monitor was supposed to notify the vNSFO about the need to terminate a compromised vNSF or NFVI physical node (e.g. compute node, switch) as well. This functionality is still provided in the SHIELD attestation framework, although the notification is forwarded to the vNSFO through the Dashboard.

Also, the "Management API" has been updated to include an endpoint for auditing of historical logs regarding attestation of the network infrastructure.

3.2. vNSF Store

The core architecture of the vNSF Store has not been modified, yet new features were introduced during the implementation of several sub-components.

The vNSF and NS lifecycle management was improved by the integration with the vNSFO, Trust Monitor and Dashboard components. The integration of the Store with the vNSFO supports all the onboarding capabilities of vNSFs and NSs. For the integration of the Trust Monitor, the Store persists the attestation data retrieved from the security manifest to later on provide it to the TM. The multi-user support was also enforced; thus restricting users -previously registered in the Dashboard- to particular particular actions associated with their role, for instance administrator to onboard NSs, developer to onboard vNSFs.

Driven to ensure, as much as possible, the viability of onboarded descriptors, the Store is now integrated with a validation tool called NSFVal which is capable of analysing the syntax, the integrity and the network topology of descriptors. The validation is triggered during the Store onboarding process, just before passing the descriptors to the vNSFO. Should any validation issue arise during this process, the user is alerted and enquired about the following actions.

3.3. vNSF Orchestrator

On the design and architecture of the vNSF Orchestrator, the definition of its subcomponents has not suffered modifications since D3.2; since the functional blocks covering these remain available over the different releases of the NFVO, even yet mapped to different software subcomponents.

Regarding the SHIELD-specific interfaces (APIs used by the Store, Dashboard and Trust Monitor) and the overall logic (to onboard packages, configure vNSFs and interact with the

infrastructure), these remained mostly intact; with the exception of the "DARE API", whose logic is available but is not called from the DARE -- but from other components.

Summing up, the following interfaces (SHIELD-specific) are provided by the vNSFO:

- Store API: Provides the vNSF/NS Store with an endpoint to register the onboarded vNSF and NS packages into the NFVO instance; so the service can be instantiated.
- Dashboard API: The application of a recommendation through the Dashboard will contact the vNSFO, and indirectly the NFVO, to communicate with a specific vNSF and transmit the MSPL to that running instance.
- Dashboard Connector: Data regarding the list of running NS and vNSF instances are provided for visualisation purposes, as well as data on the NFVI (e.g., related to the trusted status of the different nodes in the infrastructure).
- Trust Monitor Connector: The newcomer nodes (ingressing the NFVI) can be registered in the Trust Monitor; so that the TM can periodically perform the attestation.
- Trust Monitor API: Information of the network, the flow tables and the list of active nodes can be provided to the Trust Monitor for its own purposes.

3.4. NSs and vNSFs

Modifications on specific NSs and vNSFs since D3.2 are described below, where each NS is described in a separate subsection.

3.4.1. DPI

The ORION DPI's NS (also called vDPI) presents minor changes with respect to the status documented in D3.2. The main functionality remains the Deep Packet Inspection (DPI). As discussed in D3.2, 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 multiple cyber-attacks 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).

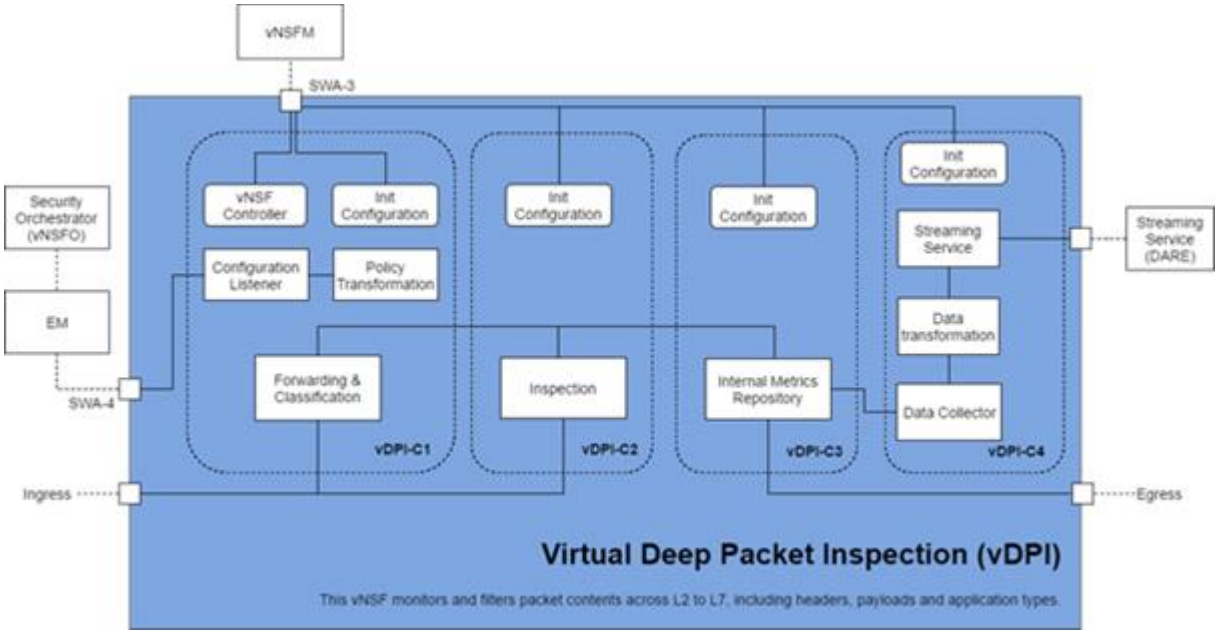

Figure 2: DPI architecture.

The DPI vNSF consists of vNSF components (VNFCs), as illustrated in Figure 2:

- **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 DPI NS 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 illustrates the classified traffic and GDPR information. The distributed collectors send data to the DARE.

The key advances since D3.2 include:

- The vNSF graphical user interface (developed to suit the SHIELD demonstration scenarios and GDPR information)
- The integration of the distributed collectors to stream data to the DARE (vDPI-C3).

As with any other NS, the DPI lifecycle is managed by the vNSFO -- specifically by interacting with the vNSF Configuration Manager subcomponent. The vNSFO is in charge of starting, stopping, pausing, scaling and configuring the DPI. 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. Policies are relayed from the vNSFO and translated within the managing vNFC. No significant changes have been made to the lifecycle management since D3.2.

3.4.2. Forward L7 Filter

The L7 filter vNSF presents no substantial changes from the D3.2. This NS provides the following functionalities:

- 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);
- Reverse Proxy.

3.4.3. HTTPS Analyzer

Since D3.2 the component architecture has undergone no subsequent modifications. In this period, the engine has been developed and the interfaces to the vNSFO's, DARE's and vNSFM's API have been completed.

The objective is to provide a category classification of HTTP and HTTPS traffic without analyzing the payload content in a privacy-friendly way. Categories are dependent on the trained model. Currently 3 categories are included: Browsing, video streaming and file download. This NS filters out unnecessary traffic from security inspection. A second functionality of this NS is to provide a netflow format feed adapted to Apache Spot streaming service from traffic captured.

The vNSF integrates both raw and processed traffic in Apache Spot cluster. This service is composed by two different parts:

- Raw traffic feed: It captures the traffic arriving to the vNSF, converts it to netflow_v9 format and sends it to the Apache Spot cluster via a Kafka bus using the d-collector developed in SHIELD.
  Netflow traffic is the base for subsequent tasks in Apache Spot: worker module that collect, transform and store the information, and the Apache Spot Machine Learning module.
- Processed traffic feed: It captures the traffic arriving to the vNSF, converts it to "tstat" format and processes it in real time. It is a passive sniffer able to provide several insights on the traffic patterns at both the network and the transport levels. It processes the output "tstat" data and classify them via a local machine learning algorithm pre-trained. It allows to apply ML locally in a vNSF. The machine learning result is sent in CSV format to the Apache Spot via a Kafka bus (very similar to the netflow format, but with a classification label).

