Advanced Networked Agents for Security and Trust Assessment in CPS/IoT Architectures

D3.5

Final Security Orchestrator Report

This deliverable presents the final result of the ANASTACIA Task 3.2, which aims to provide efficient orchestration of the SDN, NFV and IoT domains in order to enforce the security policies.

Distribution level	PU
Contractual date	31.10.2019 [M34]
Delivery date	31.10.2019 [M34]
WP / Task	WP3 / T3.2
WP Leader	AALTO
Authors	Miloud Bagaa, Mohammed Boukhalfa, Tarik Taleb (AALTO) and Alejandro Molina Zarca (UMU)
EC Project Officer	Carmen Ifrim carmen.ifrim@ec.europa.eu
Project Coordinator	Softeco Sismat SpA Stefano Bianchi Via De Marini 1, 16149 Genova – Italy +39 0106026368 stefano.bianchi@softeco.it
Project website	www.anastacia-h2020.eu

ANASTACIA has received funding from the European Union's
Horizon 2020 research and innovation programme under Grant Agreement N° 731558
and from the Swiss State Secretariat for Education, Research and Innovation.

Table of Contents

Index of figures

PUBLIC SUMMARY

This deliverable presents the final results of the ANASTACIA Orchestration plan Task 3.2, which aims to deploy an efficient orchestrator based on the SDN and NFV technologies, and ensures interacting with others framework components to refine and enforce the security policies. In addition, this document describes the system model that can store and provide the necessary information.

The security orchestrator is a fundamental component responsible on transforming the security policy provided by policy interpreter to enabler configuration. To clarify although the orchestrator is involved in refining the policy and selecting the enablers, its main role is deploying the security enabler and execute the final configuration. It also supervises the underlying infrastructure for any potential flaws. And it supports a variety of security capabilities of different categories, namely: SDN security capabilities, NFV security capabilities and IoT security controls.

The system model is an independent component in charge of storing the cloud data and the relevant infrastructure with the devices information, also it keeps track of the enforcement process. All Anastacia components have access to the system model in order to post and retrieve data.

In this report, we describe the implemented orchestration architecture and we detail the functionalities provided by each component. We discuss the system model schema and the API endpoints.

1 INTRODUCTION

1.1 AIMS OF THE DOCUMENT

This document is included in ANASTACIA WP3 “Policy Enforcement and Run Time Enablers”, in which we develop the orchestration system and interact with plan elements based on the designed mechanisms to refine and enforce the convenient high abstraction policy offered by the admin in the user plan. The enforcement process takes advantage of the SDN and VNF technologies, furthermore the orchestrator implement an optimization layer that monitors the system resource and quality-of-service in order to ensure efficient attack mitigation without affecting the quality-of-service in different verticals.

This deliverable as mentioned is about the centric orchestration component that ensures deploying the necessary security policies either as a reaction to a detected attacker as proactive measures set by the users of the framework. Additionally, the enforcement process include optimization layer in which we monitor periodically the resources and quality of service in different deployed instance

The rest of the document is organized as follows: Section 2 will tackle the relevant concepts to the refinement and enforcement process of the high-level security policies, as part of the interaction within WP3 and WP4 components. Section 3 will discuss the core Security Orchestration and Security Enforcement Plane. Section 4 explains the monitoring which collect resources and QoS data. Section 5 will explain the analysis of the data collected in the previous section using intelligent algorithms. Section 8 presents the security Orchestrator optimizer. Section 9 details the system model and the rest API communication. Finally, section 10 Implementation and Integration process used to deploy all the developed parts.

1.2 APPLICABLE AND REFERENCE DOCUMENTS

This document refers to the following documents:

- ANASTACIA project deliverable D1.3 – Initial Architecture Design.
- ANASTACIA Grant Agreement N°731558 – Annex I (Part A) – Description of Action.
- ANASTACIA Consortium Agreement v1.0 – December 6th 2016.
- ANASTACIA deliverable D1.1 – Holistic Security Context Analysis.
- ANASTACIA deliverable D1.2 – User-centred Requirement Initial Analysis.
- ANASTACIA deliverable D2.1 – Policy-based Definition and Policy for Orchestration, initial report.
- ANASTACIA deliverable D2.5 – Policy-based Definition and Policy for Orchestration, final report.
- ANASTACIA deliverable D3.2 – Initial Security Orchestrator Report , , initial report.
- ASASTACIA deliverable D3.4 – Final Security Enforcement Manager Report.

1.3 REVISION HISTORY

Version	Date	Author	Description
0.1	28.06.2019	Miloud Bagaa (AALTO)	First table of contents and first contribution
0.2	15.07.2019	Miloud Bagaa and Mohamed Boukhalfa (AALTO)	Table of contents update

0.3	10.08.2019	Miloud Bagaa (AALTO)	Section 2-3
0.4	25.08.2019	Miloud Bagaa (AALTO)	Section 4-5
0.5	15.9.2019	Mohammed Boukhalfa (AALTO)	Section 7
0.6	17.10.2019	Miloud Bagaa (AALTO)	Section 6
0.8	20.10.2019	Miloud Bagaa, Mohammed Boukhalfa (AALTO)	Revision sections 2 and 3
0.9	23.10.2019	Miloud Bagaa (AALTO)	Revision sections 4 and 5
1.0	24.10.2019	Mohammed Boukhalfa (AALTO)	Revision section 6
1.1	25.10.2019	Miloud Bagaa (AALTO)	Revision section 7
1.2	28.10.2019	Miloud Bagaa (AALTO)	Internal review
1.3	30.10.2019	Mohammed Boukhalfa (AALTO)	Internal review
1.4	31.10.2019	Miloud Bagaa (AALTO)	Revision section 7 and internal review
1.5	31.10.2019	Alejandro Molina Zarca (UMU)	Internal review, contribution to section 3.2

1.4 Acronyms and Definitions

Acronym	Meaning
SDN	Software Defined Networking
NFV	Network Function Virtualization
IoT	Internet of Things
QoS	Quality of service
VNF	Virtual network function
SFC	Software function chain

OSM	Open source Management and Orchestration (MANO)
SO	Security orchestrator
VIM	Virtual Infrastructure Manager
MAS	Mitigation Action Service
ONOS	Open Network Operating System
OVS	Open vSwitch
IDS	Intrusion Detection Systems
LSPL	Low-level Security Policy Language
M2M	Machine-to-machine
AAA	Authentication, Authorization, Accounting
TPM	Trusted Platform Module
PCR	Platform configuration register
TXT	Trusted execution technology
MANO	Management and Orchestration
SM	System Model
HSPL	High-level Security Policy Language
MSPL	Medium-level Security Policy Language
RQM	Resource and QoS monitoring
DA	Data analytics
SOO	Security orchestrator optimizer
LCM	Life cycle management
SOE	Security orchestrator engine
H2M	High to Medium

M2L	Medium to Low

2 Security Orchestrator And Security Enforcement Plane

In ANASTACIA framework, the security orchestrator oversees orchestrating the security enablers according to the defined security policies. The latter would be generated either by the end-user or received from the monitoring and reaction plane. The security orchestration plane [1] [2], through its components security orchestrator, security resource planning and policy interpreter, is able to coordinate the policies and security enablers to cover the security configuration needed for different communications that could happen in the network. The security orchestration plane takes into account the policies requirements and the available resources in the underlying infrastructure to mitigate the different attacks while reducing the expected mitigation cost and without affecting the QoS requirements of different verticals. The resources in the underlying infrastructure refer to the available amount of resources in terms of CPU, RAM, and storage in different cloud providers, as well as the bandwidth communication between these network clouds.