3.4.4. IDS

Since D3.2, the IDS NS has been altered to support notification of alerts to the SHIELD Dashboard. For this reason, the Event Publisher Service (which translates, curates and publishes Snort events in readable format) has been modified. When new Snort alerts are triggered, these are transformed to an agreed JSON format and sent to a RabbitMQ queue.

The format of the messages sent to the RabbitMQ message broker is the following:

```
[
  {
      "event": {
      "event-id": 1,
      "signature-id": 10000002,
      "protocol": 6,
      "event-second": 1458746598,
      "event-microsecond": 778441,
      "classification-id": 24,
      "classification": "Detection of Malicious traffic",
      "destination-ip": "192.168.1.237",
      "dport-icode": 80,
      "source-ip": "23.144.220.60",
      "sport-itype": 34276
      }
  }
]
```

3.4.5. L23 Filter

The L23 Filter NS is based on a VNF that has been developed in the frame of CHARISMA EU project and has been adapted to be deployed through the NFVO used in SHIELD. Such vNSF has to be deployed in-line on an existing network link, deciding which packets will pass through its two network interfaces. It can be instantiated with specific rules that allow only the preferred traffic to be propagated. Firewall rules can be applied to the vNSF after instantiation via its RESTful web API, providing dynamic security policies to be enforced only by higher level modules of the project (e.g DARE components). The implementation of the L23 Filter NS is based on Open vSwitch (OVS). Firewall rules; which can be used to filter the traffic passing through the vNSF. The Firewall Rule Receiver Web API receives firewall rules in JSON format by an HTTP request and translates it to the appropriate OVS flow table entry. Furthermore, for purposes of monitoring and security validation, the L23 Filter vNSF provides another service whose role is to publish information about passing or blocking of packets to external interfaces for further analysis.

The L23 Filter vNSF consists of one virtual machine that requires at least two virtual network interfaces. It uses three interfaces:

- one interface for management;
- one interface for the WAN side of the firewall;
- one interface for the LAN side of the firewall.

3.4.6. L3 Filter

The L3 Filter NS has the aim to inspect and eventually filter the passing through traffic flows. The inspection process is related to the Layer 3 ISO/OSI stack information: source and destination IP addresses, ports and transport protocol. When a traffic flow matches certain Layer 3 conditions, it will run a predefined action (allow or deny). All matching rules and related actions are described through an Access Control List (ACL), which could be in form of whitelist or blacklist. In our case we obtain the ACL through a process of translation from the high-level configurations provided by the Element Management (EM).

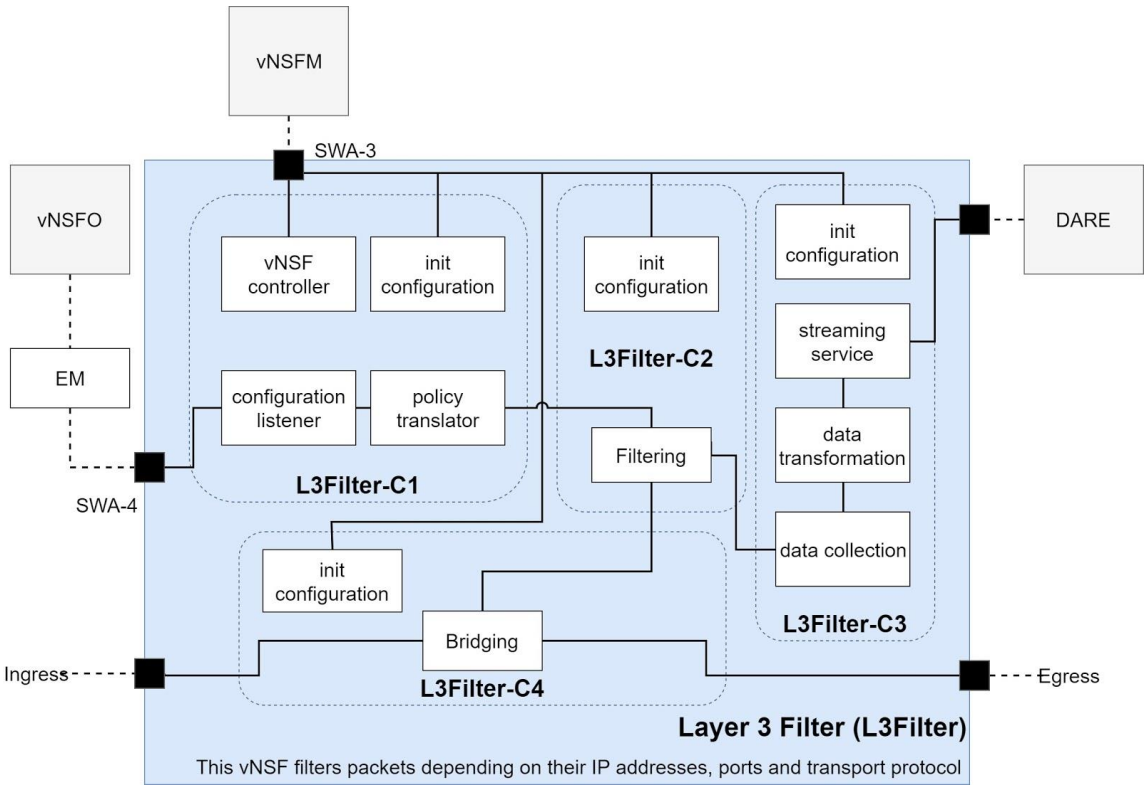

Figure 3: L3Filter architecture.

Since D3.2, in which the general architecture of L3 Filter is presented, some critical changes have been added -- mostly concerning the traffic steering. In particular, as shown in Figure 3, it has been added a new VNFC module called "L3Filter-C4" in charge of the traffic forwarding between the ingress and egress interfaces. To summarize these changes, we will show below the new L3 Filter components:

- **L3Filter-C1 (Policy translation):** we modified this VNFC in order to provide only the policy translation and the general control functionalities (e.g. configurations listener, safe start and stop of the vNSF). All this work has been made in order to relieve this component from the traffic steering. Once the policies are received from the Element Management, they will be translated and sent to the L3Filter-C2 component.
- **L3Filter-C2 (Filtering):** at this stage the policies are expressed in terms of filtering rules and they will be applied from the Filtering module which communicates with the L3Filter-C4 VNFC in order to finally block or allow the traffic flows.
- **L3Filter-C3 (DARE collector):** this module has not been changed and it is in charge of collecting the application metrics and sending it to the DARE for further analysis.

- **L3Filter-C4 (Bridging):** after receiving the information from the Filter module of the L3Filter-C3 component, this new module enforces the filtering rules and it is in charge to perform the final action of allowing or denying a specific traffic flow.

The changes on the L3 Filter NS architecture and implementation leverage the bridging capabilities of the vNSF in order to dramatically simplify the process of the packets' forwarding and to enhance the portability and scalability of the vNSF as well. This achievement is granted by the fact that the routes configurations within the L3 filter are no more needed.

Leaving aside the major changes on the vNSF, we performed a series of functional tests as well. The main purpose of those tests was fixing minor bugs in the policy translation and rules enforcement processes. This allows us to achieve a very significant result in terms of vNSF robustness and resilience.

We should also notice that the L3 Filter vNSF needs three network interfaces in order to work properly: one for management and the other two for the traffic steering (ingress, egress). The management network must be configured with a DHCP server capable of setting the preferred DNS during the instantiation of an L3 Filter. This must be granted in order to allow the cloud-init properly configure at boot time.