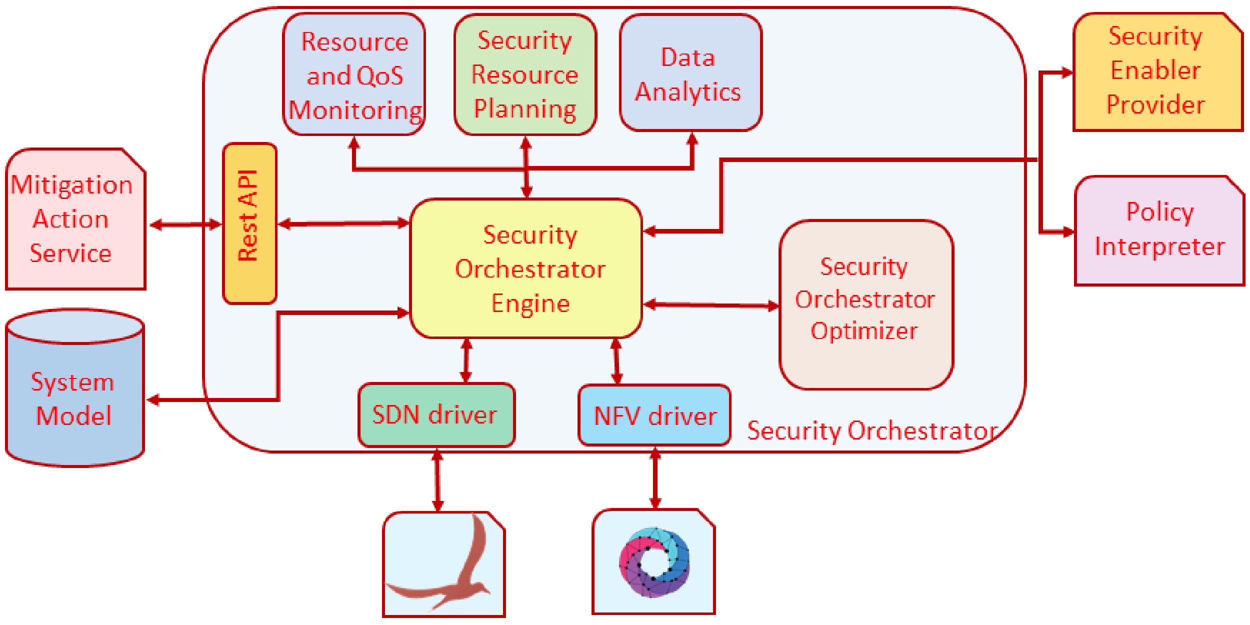

Resource and QoS Monitoring
Security Resource Planning
Data Analytics
Security Enabler Provider
Mitigation Action Service
Rest API
Security Orchestrator Engine
Security Orchestrator Optimizer
Policy Interpreter
System Model
SDN driver
NFV driver
Security Orchestrator

Figure 1 Security orchestration plane

In the second phase of the project, we have revised the security orchestration system mainly by adding three new components: *i)* Security orchestrator optimizer (SOO); *ii)* Resource and QoS monitoring component (RQM); *iii)* Data analytics (DA) component. More information about these components will be provided in sections 4 and 5, respectively.

Figure 1 depicts the updated version of the security orchestration and enforcement plane suggested within ANASTACIA framework. Using SDN technology, the IoT domain will be connected to the cloud domain, whereby different IoT services are running. Mainly, the user accesses to the IoT devices, first, through the cloud domain, then the SDN enabled network and the IoT brokers (e.g., MQTT broker). In fact, in ANASTACIA, the communication between a user and an IoT device happens through a chain of virtual network functions (VNFs) named service function chaining (SFC). The latter consists of three parts:

(i) The ingress point, which is the first VNF in the SFC. The user initially attaches to the ingress point;
(ii) The intermediate VNFs;
(iii) The egress point, which is the last VNF in the SFC. The egress point should be connected to the IoT controller. The order of the communications between the VNFs is defined according to the

different SDN rules enforced thanks to the SDN controller. The nature and the size of the SFC would be defined according to the nature of the user (a normal or a suspicious).

The security Orchestrator leverages open source MANO (OSM) framework designed by ETSI for managing and orchestrating VNF [3]. It used to instantiate and orchestrate different security enablers based on its main components bellow:

- The Service Orchestrator (SO): is responsible of end-to-end service orchestration and provisioning. It offers a web interface and a catalog that holds the different NFV descriptors.
- The Resource Orchestrator (RO): is used to provision services over a particular IaaS provider in a given location. It interacts directly with the VIM in order to instantiate virtual resources.
- The VNF Configuration and Abstraction (VCA): performs the initial VNF configuration and constant monitoring using Juju Charms LXD containers [4].

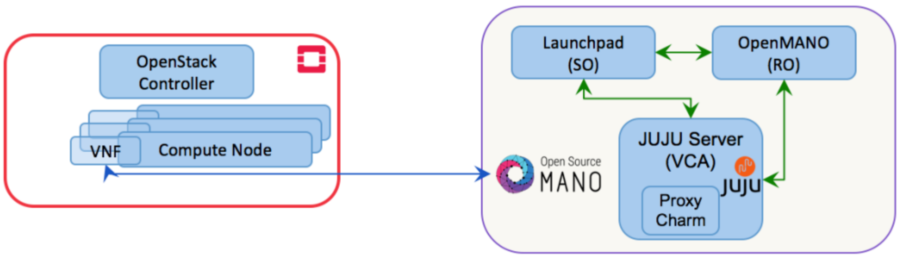


Figure 2 Overview of Open Source Mano components

The network managing is done using Open Network Operating System which is an open source project that aims to create an SDN operating system for communications and service providers, ONOS is accountable for managing network resources and enabling the programming of the underlying network. It uses standard protocols, such as OpenFlow and NetConf in order to expose advanced traffic manipulation functions through its applications. It controls multiple Open Virtual Switches (OVS) in order to enable the following functionalities: Traffic forwarding (steering) to VNFs, Traffic mirroring to different VNFs. Traffic dropping [5].