3.4.7. Proxy TLS

Since D3.2, the development of the NS logic and its interfaces towards the vNSFO, the DARE and the vNSF Manager APIs have been finished. The definition of the NS remains unchanged.

The ProxyTLS allows to inspect HTTPS traffic with the aim of solving cybersecurity threats such as malicious URLs. One key difference in this proxy is the capability to inspect and log the complete URLs in the HTTPS header, in contrast to other security tools that can only see the TLD domain from the issued Certificate. The purpose of ProxyTLS is to monitor all HTTPS connections and log the URLs used by the clients. In addition, it can be populated with URL blacklists (i.e. malware droppers, C&C, phishing servers, etc.) to generate alerts and, in the case of a mitigation policy, block the connectivity to that specific URLs.

The NS and its vNSF provide a log generation compliant with Apache Spot "proxy" collector format and sends all captured proxy log to the Apache Spot cluster through the Kafka bus (both fully integrated).

Besides the list of the NSs presented above, additional ones ("l3attest" and "vpnattest" instances) were developed in the scope of the project to demonstrate how the vNSF attestation process alone works in the SHIELD platform. These are implemented as Docker containers that run a minimal Linux CentOS container image. These NSs are not documented in D3.2 and no architecture work was required prior to their development, since they do not carry any actual vNSF logic and are simple targets for attestation.

4. REQUIREMENTS MAPPING

The requirements, initially identified in D2.2, were categorised into four categories:

- Platform functional (PF) requirements, which detail the functionality required by the platform;
- Platform non-functional (NF) requirements, which detail the performance, ease of use and security of the platform;
- Service functional (SF) requirements, which describe the functionalities of the cybersecurity Network Services that the platform deploys; and
- Ethical and regulatory compliance (ERC) requirements, which focus on maintaining the platform's alignment with the EU regulatory landscape.

4.1.1. Compliance to requirements

The tables below map each WP3 component to specific requirements and justify how these requirements were fulfilled.

vNSF-related components

Table 1: Functional and non-functional requirements for vNSF-related components.

Components	Requirements	Justification
Store	PF02, PF10, PF11, PF15, PF17, PF22	The Store provides the ability to perform onboarding of NSs and vNSFs in an inter-operable and secure environment. At the time of onboarding, this component validates that the content is both valid and trusted. Multiple interfaces are provided, allowing this component to expose its persisted information to authorised parties.
Orchestrator	PF01, PF02, PF03, PF07, PF11, PF13, PF22	The vNSFO can deploy and perform lifecycle and status management on the vNSFs via the NFVO. It can also configure the vNSFs to deploy mitigation actions; interact with the Trust Monitor to register nodes for periodic attestation and provide secured communications with any client connecting to it.
	NF03	The vNSFO is aware of new nodes registered into the NFVI and propagates these to the Trust Monitor for attestation purposes.
Trust Monitor	PF08, PF11, PF16, PF19, PF22	The TM provides APIs to integrate attestation with other components of the platform, both for notification (through Dashboard and DARE) and for the

		implementation of security services (periodic attestation workflow supported by the vNSFO and Store). The TM implements both vNSF and network infrastructure attestation, and generates historical logs about attestation that are later stored in the DARE. The TM provides secure communication with other SHIELD components.
	NF01, NF07, NF08	The TM periodically attests (with a configurable latency) the NFVI in order to identify occurring incidents. It performs parallel attestation per node so as to minimise the response time of periodic attestation, whose bottleneck is represented by the latency introduced by TPM. The TM adopts well-established standards proposed by the Trusted Computing Group for the definition of integrity verification formats and remote attestation workflow. The TM has been implemented as a multi-container application for automated deployment; with minimal user intervention.

NSs and vNSFs

Table 2: Functional and non-functional requirements for NSs and vNSFs.

Components	Requirements	Justification
DPI	PF01, PF02, PF03, PF04, PF05, PF06, PF10, PF13, PF14, PF15, PF20, PF22	The DPI fulfills all requirements necessary for lifecycle management and has demonstrated its main functionality within two Y2 demos (Worm & Slowloris scenarios).
	NF01, NF02, NF03, NF05, NF06, NF07, NF09	The deployment time for the selected flavour was adequate; the service did not feature any significant downtime or functional problems.
	SF02, SF04, SF05, SF06, SF08, SF09, SF10	The main functionality of the DPI is to classify L7 traffic per domain (e.g. YouTube, twitter etc.) or per application type (e.g. bittorrent, NetBIOS/SMB etc.). These capabilities allow the vDPI to monitor suspicious traffic (e.g. a rise in SMB traffic can signify existence of WannaCry malware etc.). The DPI monitored offensive network traffic in the Slowloris scenario.
Forward L7 Filter	PF01, PF02, PF03, PF06 PF10, PF13,	The Forward L7 Filter can be deployed by the orchestration platform on different PoPs and

	PF14, PF15, PF20, PF22	domains. All installation and configuration processes are implemented in an automatic manner, without further manual intervention. The two-step configuration method (Day-0 - Day-1) besides allows managing the entire life-cycle of this vNSF (e.g. start, stop, deploy, set-policies, delete-policies). It is also possible to access to the life-cycle management through a Graphic User Interface (GUI) provided by OSM. The NS provides secure management communication with the SHIELD environment.
	NF05	When the traffic passes through the Forward L7 Filter, in order to be inspected and eventually filtered, there is no perceived impact on its performance.
	SF09	This NS provides a filtering functionality for specific Layer 7 protocols and headers (e.g. HTTP, FTP). The traffic passes through its vNSF and is redirected to a server only if it doesn't match the defined rules of a blacklist. This filtering functionality is supported by a log in which are stored all the filtering events. This log could be used in order to monitor the filtered traffic and to generate alerts.
HTTPS Analyzer	PF01, PF02, PF03, PF04, PF10, PF15, PF22	This NS can be deployed in any SHIELD PoP, covering all requirements to allow its lifecycle management and it provides interfaces to send the collect events to SHIELD.
	NF07	This VNF can generate NetFlow v9 standard format for HTTP and any other type of traffic.
	SF09	HTTPs Analyzer allows the classification of encrypted traffic traversing the network and therefore enable its correlation with other security sources to detect attacks therefore improving its detection/mitigation mechanisms.
IDS	PF01, PF02, PF03, PF04, PF14, PF15, PF18	This NS can be deployed on different PoPs. Full lifecycle management (on-boarding, instantiation and termination) of its vNSF is supported through the vNSFO and OSM GUI. Occuring alerts, identified through the IDS vNSF can be provided to DARE (RabbitMQ notification of incidents to the SHIELD Dashboard). Each tenant might be associated with a specific and single instance of this vNSF.

	NF01, NF03, NF05, NF07	New events are reported to DARE within a short time (less than a second). If required, horizontal scaling of this NS is possible. When this NS was used, no impact on the user perceived delay was noted as in all configurations we experimented its vNSF was connected off-path. The output of this vNSF is in JSON format.
	SF02, SF04, SF06, SF08, SF09	This NS provides the means to detect a variety of attacks (malware, DoS, volumetric attacks, access to blacklisted sites and domains) by analysing the network traffic (L3, L4 and L7). The ability to deploy this NS in an IDPS mode exists, so filtering the traffic based on the predefined rules is possible.
L23 Filter	PF01, PF02, PF03, PF14, PF15, PF18	This NS can be deployed on different PoPs. Full lifecycle management (on-boarding, instantiation and termination) of this vNSF is supported through the vNSFO (OSM GUI). Each tenant might be associated with a specific and single instance of this vNSF.
	NF03, NF05	If required, horizontal scaling of this vNSF is possible with the use of load balancer in the front. When this vNSF was used, no significant impact to the user perceived delay was noted.
	SF04, SF05, SF06, SF08, SF09	The L23 Filter is capable of filtering traffic packets at L2 and L3 of ISO/OSI stack. The filtering process is driven by a set of rules, typically based on the 5-tuple (src/dst IP addresses, src/dst ports, transport protocol). These and traffic limiting values are the fields supported for translation to MSPL. However, the actual configuration of the L23 Filter is based on OpenFlow rules, since our implementation is based in Open vSwitch (OVS). The L23 Filter could be easily integrated with the IDS or DPI NSs as a remediation NS, blocking malicious traffic detected by the IDS.
L3 Filter	PF01, PF02, PF03, PF06, PF10, PF13, PF14, PF15, PF20, PF22	This NS can be deployed on different PoPs and domains. All installation and configuration processes can be done automatically, with no further manual intervention. The two-step configuration method (Day-0 - Day-1) besides allows managing the entire life-cycle of this vNSF (e.g. start, stop, deploy, set-policies, delete-policies). It is also possible to access to the life-cycle management through a Graphic User Interface (GUI) provided by OSM. Thanks to the Day-1