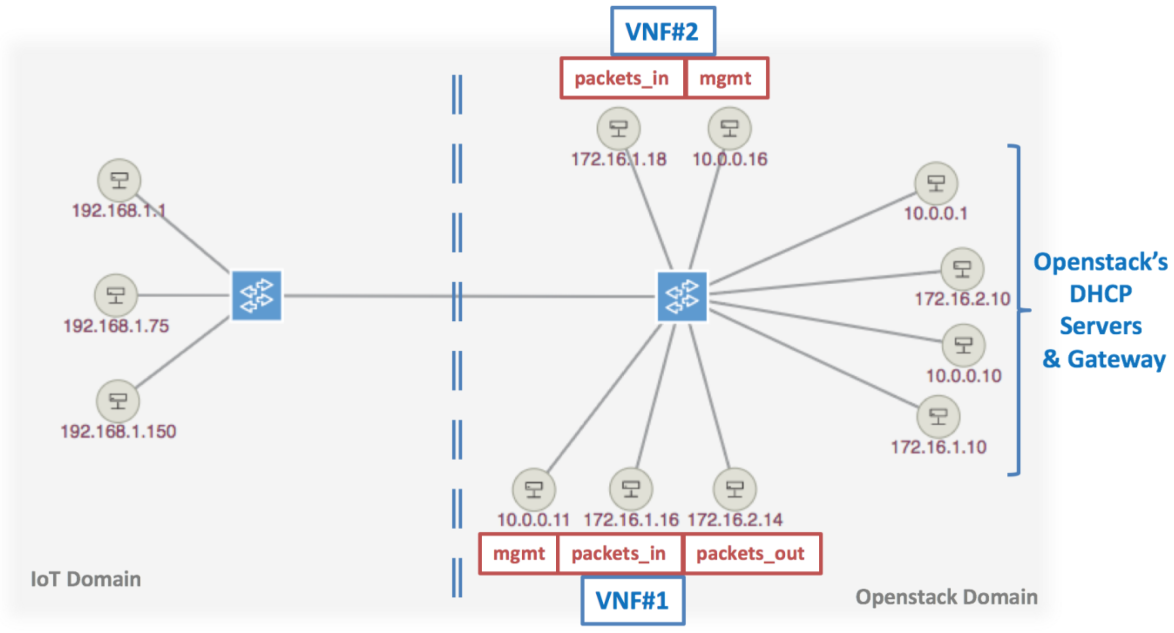
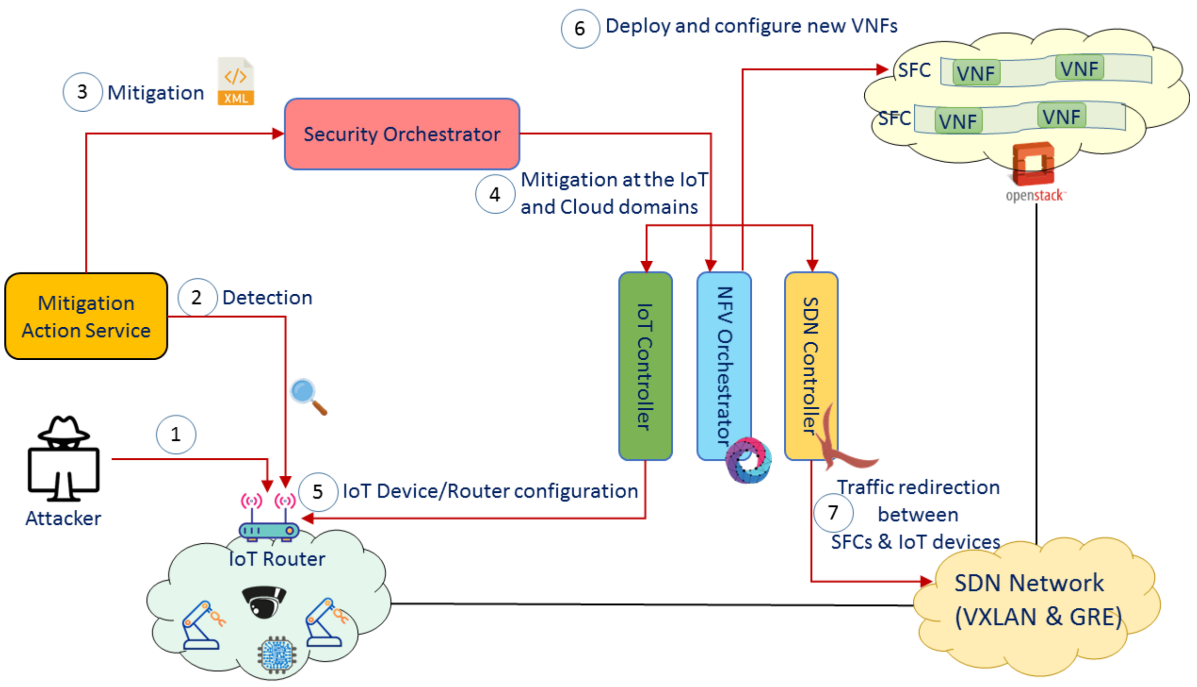


Figure 3 SDN Topology View at Aalto University

The combined usage of these components enables the security orchestrator to enforce the relevant security policies either through direct actions such as: traffic dropping and IoT devices power on/off, or more complex actions when it comes to VNFs:

- Provisioning: Creating the appropriate VNF on a chosen VIM (According to the VNC application graph) such as: Intrusion Detection Systems (IDS) and Firewalls…
- VNF Configuration: Using the MSPL to low level translation, the security orchestrator pushes the specific configuration of each VNF (IDS rules, Firewall configuration...).
- Networking Setup: Injecting the relevant SDN flow rules to manage the traffic to be analyzed, for example: mirroring the traffic to a monitoring agent or steering the traffic through a firewall.
- IoT Security Controls: Enforce IoT security operations through the IoT Controller.



6 Deploy and configure new VNFs
3 Mitigation
XML
Security Orchestrator
SFC VNF VNF
SFC VNF VNF
openstack
4 Mitigation at the IoT and Cloud domains
Mitigation Action Service
2 Detection
IoT Controller
NFV Orchestrator
SDN Controller
1
Attacker
5 IoT Device/Router configuration
7 Traffic redirection between SFCs & IoT devices
IoT Router
SDN Network (VXLAN & GRE)

Figure 4 Security orchestration and enforcement in case of a reactive scenario

Figure 4 depicts the different steps of the orchestration and enforcement plane suggested in ANASTACIA for mitigating different attacks. The attack is detected thanks to the Mitigation Action Service (MAS) component. The latter sends a mitigation request (MSPL file) to the security orchestrator (Figure 4, Step 3). To mitigate the attacks, the security orchestrator interacts with three main actors, which are (Figure 4):

IoT controller: It provides IoT command and control at high-level of abstraction in independent way of the underlying technologies. This component is able to carry out the IoT management request through different IoT constrain protocols like CoAP or MQTT. It also maintains a registry of relevant information of the deployed IoT devices like the IoT device properties and available operations. Since it knows the IoT devices status, it could be able to perform an effective communication to avoid the IoT network saturation when it is required a high-scale command and control operation. It can be found an example and performance of IoT management as part of a building management system [6]. To mitigate different attacks, the security orchestrator interacts with the IoT controller to mitigate the attacks at the level of the IoT domain and prevent the propagation of the attack to other networks (Figure 4: 4). The IoT controller enforce different security rules at the IoT router (data plane) to mitigate the attack (Figure 4: 5).

NFV orchestrator: In ANASTACIA, to ensure efficient management of SFC, we have integrated SDN controller (ONOS) with the used Virtual Infrastructure Manager (VIM), in our case OpenStack [2]. The integration of SDN with the VIM enable the smooth communication between different VNFs that form the same SFC. After receiving the MSPL message from the MAS, the security orchestrator identifies the right mitigation plane should be implemented. If the mitigation plan requires the instantiation of new VNFs, the security orchestrator instructs the NFV orchestrator to instantiate and configure the required VNFs. To instantiate the required VNFs, the NFV orchestrator interacts with the VIM (Figure 4: 6). In addition, the security orchestrator interacts with the policy interpreter to translate the received MSPL to the low configuration (LSPL) needed for different VNFs. After the successful instantiation of a security VNF, the security orchestrator configures that VNF with the received LSPL (Figure 4: 6).

In ANASTACIA, we have also developed different virtual security enablers that should be instantiated to mitigate the different attacks. For instance, we have developed a new VNF firewall based on SDN-enabled switch and OpenFlow. OVS-Firewall is a newly developed solution that relies on OpenFlow protocol to create a sophisticated firewalling system. We have also proposed and developed a new security VNF, named virtual IoT-honeynet, which allows replicating a real IoT environment in a virtual one by simulating the IoT devices with their real deployed firmware, as well as the physical location. The IoT-honeynet can be represented by an IoT-honeynet security policy, and the final configuration can be deployed transparently on demand with the support of the SDN network. "Virtual IoT HoneyNets to mitigate cyberattacks in SDN/NFV-enabled IoT networks" (Under review) shows the potential and performance of this approach.

SDN controller: This component helps in rerouting the traffic between the VNFs in different SFCs. As depicted in Figure 2.4, when the mitigation action service notifies the orchestrator about an attack, the SFC would be updated by adding/inserting new security VNFs in the SFCs. The security orchestrator should push the adequate SDN rules to reroute the traffic between different VNFs in the SFC and the IoT domain (Figure 4: 7). Also, according to the different situations, the security orchestrator can choose the SDN as security enabler. In this case, it can be the attack mitigated by pushing exploring the strength of the SDN technology. If so, the security orchestrator can instruct the SDN controller to push some SDN rules to prevent, allow or limit the communication on specified protocols and ports between different communication peers (Figure 4: 7).