		configuration, the vNSF is able to trigger, in case of a malicious event, proper actions to mitigate the threat. The scalability of the vNSF is managed by OSM and granted by the L3 Filter design, which is independent from the NFVI used for the deployment. L3 Filter provides secure management communication with the SHIELD environment.
	NF05	When the traffic flows pass through the L3 Filter, in order to be inspected and eventually filtered, there is no perceived impact on the performance.
	SF04, SF05, SF06, SF08, SF09	The L3Filter is capable of filtering traffic packets at Level 3 of ISO/OSI stack. The filtering process is driven by a set of rules typically based on IP addresses, ports and transport protocol. Since the L3 Filter is reconfigurable during its life-cycle, It could be easily integrated with the IDS/DPI as a remediation vNSF, blocking any malicious traffic detected by the IDS.
Proxy TLS	PF01, PF02, PF03, PF04, PF10, PF13, PF15, PF22	Proxy TLS allows its deployment in any SHIELD PoP, covering all requirements to allow its lifecycle management and It provides the needed interfaces tow send the collect events towards DARE and receiving instructions to mitigations based in http traffic filtering.
	NF04	When network traffic is proxied or analysed, the user experience is not degraded.
	SF02	The vNSF (middlebox) allows the monitoring of HTTP ciphered traffic directed at any server (in order to identify attacks). The ProxyTLS will inspect the HTTP headers. It will compare the URLs from the headers against blacklists to detect any of the malicious endpoints.

5. ENVIRONMENT SETUP GUIDE

This guide covers the setup of those tools in use to define the functionality of the infrastructure. Here, we provide information on how to setup the Virtual Infrastructure Manager (where the Virtual Machines and Networks are defined for the vNSFs), the NFV Orchestrator (to deploy any NS and vNSF in specific VIMs) and the ODL controller and Aruba 3800 switch (to control and allow, specifically, the insertion of flows to dynamically control the network).

5.1.1. NFV Orchestrator

Two versions of the NFVO were deployed, following the adoption of new releases.

5.1.1.1. OSMr2

Initially, the NFVO (OSMr2) was installed in ORION's VIM. The deployment of this tool was carried out in a VM with 8 CPUs, 16 GB RAM and 100 GB disk; as required by the direct deployment using its source code -- potentially allowing modifications on the source code of OSM. Instructions are available in [1].

This instance of the NFVO is located within the ORION VPN and is exposed through a public IP. There were no changes in the source code or extra configuration, other than a minor modification on the address to upload the packages, which is hosted in the "SO-ub" container.

5.1.1.2. OSMr4

OSMr4 introduced several changes, both at the end-user side and at the northbound APIs. This release was later installed in i2CAT's VIM. Even though the requirements are considerably smaller, he deployment of this tool was carried out in a VM with the same characteristics as OSMr 2 (8 CPUs, 16 GB RAM and 100 GB disk). Instructions are available in [2].

This instance of the NFVO is exposed through a public IP. In this deployment there were no changes in the source code of the containerised processes.

5.1.2. Virtual Infrastructure Manager

During the lifetime of the project we have used 2 production-enabled VIMs using OpenStack and deployed in Athens and Barcelona. After this we incorporated a third site to test the container-based NSs, using VIM-emu and deployed in Torino.

Besides these, other VIMs were in place during the lifetime in the project to serve for local deployments, for instance for the development and testing of any Network Service. As an example, TID deployed its own OpenStack, based on RDO [3]; where their services were first tested.

5.1.2.1. OpenStack (Athens)

The OpenStack environment hosted by ORION is an All-In-One deployment based on Ocata release on top of CentOS 7.5 and it is used for the deployment of the NSs in SHIELD. Instructions to deploy are available in the official OpenStack documentation [4]. Furthermore, instances of additional components of SHIELD (i.e. the Store, vNSFO, DARE, Dashboard, TM etc) are hosted on 3 ESXi servers.

After the deployment, the OpenStack server must be connected to the nodes where the NSs and vNSFs are deployed. This is linked to the HP Aruba (3800) switch, which directs the traffic in and out of the virtualisation nodes by tagging with specific VLANs, each assigned to its own network. The HP 3800 is connected to another switch, and that to a router and a firewall; before reaching the public Internet. The ORION infrastructure can be accessed via the CISCO's AnyConnect VPN or OpenConnect VPN. Partners accessing the infrastructure must point their VPN client towards "https://vpn.medianetlab.gr" and log-in with their given credentials.

5.1.2.2. OpenStack (Barcelona)

The OpenStack instance in Barcelona a production-enabled environment that runs the Ocata distribution on top of CentOS 7.5.1804. This instance was installed as an All-In-One (AiO) Openstack through Ansible scripts [5].

Before starting the installation, the partitions for root ("`/`"), swap, volumes ("`/cinder`") and other OpenStack data ("`/openstack`") must be defined first, then its mounting point must be set.

Then, the configuration for OpenStack Ansible starts. Features like VLAN shall be enabled, bridges must be created (manually work best), the definition of the Swift and Nova loopback disks shall take place, as well as other data such as the bridge to link the virtual interfaces of the VMs deployed by OpenStack.

Finally, run the OpenStack bootstrap script, verify that the environment is according to the configuration defined before running such script and run the whole setup via its specific playbook.

After the deployment, the OpenStack server must be running a total of 23 LXC containers, each with its own specific service and have enabled different bridges for the management, storage, connectivity (VLAN, VxLAN and/or others, as defined in the previous configuration step). This is linked to the HP Aruba (3800) switch and to another switch, which is connected to a router for Internet access. Partners accessing the infrastructure are to given access via their OpenVPN client and log-in with their given credentials.

5.1.2.3. VIM-emu (Torino)

The VIM-emulator instance for testing the container-based NSs does run on a single host machine equipped with CentOS Linux 7.5.1804 and Trusted Platform Module (TPM) version 1.2.

The host runs kernel 4.4.19 with a modified version of the Integrity Measurement Architecture (IMA) module to support container attestation. It also runs an instance of the OpenAttestation (OAT) HostAgent software for measuring the platform trust along with containers' individual

trust [6]. Also, the host runs the Docker Container engine version 19.03.0-dev, which can be installed from package system [7].

The VIM-emulator platform is installed on the same host through a modified version of the OSM-provided scripts [8] in order to support the CentOS distribution.

5.1.3. OpenDayLight Controller

The OpenDayLight (ODL) controller enables the control of the HP Aruba 3800 switch and allows to dynamically manage its network flow rules. The OpenDaylight distribution instantiated is Carbon 0.6.3, it is deployed in both Athens and Barcelona; yet the latter is to be used from OSMr4.

Java 8 is required for its installation. Then, simply download the corresponding OpenDaylight packaged archive with the precompiled binaries, extract it and run the Karaf subsystem binary ("`bin/start`"). Through the Karaf console, numerous "feature" packages must be installed to have a working and compatible environment. The feature "`odl-dluxapps-applications`" enables the DLux GUI for instance, and since OpenDaylight does not support the control of layer 2 devices out of the box, multiple other features are required to enable this functionality. The following command in Karaf console will install the comprehensive list of features needed for this setup:

```
feature:install odl-mdsal-models odl-aaa-shiro odl-akka-scala-2.11 odl-akka-
system-2.4 odl-akka-clustering-2.4 odl-akka-leveldb-0.7 odl-akka-
persistence-2.4 odl-netty odl-netty-4 odl-guava-18 odl-lmax-3 odl-triemap-
0.2 odl-restconf-all odl-restconf odl-restconf-noauth odl-mdsal-apidocs odl-
yangtools-yang-data odl-yangtools-common odl-yangtools-yang-parser odl-aaa-
api odl-aaa-authn odl-aaa-encryption-service odl-aaa-cert odl-
openflowplugin-southbound-he odl-openflowplugin-nsf-model-he odl-
openflowplugin-app-lldp-speaker-he odl-mdsal-dom odl-mdsal-common odl-mdsal-
dom-api odl-mdsal-dom-broker odl-mdsal-binding-base odl-mdsal-binding-
runtime odl-mdsal-binding-api odl-mdsal-binding-dom-adapter odl-mdsal-eos-
common odl-mdsal-eos-dom odl-mdsal-eos-binding odl-mdsal-singleton-common
odl-mdsal-singleton-dom standard config region package http war kar ssh
management odl-openflowplugin-flow-services-ui odl-openflowplugin-flow-
services-rest odl-openflowplugin-flow-services odl-openflowplugin-southbound
odl-openflowplugin-nsf-model odl-openflowplugin-app-config-pusher odl-
openflowplugin-app-topology odl-openflowplugin-app-forwardingrules-manager
odl-l2switch-switch odl-l2switch-switch-rest odl-l2switch-switch-ui odl-
l2switch-hosttracker odl-l2switch-addresstracker odl-l2switch-arphandler
odl-l2switch-loopremover odl-l2switch-packethandler odl-dluxapps-
applications odl-dluxapps-nodes odl-dluxapps-topology odl-dluxapps-yangui
odl-dluxapps-yangman odl-dluxapps-yangvisualizer pax-jetty pax-http pax-
http-whiteboard pax-war odl-config-persister odl-config-startup odl-dlux-
core odl-config-netty odl-config-api odl-config-netty-config-api odl-config-
core odl-config-manager odl-openflowjava-protocol odl-mdsal-all odl-mdsal-
common odl-mdsal-broker-local odl-toaster odl-mdsal-xsql odl-mdsal-
clustering-commons odl-mdsal-distributed-datastore odl-mdsal-remoterpc-
connector odl-mdsal-broker
```

The VM or server running ODL must be in the same network as the Aruba 3800 switch. For example, the ODL VM in use for SHIELD is running on an OpenStack server, which is physically connected to the switch.

The setup of the interfaces is as follows:

- The server has a bridge that links the interface connected to the switch with the interface assigned to the VLAN in use for the switch (as well as external connectivity).
- The ODL VM has one interface connected to a virtual network (10.102.20.0/24).

The IP of the bridge in the server (e.g., 10.102.20.3) and the IP of the interface of the ODL VM (e.g., 10.102.20.9) must be in the same range.

To enable TLS encryption of the OpenFlow traffic, a CA must be created [9]. The intermediate pair has to be skipped, as the switch does not support chained certificates. The leaf certificates are signed directly with the root certificate by using the "`policy_loose`" preset.

On the switch, the following commands must be issued:

```
config term
  crypto pki ta-profile <CAProfile>
  copy tftp ta-certificate <CAProfile> <tftp server IP address> <certificate
file>
  crypto pki identity-profile SwitchIdentity subject <name/switch serial
number>
  crypto pki create-csr certificate-name <certName> ta-profile <CAProfile>
usage openflow
```

This will generate a Certificate Signing Request (CSR) in the terminal. The corresponding certificate, using the "`usr_cert`" preset, is created and then installed with the following command:

```
crypto pki install-signed-certificate
```

For ODL, a certificate must first be generated by using the "`server_cert`" preset:

```
sudo openssl pkcs12 -export -in <ODLCert.pem> -inkey <ODLPrivkey.pem> \
      -out ctl.p12 -name <odlserver>
```

The export password must be set to "opendaylight" - otherwise the ODL configuration shall be changed according to the password of choice. The following commands use this default password. Then, the certificate must be imported to the keystore:

```
keytool -importkeystore \
      -deststorepass opendaylight -destkeypass opendaylight -destkeystore
ctl.jks \
      -srckeystore ctl.p12 -srcstoretype PKCS12 -srcstorepass opendaylight \
      -alias odlserver
```

Also, the switch certificate must be imported into the truststore:

```
keytool -importcert -file SwitchCert.pem -keystore truststore.jks -storepass
opendaylight
```

Then the Java Key Store (JKS files: `ctl.jks` and `truststore.jks`) files are copied into the ODL folder "`./configuration/ssl`".

Finally, TLS is enabled in the OpenFlow connection via the following configuration:

```
#File:          /etc/opendaylight/datastore/initial/config/default-openflow-
connection-config.xml
      <transport-protocol>TLS</transport-protocol>
      <tls>
            <keystore>configuration/ssl/ctl.jks</keystore>
            <keystore-type>JKS</keystore-type>
            <keystore-path-type>PATH</keystore-path-type>
            <keystore-password>opendaylight</keystore-password>
            <truststore>configuration/ssl/truststore.jks</truststore>
            <truststore-type>JKS</truststore-type>
            <truststore-path-type>PATH</truststore-path-type>
            <truststore-password>opendaylight</truststore-password>
            <certificate-password>opendaylight</certificate-password>
      </tls>
```

5.1.4. Aruba 3800 switch

The setup of the Aruba 3800 switch requires its connection to the OpenFlow controller.

In order to be managed by the OpenDaylight controller, a dedicated VLAN must also be instantiated: the management VLAN. At least one port must be part of it and configured to reach the ODL controller (it is instantiated similarly to the managed network described below).

In order to create a managed network, another VLAN is necessary. To create a VLAN, instantiate an OpenFlow instance and assign ports to it, these commands should be issued on the switch console:

```
config term
  vlan <VLAN#>
    untagged <port number or range of port numbers>
    exit
  exit
```

At least one port shall be set as untagged for the VLAN, so that it is assigned to the VLAN.

On the managed network, a virtual OpenFlow instance is defined and configured to connect to the ODL controller. OpenFlow 1.0 must be enabled:

```
config term
  openflow
    controller-id  <controller_instance_id>  ip  <odl_vm_ip>  controller-
interface vlan <controller_vlan>
```

```
    instance "odl_<infrastructure_name>"
      version 1.0
      member vlan <managed_network_vlan>
      controller-id <controller_instance_id> [secure]
      flow-location hardware-only
      enable
      exit
    enable
    exit
  exit
```

If the ODL instance is secured with TLS, the `secure` option must be set when configuring the controller ID for the instance.

Following these commands, the general status of the OpenFlow instance can be monitored with `show openflow`, while the details of the instance can be checked with `show openflow instance <instance_name>`.

To allow communication between the switch and the Trust Monitor, SNMP has to be enabled. To enable SNMPv3, type:

```
config term
  snmpv3 enable
```

Then follow the on-screen instructions. Once finished, type:

```
snmpv3 user <username> auth sha <auth_password> priv aes <priv_password>
snmpv3 group managerpriv user <username> sec-model ver3
```

This sets the user passwords and encryption algorithms, and it also assigns the user to the manager/privacy group.