By relying in the aforementioned orchestration properties and features, as well as the SDN and IoT controllers, the ANASTACIA framework aims to cope with the research challenges related with Orchestration of SDN/NFV-based security solutions for IoT environments and currently several experiments have been carried out in different security areas.

For instance, several experiments have been carried out regarding virtual IoT-honeynets. This kind of VNF allows replicating a real IoT environment in a virtual one by simulating the IoT devices with their real deployed firmware, as well as the physical location. The IoT-honeynet can be represented by an IoT-honeynet security policy, and the final configuration can be deployed transparently on demand with the support of the SDN network. "Virtual IoT HoneyNets to mitigate cyberattacks in SDN/NFV-enabled IoT networks" (paper under review) shows the potential and performance of this approach.

Furthermore, the security orchestration of ANASTACIA enables continuous and dynamic management of Authentication, Authorization, Accounting (AAA) as well as Channel Protection virtual security functions in IoT networks enabled with SDN/NFV controllers. The virtual AAA is deployed as VNF dynamically at the edge, to enable scalable device's bootstrapping and managing the access control of IoT devices to the network. Besides, our solution allows distributing dynamically the necessary crypto-keys for IoT M2M communications and deploy virtual Channel-protection proxies as VNFs, with the aim of establishing secure tunnels (e.g. through DTLs) among IoT devices and services, according to the contextual decisions inferred by the cognitive framework. The solution was implemented and evaluated, demonstrating its feasibility to manage dynamically AAA and channel protection in SDN/NFV-enabled IoT scenarios.

A telco cloud environment may consist of multiple VNFs that can be shipped and provided, in the form virtual machine (VM) images, from different vendors. These VNF images will contain highly sensitive data that should not be manipulated by unauthorized users. Moreover, the manipulation of these VNF images by unauthorized users can be a threat that can affect the whole system setup. In ANASTACIA, we have designed and developed different tools to prevent the manipulation of different VNF images should run on top of different network clouds. In ANASTACIA, we have devised efficient methods that verify the integrity of physical machines before using them and also the integrity of virtual machine and virtual network function images before launching them [7] [8]. For this purpose, different technologies have been investigated, such as i) Trusted Platform Module (TPM); ii) Linux Volume Management (LVM); iii)

Linux Unified Key Setup (LUKS). For instance, we have provided a trusted cloud platform that consists of the following components:

- TPM module that is used to store passwords, cryptographic keys, certificates, and other sensitive information. TPM contains platform configuration registers (PCRs) which can be used to store cryptographic hash measurements of the system's critical components. There are in total 24 platform configuration registers (PCRs) in most TPM modules starting from 0 till 23.
- Trusted boot module, which is an open source tool, uses Intel's trusted execution technology (TXT) to perform the measured boot of the system. Trusted boot process starts when trust boot is launched as an executable and measures all the binaries of the system components (i.e., firmware code, BIOS, OS kernel and hypervisor code). Trust boot then writes these hash measurements in TPM's secure storage.
- Remote attestation service, which is the process of verifying the boot time integrity of the remote hosts. It is a software mechanism integrated with TPM, to securely attest the trust state of the remote hosts. It uses boot time measurements of the system components such as BIOS, OS, and hypervisor, and stores the known good configuration of the host machine in its white list database. It then queries the remote host's TPM module to fetch its current PCR measurements. After receiving the current PCR values, it compares them against its white list values to derive the final trust state of the remote host.
- OpenStack Resource Selection Filters component that should be integrated with the nova scheduler. In OpenStack, when a VNF is launched, the nova-scheduler filters pass through each host and select the number of hosts that satisfy the given criteria. Each filter passes the list of selected hosts to the proceeding filter. When the last filter is processed, OpenStack's default filter scheduler performs a weighing mechanism. It assigns weight to each of the selected hosts depending on the RAM, CPU and any other custom criteria to select a host, which is most suitable to launch the VM instance.

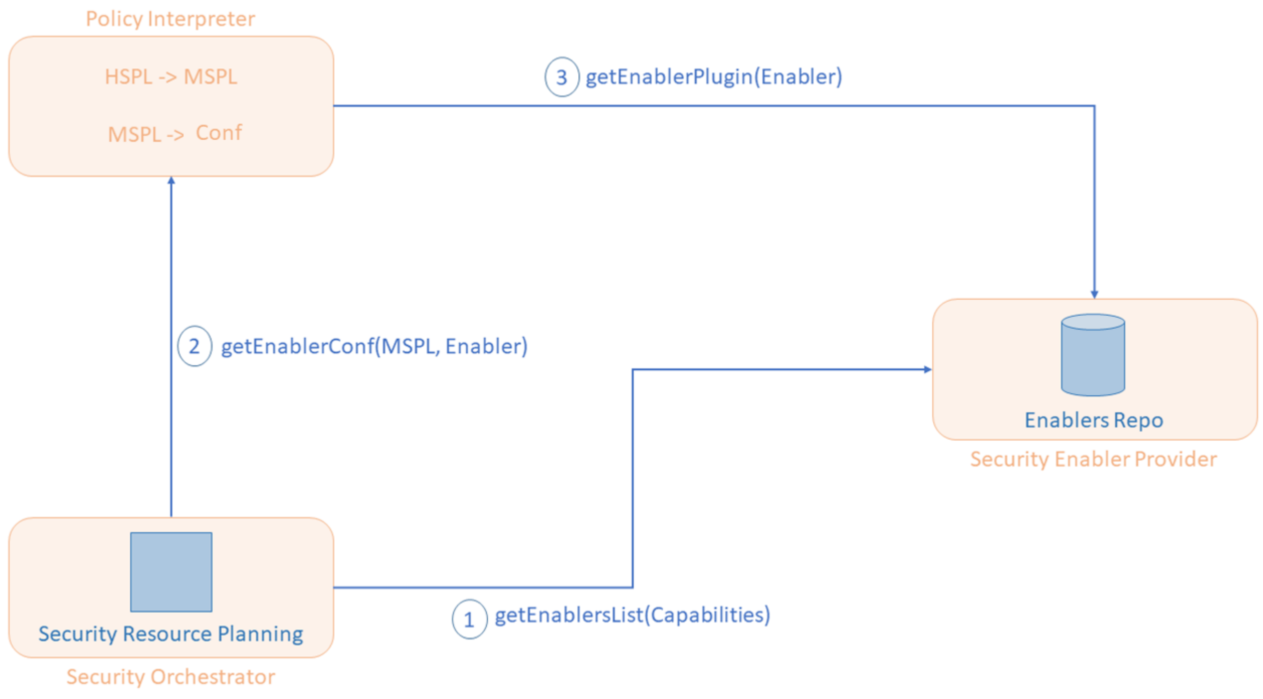
3 Translation Of Security Policies For Enforcement

In this section, we will present the interactions and mechanisms behind the security policy enforcement from the security orchestrator point of view.

3.1 SECURITY ENABLER PROVIDER

The Security Enabler Provider is an external component to the security orchestrator able to identify the list of security enablers in order to provide a specific security capabilities in the Orchestration Plane based on the policy requirements. For example, in case of filtering capabilities the security orchestrator receives the following list (Open vSwitch 'OVS', Snort, IpTable...etc.). Moreover, The Security Enabler Provider provides the corresponding plugin to perform the policy translation after selecting the security enabler by one of subcomponents of the Security Orchestrator named Security Resource Planner.

In contrast to the enabler provider, the security resource planning is an internal component, its main objective is to select the adequate enabler(s) among the list returned by the Security Enabler Provider. This selection done through an Integer Linear Programming (ILP) model.