6. INSTALLATION GUIDES

6.1. Open Source status

The WP3 components are available at the SHIELD organization's public repository in GitHub [10]. All of them (contained in the repositories "trust-monitor", "store", "nfvo" and "vnsfs") are shipped under the Apache 2.0 license. Sources are tagged for specific versions.

The Trust Monitor's, the vNSF Store and the vNSF Orchestrator contents are located at the repositories named "trust-monitor", "store" and "nfvo" under the SHIELD organization with the link provided above. The contents of all these repositories are protected under the Apache 2.0 license and provide the source code of their component, the installation and deployment scripts and the documentation on how to deploy (manual or automated/Docker compose-based), to configure and on the definition of their northbound APIs.

On the other hand, the vNSF repository, located The vNSF Store's contents are located at the "vnsfs" repository under the same organization. This repository provides all the data required to create the NFVO-specific (OSM) packages and to wrap them for SHIELD (using the SHIELD packages). Typically, a NS and their vNSFs are expected to provide the following: the NS and vNSF descriptor (with the definition of the required metadata for the NFVO), optionally the Juju charm (for configuration through the NFVO), the SHIELD security manifest (for attestation purposes), the source code for the vNSF(s) and finally, the documentation on the deployment and verification for each NS, the links per interface for each vNSF and the physical requirements expected to run the NS.

We provide below the details on how each component is installed and configured.

6.2. Trust Monitor

The Trust Monitor component is deployed as a multi-container application which leverages the Docker Compose tool [11]. In order to install the application, the target environment should have both the Docker Container Engine and Docker Compose installed.

The component requires configuring the following:

- TLS encryption key/certificate for the web APIs;
- Whitelist database with reference measurements of the NFVI platform nodes;
- Different API connectors to integrate with other SHIELD components, namely vNSFO and vNSF Store (as well as Dashboard and DARE);
- Different attestation drivers to verify the integrity of different NFVI elements (compute hosts, switches).

Installation of dependencies, setup of the different sub-components and deployment of the full-fledged Trust Monitor application require minimal interaction of the user thanks to the automation of the whole process via the Docker Compose tool.

From the network perspective, the Trust Monitor requires network access to the NFVI (compute nodes and switches, as they are the target of attestation) and to the other SHIELD components (for proper integration in the platform).

6.3. vNSF Store

The vNSF Store component is deployed as a multi-container application using the Docker Compose tool. Hence, the only application requirements are Docker (17.06.0 or later) and Docker Compose (3.0 or later).
The setup and execution of the vNSF Store can be performed using the script "`run.sh`":

```
./run --environment .env.production --verbose
```

When the vNSF Store is run for the first time the DB must be initialized using the following:

```
docker exec docker_store-persistence_1 bash -c
"/usr/share/dev/store/docker/setup-datastore.sh --environment
/usr/share/dev/store/docker/.env.production"
```

The vNSF Store can be executed under different configuration environments. In the example above it is provided the default environment for production ("`.env.production`" file) however, different environments can be created or modified to meet the needed requirements.

6.4. vNSF Orchestrator

The vNSFO component is deployed using Docker Compose to run different containers (e.g., one for the vNSFO API, another for the Mongo database). Before deploying it, the target environment will be required to install the Docker Container Engine, the Docker Compose and other libraries.
The component requires configuring the following:

- TLS encryption key/certificate for the web APIs;
- Definition of path and behaviour for isolation scripts;
- Different API connectors to integrate with other SHIELD components, namely with the NFVO northbound API and address for subcomponents, the DB and the TM.

The generation of the certificate, the installation of dependencies and the setup of the different sub-components is performed automatically for the most part through the provided scripts.
From the network perspective, the vNSFO must be connected to the management network where the NSs are deployed (in order to configure their vNSFs) and reachable by all other SHIELD components interacting with it (i.e., the vNSF Store and TM, and also the Dashboard).

6.5. NSs and vNSFs

6.5.1. DPI

The version of DPI used in Y2 was implemented with PF_RING (a set of library drivers and kernel modules, which enable high-throughput packet capture and sampling).

The installation of the DPI components requires 2 virtual processors and is deployed in 40GB storage/4GB memory flavour. Traffic redirection is performed via the NFVI. The vDPI is configured to accept incoming traffic and outputs captures to the DARE for further analysis via

the integrated collectors. The required tools (PF-ring, DPI dashboard) and their dependencies (Git, Go, Beego, InfluxDB and Grafana) were installed via scripts, and are now bundled in a pre-installed image with the modules.

NFVI infrastructure side requirements:

- **Network requirements:** the management interface has to be attached to a link which grants the connection between the vNSFO and NFVO (OSM), the DARE (Spot) and the vNSF instance itself.
- **Traffic requirements:** the other two data interfaces must be used for the incoming and outgoing traffic flows. The strategy used to steer the traffic outside the vNSF depends completely on the NFVI infrastructure (e.g. SDN controller, etc.).

6.5.2. Forward L7 Filter

The Forward L7 Filter is implemented mainly using two web application-oriented technologies: Apache HTTP Server (HTTPD) and ModSecurity as detailed in D3.2. Regarding the installation and configuration process we can refer to the L3 Filter, which uses basically the same technologies. It all starts from a clean Ubuntu cloud image (16.04 LTS version) on which two main configuration steps are performed and summarized as follows:

- **Day-0 configuration:** at this stage, typically at the boot of an instance, every software-based requirement is installed through the cloud-init technology (e.g. Apache Server, ModSecurity) and initial configurations of vNSF (e.g. policy translation GO module). All this process is automatically performed, requiring no manual intervention.
- **Day-1 configuration:** after all the requirements are satisfied, we have a second stage of configuration, in which the element management (the juju charm) performs a critical role. At this stage we can configure and reconfigure every time, during the vNSF instance life-cycle, the set of rules enforced by the L7 filter. This process, as we said, is performed using a juju charm which is in charge of performing the following actions on an L7 Filter instance:
    - **set-policies:** push a new set of rules on the L7 filter and enforce them;
    - **get-policies:** return a set of rules enforced at the moment;
    - **delete-policy:** delete a specific rule in the set of the enforced rules;
    - **delete-policies:** delete all the rules at the moment enforced on the L7 filter.

Both the Day-0 and Day-1 configurations, included the juju charm installation, are performed automatically by the NFVO, requiring no other manual intervention or further configurations. After the installation of an L3 filter instance and its first configuration, we will be able to re-configure it, as we mentioned above, during all the vNSF life-cycle.

In order to deploy properly an L7 Filter instance, we need some requirements on the NFVI infrastructure side as well:

- **Network requirements:** since we have three different network interfaces, the NFVI infrastructure is in charge to attach properly the virtual links on each one of them. In particular, the management interface has to be attached to a link which grants the connection between the NFVO (OSM) and the vNSF instance itself.

- **Traffic steering requirements:** for what concerns the other two interfaces, they must be used for the incoming and outgoing traffic flows. The strategy used to steer the traffic outside the vNSF depends on the NFVI infrastructure (e.g. SDN controller, etc.).

6.5.3. HTTPS Analyzer

The process of installation is described in detail on its GitHub SHIELD public repository [12]. The process requires a standard Ubuntu 16.04LTS distro with 4GB of memory (8GB recommended) and 10GB hard disk (20GB recommended):

- Prerequisites: install related applications: softflowd, spot-nfdump, d-collector and tstat.
- Configure each of the components, including the interface that receives a copy of the client traffic and the DARE connectivity.
- Download and clone to the server the code available in the repository.
- Create a system init services to be used by the vNSFO API.

Alternatively, the https-analyzer vNSF is packetized in an image.

The NS has two physical interfaces: i) the management interface to transport all logic interfaces and APIs to interact with vNSFO and DARE, and the ii) data plane interface where the client traffic is captured.

The deployment process (onboarding & instantiation) for the vNSFs are described [13] in a standard way and provided in the “vnsfs” repository.

After instantiating, enforcing the security policies through vNSFO (juju charm) will allow to:

- **start-softflowd**: starts Softflowd - NetFlow network traffic analyzer.
- **stop-softflowd**: stops Softflowd - NetFlow network traffic analyzer.
- **restart-softflowd**: restarts Softflowd - NetFlow network traffic analyzer.
- **start-analyzer**: starts the HTTPS analyzer.
- **stop-analyzer**: stops the HTTPS analyzer.
- **restart-analyzer**: restarts the HTTPS analyzer.
- **forensic-mode**: changes tstat to real-time mode. It records one entry flow per completed transaction.
- **realtime-mode**: changes tstat to real-time mode. It records multiple entry flows per only one transaction.
- **change-network**: changes the path where the trained network for machine learning is placed.

NFVI infrastructure side requirements:

- **Network requirements:** the management interface has to be attached to a link which grants the connection between the vNSFO and NFVO (OSM), the DARE (Spot) and the vNSF instance itself.
- **Traffic requirements:** the dataplane interface must be used to receive a copy of the user traffic. The strategy used to copy the user traffic depends on the network infrastructure.

6.5.4. IDS

A base image of CentOs 7.0 was used for the installation of the IDS vNSF. The following installations were made on top of it:

- Snort IDS
  The public v2.9.12 Snort version was set up according to the Snort documentation.
- Rule Configuration service
  snort_configuration_api repository was checked out in the CentOs 7.0 server.
- Event Publisher service
  snort_data_exporter repository was checked out in the CentOs 7.0 server.
- CentOs services
  Five services were installed for automate execution of the above tools during start-up:
  - snort.service
  - snort_data_exporter.service
  - snort_data_exporter.timer
  - snort_rules_api.service
  - snort_u2json.service

Barnyard2, PulledPork and Snorby services (originally in the CHARISMA's vIDS vNSF image) were removed, as they were not required in the frame of SHIELD.

6.5.5. L23 Filter

Open vSwitch v2.9.0 was installed on a CentOs 7.0 base machine. Installation was according to the publicly available documentation for OVS. On top of it, the following installations were made:

- Rule Configuration service
  fw_configuration_api repository was checked out and configured to run as a service.
- Event Publisher service
  fw_data_exporter repository was checked out and configured to run as a service.

6.5.6. L3 Filter

The L3 Filter is implemented mainly using iptables, ebtables and the br_netfilter kernel module for the filtering functionalities. The traffic steering between the ingress and egress interfaces is enabled creating a Linux bridge between them. Almost all other configurations are performed using Python as scripting language. From these premises we can describe in the following steps the installation and configuration processes. It all starts from a clean Ubuntu cloud image (16.04 LTS version) on which are performed two main configuration steps summarised as follows:

- Day-0 configuration: typically at the boot of an instance, every software requirement (e.g. iptables, bridge-utils) is installed through cloud-init technology and initial configurations of vNSF (e.g. policy translation python module) take place. This process is automatically performed by cloud-init, requiring no manual intervention.
- Day-1 configuration: after all the requirements are satisfied, we have a second stage of configuration, in which the element management (the juju charm) performs a critical

role. At this stage we can configure and reconfigure every time, during the vNSF instance life-cycle, the set of rules enforced by the L3 filter. This process, as we said, is performed using a juju charm which is in charge of performing the following actions on an L3 Filter instance:

- **set-policies:** push a new set of rules on the L3 filter and enforce them;
- **get-policies:** return a set of rules enforced at the moment;
- **delete-policy:** delete a specific rule in the set of the enforced rules;
- **delete-policies:** delete all the rules at the moment enforced on the L3 filter.

Both the Day-0 and Day-1 configurations, included the juju charm installation, are performed automatically by the NFVO, requiring no other manual intervention or further configurations. After the installation of an L3 Filter instance and its first configuration, we will be able to re-configure it, as we mentioned above, during all the vNSF life-cycle.

In order to deploy properly an L3 Filter instance, we need some requirements on the NFVI infrastructure side as well:

- **Network requirements:** since we have three different network interfaces, the NFVI infrastructure is in charge of attaching properly the virtual links on each one of them. In particular, the management interface has to be attached to a link which grants the connection between the vNSFO and NFVO (OSM) and the vNSF instance itself.
- **Traffic steering requirements:** the other two interfaces must be used for the incoming and outgoing traffic flows. The strategy used to steer the traffic outside the vNSF depends completely on the NFVI infrastructure (e.g. SDN controller, etc.). After the latest changes on L3 Filter, this NS became totally transparent to the way we perform this task.

6.5.7. Proxy TLS

The process of installation is described in detail on the GitHub SHIELD public repository [14]. The process requires a standard Ubuntu 16.04LTS distro with 4GB of memory (8GB recommended) and 10GB hard disk (20GB recommended):

- Prerequisites: install procedure from SECURED PSA re-encrypt.
- Configure each of the components, including the interface that receive a copy of the client traffic and the DARE connectivity.
- Download and clone to the server the code available in the repository.
- Create a system init service to be used by the vNSFO API.

Alternatively, the Proxy TLS vNSF is packetized in an image.

It has three interfaces: i) management interface to transport all logic interfaces and APIs to interact with vNSFO and DARE, ii) client side data plane interface where the client traffic is received, and iii) Internet gateway interface.

The deployment process (onboarding & instantiation) for the vNSFs are described [15] in a standard way and provided in the “vnsfs” repository.

After instantiating, enforcing the security policies through vNSFO (juju charm) will allow to:

- **get-policies**: get the MSPL policies (GET).

- set-policies: set the MSPL policies. Non-empty fields are allowed (POST).
- delete-policy: delete a specific MSPL policy (POST).
- delete-policies: delete all MSPL policies (GET).
- add-url: adds any number of URLs to the alert/monitor list (POST).
- delete-url: deletes any number of URLs in alert/monitor list (POST).
- start: starts the man in the middle proxy (start_demo.sh) (GET).
- stop: stops the man in the middle proxy (GET).

The requirements on the NFVI infrastructure side are as follows:

- Network requirements: The management interface has to be attached to a link which grants the connection between the vNSFO and NFVO (OSM), DARE (Spot) and the vNSF instance itself.
- Traffic requirements: The other two interfaces must be used for the incoming (client side) and outgoing (network side) traffic flows. All HTTPS client traffic has to be redirected to the Proxy TLS. The strategy used to redirect the user traffic depends completely on the NFVI infrastructure (policy routing, SFC, etc.).

7. CONCLUSIONS

7.1. Status of vNSF ecosystem

This document presents the final version of the technical details of the vNSF environment. The document starts describing the final view of the architecture and description of the capabilities per component (in the "functional layout") as well as the updates from the last deliverable in WP3 (D3.2). This is done in order to provide concise documentation on these components, based on previous provided information.

After this we provide the final mapping of requirements and how each component fulfils them. That covers platform functional and non-functional requirements, as well as service functional and ethical compliance requirements. The installation and configuration guides are provided to give a high-level view on how to deploy the SHIELD platform from scratch, by following these steps -- related both to the environment and 3rd party tools required for the platform, but also to the deployment and configuration for the WP3 components described above. Concluding, it can be stated that all components of the vNSF environment have been developed, fulfilling all mandatory requirements, as listed in D2.2. All modules are available as open-source under the project's GitHub organisation account.

7.2. Future work

The work of all WP3 tasks (i.e., mostly on the development side) concludes with the delivery of this report. The integration and assessment procedures of the WP3-based components with the rest of components in the SHIELD platform will continue until the end of WP5 and the termination of the project itself. The results of the WP3 activities will be provided in the final, upcoming demonstrations; and may be advertised and exposed through dissemination efforts like events (Winter School) and papers.

REFERENCES

[1] "OSM release TWO," [Online]. Available: https://osm.etsi.org/wikipub/index.php/OSM_Release_TWO. [Accessed November 2018].