Policy Interpreter
HSPL -> MSPL
MSPL -> Conf
3 getEnablerPlugin(Enabler)
2 getEnablerConf(MSPL, Enabler)
Enablers Repo
Security Enabler Provider
Security Resource Planning
Security Orchestrator
1 getEnablersList(Capabilities)

Figure 5 Security Enablers Provider Interactions

The figure presents the interaction between the mentioned components; firstly, the security orchestrator requests the list of enablers from the security enabler provider after extracting the capabilities form the received MSPL, and then the Security Resource Planning selects the best services among the list. Once the best enabler identified, the Orchestrator requests the enabler configuration from the Policy interpreter providing the selected enabler. Then, the policy interpreter calls the security enabler to get the plugin from the enablers' repository, and then it returns the adequate enabler configuration to the orchestrator.

3.2 Interactions With Policy Interpreter For Translating Security Policies

This section provides an overview of the interactions between the Policy Interpreter and the Security Orchestrator. A whole description of the interactions and functionality can be found in the deliverable D3.4: Final Security Enforcement Report.

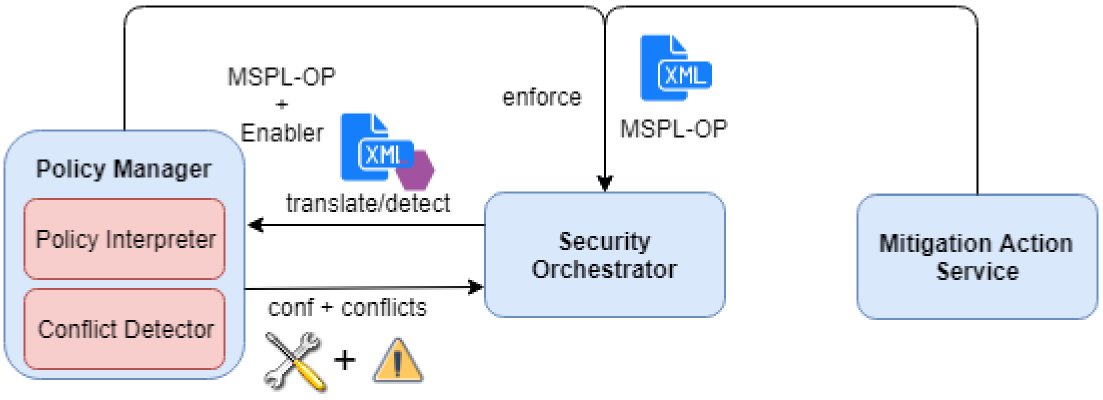


Figure 6: Policy translation overview

Figure 6 shows the main interactions for a policy-based deployment between the Policy Interpreter and the Security Orchestrator. Two approaches are discussed: the proactive approach and the reactive approach. In the proactive approach, the security policy can be part of a preventive measure. In this case, the administrator decides to deploy it with the aim to prevent issues or to establish some default behaviour from start-up, this is, the security administrator defines a High-level policy for orchestration specified in the High-level Security Policy Language (HSPL-OP) by using the Policy Editor Tool. Then it requests the HSPL-OP enforcement to the Policy Interpreter. The Policy Interpreter verifies if the capabilities could be enforced by using any of the available security enablers. If so, it performs an orchestration policy refinement in order to obtain a Medium-level orchestration policy (MSPL-OP). Once the MSPL-OP has been generated, the Policy Interpreter requests the policy enforcement to the Security Orchestrator which analyses the security policies in order to decides the best security enablers for the orchestration policy enforcement. The Security orchestrator then requests the policy translation and conflict detection to the Policy interpreter which obtains the security enablers plugins for the selected security enablers and performs the policy translation as well as the conflict and dependencies detection by taking into account the current status of the system. Once the Security Orchestrator receives the final enablers configurations and the conflicts and dependencies it manages the policies enforcement by taking into account priorities and dependencies, queuing policies with unsolved dependencies and notifying conflicts if required.

Regarding the reactive approach, the workflow is quite similar but in this case there is not HSPL-OP to MSPL-OP refinement process since the MSPL-OP is generated directly for the Mitigation Action Service (MAS) according on the received alerts as well as the desired mitigations, this is, MAS module generates different kind of MSPL-OP in order to provide different kind of mitigations depending on the received alert. Once the Security Orchestrator receives the MSPL-OP, the process continues in the same way that in the proactive use case.

4 Resource and QoS Monitoring

The orchestration system provides an internal component, named "Resource and QoS monitoring", that serves to monitor the resource utilization at the network level in terms of end-to-end delay and bandwidth. This component also serves for monitoring different resources at deployed Virtual Network Functions (VNFs) in terms of CPU, RAM, and storage. The security orchestrator uses this component for ensuring life cycle management (LCM) of deployed services. The collected information enables the security orchestrator to ensure the efficient LCM of VNFs by either deploying or destroying VNFs according to the VNF and network states. This strategy helps for ensuring the KPIs in terms of security, delay, bandwidth, and QoS by mitigating the overload of VNFs and minimizing the cost by deploying the minimum number of VNFs at the right locations. In the literature, many tools have been suggested to monitor the network and the resources at different VNFs.

4.1 Open-Source Resource and Network Monitoring Tools

4.1.1 Nagios Core

Nagios [9] is one of the oldest and best monitoring systems that has been used to monitor the network. The open-source version of Nagios is Nagios Core that can enable different users to get real-time visibility about different hosts and services belonging to the network. This software also offers several alerts that could be used for optimizing and customizing the network and resources at each VNF. Nagios can monitor RAM, CPU and disk loads, and the number of currently running processes at each VNF. Nagios also can monitor services, such as SMTP, POP3, HTTP and other common network protocols used at each VNF.

4.1.2 Icinga

Icinga [10] is a tool that is developed as a branch from Nagios to offer more functionality to the Nagios Core suite. Icinga is designed to be easy to install and use. It offers more functionalities than Nagios for monitoring the network. This tool utilizes text-based configuration files for configuring different parameters. From the server, different agents would be installed at different hosts, and then the server periodically keeps monitoring the resources at each host.

4.1.3 Zabbix

Zabbix is one of the first tools that has been proposed for monitoring the network and services. This tool provides many out-of-the-box features that make it easy to manage and extend. The users do not have to deal with a glut of plugins. It can also offer real-time monitoring metrics across large-scale networks.

4.1.4 Prometheus

Similar to the previous tools, Prometheus is also an open-source system that offers the network and resource monitoring and alerting toolkit. Prometheus has been adopted by many companies and organizations to monitor their networks. Recently, in 2016, Prometheus joined the Cloud Native Computing Foundation. Prometheus has the following features. IT offers a multi-dimensional data model with time series data identified by metric names and key/value pairs. The time series collection happens via a pull model over HTTP. It adopts also a flexible query language, named PromQL, which offers more flexibility for retrieving different resource information. The hosts can be discovered using a service discovery or through manual configuration. Finally, it offers multiple modes of graphing and dashes boarding tools.

4.1.5 Psutil

Psutil is a Python library that provides process and system utilities for monitoring different hosts and networks. Psutil is a cross-platform library for retrieving information on running processes and system utilization (CPU, RAM, disk, network, sensors). It is useful mainly for system monitoring, profiling and limiting process resources and management of running processes. It implements many functionalities offered by UNIX command-line tools including ps, top, lsof, netstat, ifconfig, who, df, kill, free, nice, ionice, iostat, iotop, uptime, pidof, tty, taskset and pmap. It supports many platforms including Linux, Windows, macOS and others.