[2] "OSM release FOUR," [Online]. Available: https://osm.etsi.org/wikipub/index.php/OSM_Release_FOUR. [Accessed November 2018].

[3] "Red Hat OpenStack," [Online]. Available: https://www.rdoproject.org. [Accessed November 2018].

[4] "OpenStack RDO installation for Ocata on CentOS," [Online]. Available: https://docs.openstack.org/ocata/install/rdo-services.html. [Accessed November 2018].

[5] "OpenStack Ansible installation guide for Ocata," [Online]. Available: https://docs.openstack.org/openstack-ansible/ocata/. [Accessed November 2018].

[6] "Open Attestation framework," [Online]. Available: https://github.com/OpenAttestation/OpenAttestation. [Accessed November 2018].

[7] "Getting Docker Community Edition for CentOS," [Online]. Available: https://docs.docker.com/install/linux/docker-ce/centos/. [Accessed November 2018].

[8] "VIM-emu component," [Online]. Available: https://osm.etsi.org/wikipub/index.php/VIM_emulator. [Accessed November 2018].

[9] "OpenSSL Certificate Authority," [Online]. Available: https://jamielinux.com/docs/openssl-certificate-authority/. [Accessed November 2018].

[10] "SHIELD organisation in GitHub," [Online]. Available: https://github.com/shield-h2020. [Accessed November 2018].

[11] "Docker Compose," [Online]. Available: https://docs.docker.com/compose. [Accessed November 2018].

[12] "HTTPS Analyzer in SHIELD," [Online]. Available: https://github.com/shield-h2020/vnsfs/tree/master/src/vnf/httpsanalyzer. [Accessed November 2018].

[13] "HTTPS Analyzer's deployment in SHIELD," [Online]. Available: https://github.com/shield-h2020/vnsfs/blob/master/doc/ns/httpsanalyzer/deployment.md. [Accessed November 2018].

[14] "ProxyTLS NS in SHIELD," [Online]. Available: https://github.com/shield-h2020/vnsfs/tree/master/src/vnf/proxytls/star. [Accessed November 2018].

[15] "ProxyTLS NS deployment in SHIELD," [Online]. Available: https://github.com/shield-h2020/vnsfs/blob/master/doc/ns/proxytls/deployment.md. [Accessed November 2018].

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

Acronym	Meaning
ACL	Access Control List
AiO	All-in-One
API	Application Programming Interface
DB	Database
C&C	Command and Control
CA	Certification Authority
CSR	Certificate Signing Request
DARE	Data Analysis and Remediation Engine
DoS	Denial of Service
DPI	Deep Packet Inspection
EM	Element Management
ERC	Ethical and Regulatory Compliance (requirements)
ESXi	Elastic Sky X
IDPS	Intrusion Detection and Prevention System
IDS	Intrusion Detection System
IMA	Integrity Measurement Architecture
IP	Internet Protocol
IPS	Intrusion Prevention System
ISP	Internet Service Provider
JKS	Java Key Store
JSON	JavaScript Object Notation
LXC	LinuX Containers
MAC	Media Access Control
ML	Machine Learning
MSPL	Medium-level Security Policy Language

NF	Non-functional (requirements)
NFV	Network Function Virtualisation
NFVI	NFV Infrastructure
NFVO	NFV Orchestrator
NS	Network Service
ODL	Open Day Light
OAT	Open Attestation
OSI	Open Systems Interconnection
OSM	Open Source MANO
OVS	Open vSwitch
PF	Platform Functional
PFR	PF Requirement
PoP	Point of Presence
PSA	Personal Security Application
RDO	Red Hat OpenStack
REST	REpresentational State Transfer
SDN	Software-Defined Networking
SF	Service Functional (requirements)
SFC	Service Function Chaining
SMB	Server Message Block
SNMP	Simple Network Management Protocol
SO	Service Orchestrator
SPI	Stateful Packet Inspection
TC	Trusted Computing
TLS	Transport Layer Security
TM	Trust Monitor
TPM	Trusted Platform Module
TSTAT	TCP STatistic and Analysis Tool

VIM	Virtual Infrastructure Manager
VLAN	Virtual Local Area Network
VxLAN	Virtual Extended Local Area Network
VM	Virtual Machine
VNF	Virtual Network Function
VNFC	VNF Component
vNSF	Virtual Network Security Function
VNFC	vNSF Component
vNSFM	vNSF Manager
vNSFO	vNSF Orchestrator
VPN	Virtual Private Network

ANNEX A. REGULATORY COMPLIANCE

D3.2 provides an overview of the EU regulatory ecosystem that affects SDN/NFV adoption such as GDPR, ePrivacy, net neutrality etc. A set of regulatory compliance specifications was created, for each WP3 component that parses personal data in any form. D2.3 also provides compliance requirements. This Annex updates the mapping of Requirements to the individual components.

vNSF-related components

The following table is updated from the tables with the Ethical and Regulatory Compliance (ERC) requirements in D3.2.

Table 3: Ethical requirements for vNSF-related components.

Components	Requirements	Justification
vNSF Store	-	No ERC requirements were identified for this component.
vNSF Orchestrator	-	No ERC requirements were identified for this component.
Trust Monitor	ERC07	The TM sends a notification to the SHIELD client user if it detects a breach in a SHIELD NS. The TM sends a notification to the SHIELD administrator user if it detects a breach in a SHIELD NFVI node.

NSs and vNSFs

The following table is updated from that in D3.2 (table 20).

Table 4: Ethical requirements for NSs and vNSFs.

Components	Requirements	Justification
DPI	ERC01, ERC02, ERC03, ERC06	The DPI NS does not retain network traffic thus it does not provide an interface to erase it. Information on its regulatory compliance (the specifications) is provided on its graphical user interface. The DPI does not throttle network traffic. The DPI has the capability to share network data with the DARE.
Forward L7 Filter	ERC01, ERC02, ERC05	Since D3.2, there have not been substantial changes. The NS does not retain personal data and does not allow data to be modified or erased. The NS does not

		change public IP addresses or encrypts traffic, hence it does not require to integrate a LI system.
HTTPS Analyzer	ERC01, ERC02, ERC03, ERC04, ERC06, ERC08, ERC11, ERC12	No personal data is retained to be portable, modified or erased in this NS. The classification is done by a automate process to assign a label, based in machine learning techniques, not modification or alteration is done in the labeling or in the traffic to bias the process. The process is design to use only Layer 3-4, data therefore no payload is analyzed. This is a clear effort to preserve the user privacy on the communication content in the vNSF design.
IDS	ERC01, ERC02, ERC03	The IDS NS does not retain personal data (e.g. network traffic) thus it does not provide an interface to erase it. The only information retained are the user-defined rules for alerting which are stored in a specific file of the operating system and also in a database. A REST API is provided and can be used by the tenant for CRUD operations on the IDS rules and the ability to completely remove and also re-generate the database with the same information. The IDS vNSF does not throttle network traffic and has the capability to share alert information with the DARE.
L23 Filter	ERC01, ERC02, ERC05, ERC09	Since D3.2 there have not been substantial changes. The NS does not retain personal data and does not allow data to be modified or erased. Its vNSF does not change public IP addresses or encrypts traffic, hence it does not require to integrate a LI system.
L3 Filter	ERC01, ERC02, ERC05, ERC09	Since D3.2 there have not been substantial changes. The NS does not retain personal data and does not allow data to be modified or erased. The NS does not change public IP addresses or encrypts traffic, hence it does not require to integrate a LI system.
Proxy TLS	ERC01, ERC02, ERC03, ERC04, ERC06, ETC08	No personal data is retained to be portable, modified or erased in the NS. The classification between malicious or benign traffic is done based on known blacklist or provided by the ISP; whilst traffic filtering is based on security reasons.