4.2 ANASTACIA QoS AND RESOURCE MONITORING TOOL

Using the Psutil library and Python language, we have developed the ANASTACIA QoS and resource monitor tool that monitors the resources at different VNFs and the delay and bandwidth of the network. To take the right decisions at the security orchestration plane, this component is leveraged for collecting the information about the resources of different deployed VNFs and the network in terms of end-to-end delay and bandwidth. When a VNF is instantiated, the security orchestrator injects an ANASTACIA QoS and resource monitor daemon into that instance, this daemon keeps sending the resource usage to the security orchestrator. The monitoring daemon keeps collecting in real-time the resources usage including the CPU usage percentage, memory details available, used and total. The monitoring agent also provides information about the network in terms of delay and bandwidth, as well as the security levels of different paths. We plan in the future to explore the ONOS API that exposes REST API for the net metering and JAVA API for QoS providing (Type, rate, and size).

5 Security Optimizer

In ANASTACIA framework, the nature and the size of the SFCs would be defined according to the nature of the user (a normal or a suspicious). Whenever an attack is detected, new VNF security could be inserted into to the SFC in order to protect both the legitimate IoT devices and users. The security orchestrator optimizer (SO) is the main brain that is responsible for defining different policies to ensure the life cycle management (LCM) of the SFCs by managing different VNFs and PNFs. Also, it has the responsibility to reuse as much as possible the existing VNFs and PNFs in order to reduce the cost while: i) preventing the conflicts between the different security policies already enforced in at different VNFs and PNFs; ii) ensuring the required KPIs in terms of QoS, delay and bandwidth. For instance, a SFC should not use a security VNF that could create a bottleneck on that SFC. For this reason, the quality of the links between the VNFs in the same SFC should be considered.

Figure 7 shows the interaction of the security orchestration system with other components to ensure the security and achieve the desired KPIs. To mitigate the various attacks without affecting the QoS in different verticals, the SO plane takes into account the policies requirements and the available resources in the underlying infrastructure. The latter refers to the available amount of resources in terms of CPU, RAM, and storage in different cloud providers, as well as the communication network resources including the end to end bandwidth, jitters and delay. The quality of the link varies according to the locations of different communicating peers (i.e., intra and inter cloud communication), as well as the use of secure channels with different levels including IPsec, SSL and TLS. The Security Orchestrator Engine (SOE) will interact with the network due to reasons as depicted in figure 3: *i)* A possible attack is detected by the mitigation action service (MAS); ii) A violation in the KPIs (delay and/or bandwidth) at a specified SFC.

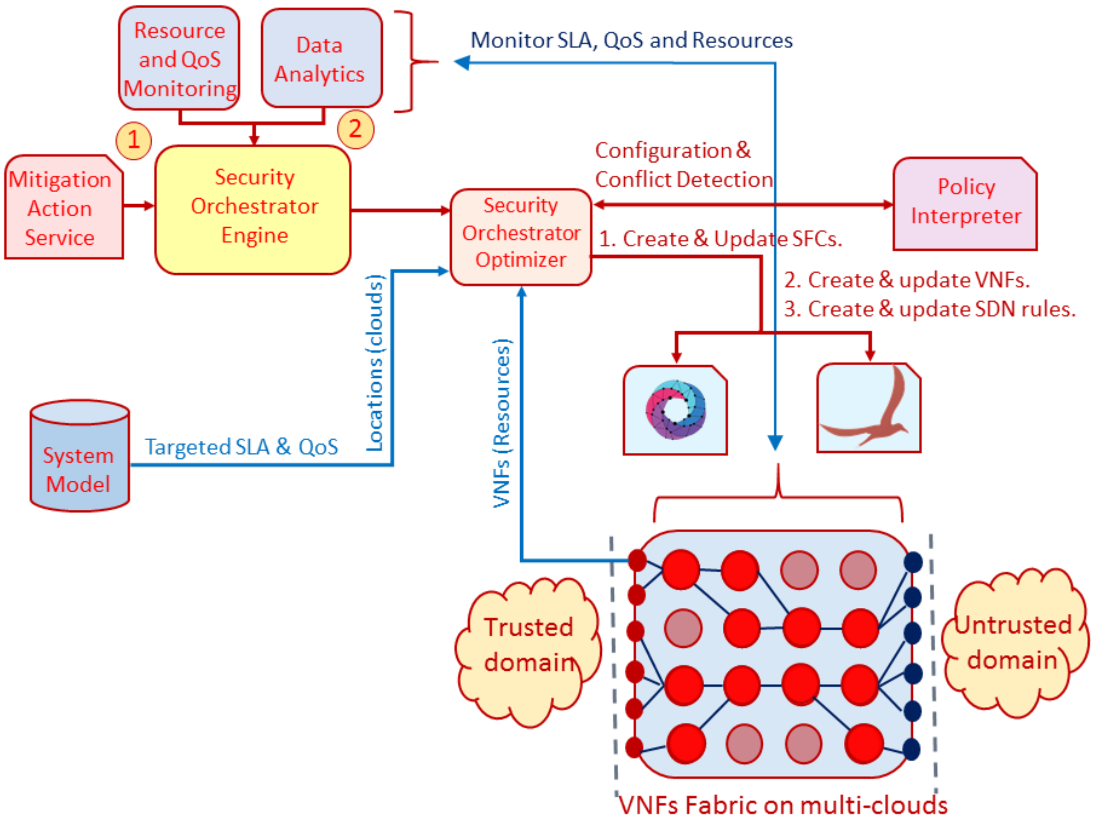
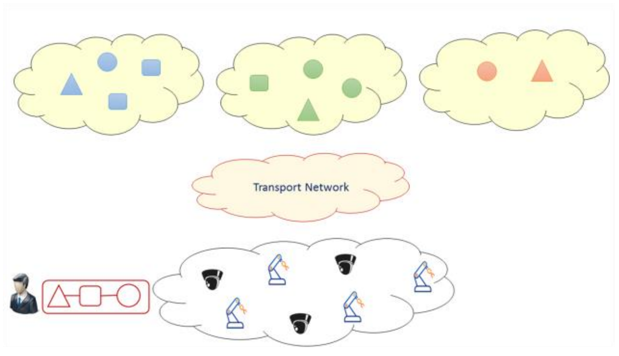

Resource and QoS Monitoring
Data Analytics
Monitor SLA, QoS and Resources
1
2
Mitigation Action Service
Security Orchestrator Engine
Configuration & Conflict Detection
Policy Interpreter
Security Orchestrator Optimizer
1. Create & Update SFCs.
2. Create & update VNFs.
3. Create & update SDN rules.
Locations (clouds)
VNFs (Resources)
System Model
Targeted SLA & QoS
Trusted domain
Untrusted domain
VNFs Fabric on multi-clouds

Figure 7 Communication between the security orchestrator optimizer and the other components

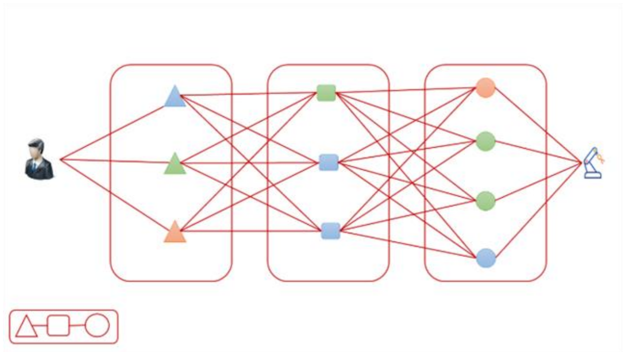
As depicted in Figure 7, when the MAS detects an attack, it sends a mitigation request (Orchestration MSPL file) to the SOE (Figure 7, Step 3). Then, the SOE will get the state of the network from the resource and QoS monitoring (RQM) agent. The latter could communicate with data analytics (DA) component to predict the amount of resources utilization, in terms of CPU, RAM and disk, would be expected at each VNF, PNF and infrastructure, as well as the quality of the links between these network components. The DA uses historical data and machine technique (regression method) to predict the future resource utilization. The SOE will forward these information and the target SLA and QoS to the SOO. The latter will communicate with the policy interpreter to detect the conflict in policies could happen between the new request and already deployed policies. The policy conflict detector in the policy interpreter will inform the SOO about the possible conflicts by taking into account the current status of the system, e.g., already deployed security policies as well as VNFs or PNFs. In order to reduce the cost, the SOO will try to use as much as possible already deployed VNFs that do not create any conflicts with already deployed policies. The SOO uses linear integer programing optimization to select these VNFs that ensure the SLA and QoS. In worst case scenario, if no one of already deployed VNFs meet the above requirements, the SOO will decide to create a set of VNFs in the right locations and using the right amount of resources. After, the decisions are taken, the SOO will communicate through SOE with different security enables, such as SDN controller and NFV orchestrator, to enforce the take decisions.

Based on the observation that the amount of traffic generation could change during the communication, we have designed the security orchestrator to ensure the life cycle management (LCM) of already deployed VNFs to prevent the violation in the KPIs. For this reason, the RQM agent keeps track of the network if all the KPIs are preserved. Whenever a violation in the KPIs is detected, the RMQ agent informs the SOE. The latter will redeploy the SFCs that have the issues in different locations. Moreover, the RMQ could prevent the violation of the KPIs a priori by involving DA component. According to the state of the network a possible violation can be detected and mitigated by the SOO.

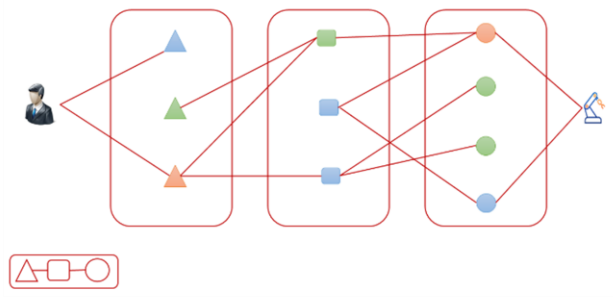
As we have explained above, a user would access to the IoT device through a specified SFC that consists of a set of VNFs and PNFs, respectively. For instance, a user can access to the IoT devices using an SFC that consists of: *i)* A load balancer (a triangle in Figure 8 (a)); *ii)* a firewall that is presented as rectangle in Figure 8 (a); *iii)* Finally, an MQTT broker whereby the IoT device publish its message. The communication, between different clouds, VNFs, users and devices, is characterized by different delay, bandwidth, jitter and security levels. For instance, the communication link within the same cloud has a high security level while the one with between users, devices and inter clouds without any security mechanisms, such as IPSEC and SSL, has the lowest security level. Each SFC is created from a predefined blueprint that specifies its characteristics and targeted KPIs that should be respected in terms of link security level, QoS, delay and bandwidth. According to the KPIs defined in the SFC blueprint, the VNFs and communication links should be used to satisfy the predefined requirements. Figure 8 (b) depicts a rearrangement of candidate VNFs, in different clouds, and their possible links between them to fulfill the requested SFC.



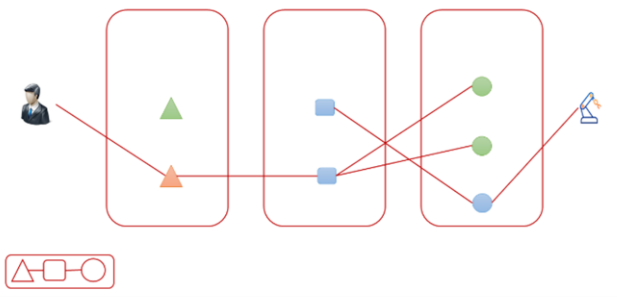
(a) Network topology



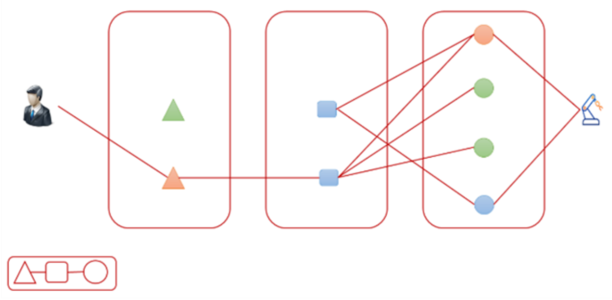
(b) All possible SFCs between the user and the IoT device



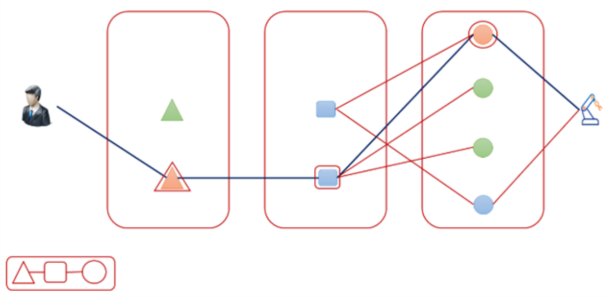
(c) Removing links that do not respect the KPIs



(d) Removing VNFs that do not have enough resources



(e) Creating a new VFN at the third cloud to generate a valid SFC

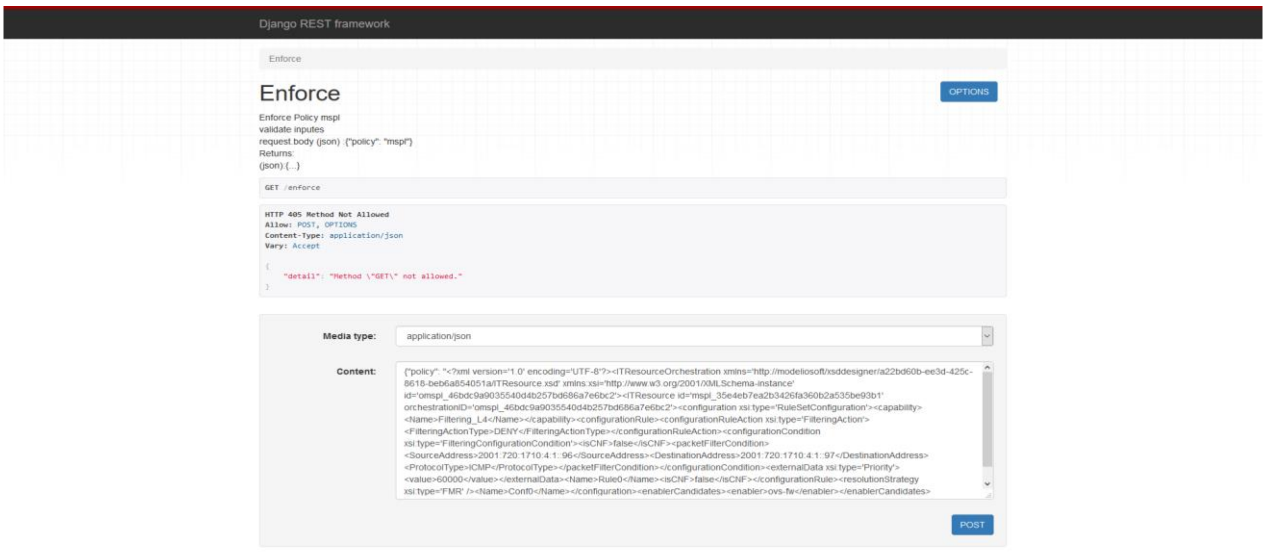


(f) Generating the final SFC between the user and the IoT

Figure 8 Main idea of the security optimizer component

As we have explained before, some links could not satisfy the KPIs specified in the SFC blueprint, and hence they should be considered when instantiating the SFC as shown in Figure 8 (c). As depicted in figure 1, the information about the links, such as bandwidth, jitters and delay, and the resources of already deployed VNFs in terms of CPU, RAM and disk would be retrieved from the module "Resource and QoS Monitoring". The latter would give predict values about different metrics by exploring the "Data Analytics" module. Using the latter module, we can predict the resources should be consumed from each VNFs if it is included in the SFC. Thus, each VNF that would use resources higher than a predefined threshold should be not considered as depicted in Figure 8 (d). However, we could have a situation whereby some VNFs are missing to complete the SFC as depicted in Figure 8 (d). In this case, the security orchestrator needs to instantiate a new VNF with the required resources and at the appropriate cloud (i.e., Links that ensure the KPIs) as depicted in Figure 8 (e). Then, the SFC would be instantiated using both NFV orchestrator and SDN controller as depicted in Figure

8 (f). Note that in a real scenario, we can receive many requests to create different SFCs with different size and KPIs. Thus, the security optimizer component is designed to consider all the requests once to prevent to generate sub-optimal configuration (i.e., local optimal situation). Figure 11 shows the rest API provided by the security orchestrator to communicate with the other ANASTACIA's components, such as MAS. This rest API enables the orchestrator to receive the Orchestration MSPL file for enforcing different security policies.



Enforce

Enforce Policy mspl
validate inputes
request body (json) :{"policy": "mspl"}
Returns:
(json):{...}

```
GET /enforce
```

```
HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "detail": "Method \"GET\" not allowed."
}
```

Media type: application/json

Content:

{"policy": "<?xml version='1.0' encoding='UTF-8'?><ITResourceOrchestration xmlns='http://modeliosoft/xsddesigner/a22bd60b-ee3d-425c-8618-beb6a854051a/ITResource.xsd' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' id='omspl_46bdc9a9035540d4b257bd686a7e6bc2'><ITResource id='mspl_35e4eb7ea2b3426fa360b2a535be93b1' orchestrationID='omspl_46bdc9a9035540d4b257bd686a7e6bc2'><configuration xsi:type='RuleSetConfiguration'><capability><Name>Filtering_L4</Name></capability><configurationRule><configurationRuleAction xsi:type='FilteringAction'><FilteringActionType>DENY</FilteringActionType></configurationRuleAction><configurationCondition xsi:type='FilteringConfigurationCondition'><isCNF>false</isCNF><packetFilterCondition><SourceAddress>2001:720:1710:4:1::96</SourceAddress><DestinationAddress>2001:720:1710:4:1::97</DestinationAddress><ProtocolType>ICMP</ProtocolType></packetFilterCondition></configurationCondition><externalData xsi:type='Priority'><value>60000</value></externalData><Name>Rule0</Name><isCNF>false</isCNF></configurationRule><resolutionStrategy xsi:type='FMR' /><Name>Conf0</Name></configuration><enablerCandidates><enabler>ovs-fw</enabler></enablerCandidates>

Figure 9 Orchestrator policy enforce API interface

6 System Model

The System Model is an independent component in charge of storing data relevant to the different components. This data is made available to all of the ANASTACIA components in order to further refine/translate the security policies and improve the detection mechanisms. Depending on the type of the desired security policy, the System Model stores the relevant VNF details, SDN rules, IoT actions and related policies.

6.1 Architecture

As the Security Orchestrator expects to receive an Orchestration MSPL file, each request is mapped with a unique ID then inserted to the system model with a pending status, the policy interpreter service can use the ID to track his request. Once the task finishes the policy enforcement process, we update its status on the SM to either 'success' or 'failed'.

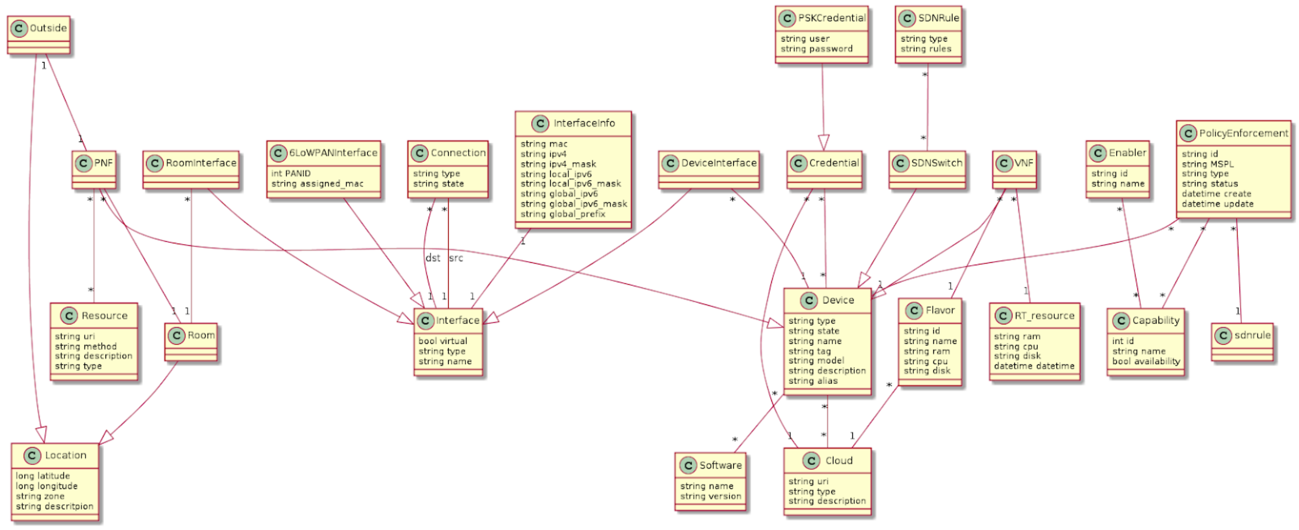


Figure 10 System Model Data Structure

Cloud: This table stores the framework clouds details.

- uri: The cloud Uniform Resource Identifier.
- type: a string field specify the cloud type (Openstack, ..).
- Description: a string field for more details about the cloud.

Software: Related to device using foreign key constraint storing each device software.

- name: A string field storing the software name.
- version: A string field the version of the software.

Resource: Iot device resource table.

- uri: The resource Uniform Resource Identifier.
- method: A string field http method
- port: remote port.
- description: A string field for more information.

- type: A string field providing the type of resource.

Flavor: Stores the cloud flavors used for each VNF Instance.

- id: A string field store the cloud generated id.
- name: A string storing the name of flavor
- ram: The memory size specified in the flavor.
- cpu: The compute capacity specified in the flavor.
- disk: The storage amount defined in the flavor.

RT_resource: storing the Real time resource of VNFs. This table is used by the VNFs daemons.

- ram: The used memory percentage.
- cpu: the used CPU percentage.
- disk: The used storage percentage.
- datetime: Time of the update.

Interface: Generic table for interface types.

- virtual: Boolean field true if the interface is virtual.
- name: the name of the interface.

DeviceInterface: Inherits from Interface and relates to device using foreign key constraint.

RoomInterface: Inherits from Interface and relates to room using foreign key constraint.

InterfaceInfo: This table stores details of the interface.

- mac: the physical address of the interface.
- ipv4: The IP version 4 of the interface.
- ipv4_mask: The mask of the ipv4 address.
- local_ipv6: the local IP address version 6.
- local_ipv6_mask: The mask of the local ipv6.
- global_ipv6: the Global IP address version 6.
- global_ipv6_mask: The mask of the global ipv6.
- global_prefix: the global prefix.

Connection: Stores the connections exists in the framework between source and destination interface.

- type: A string field contains the type of connection.
- state: A choice field (UP, DOWN).

6LoWPANInterface : IPv6 over Low interface Inherits from Interface table.

- PANID : A string field stores the PAN ID.
- assigned_mac: A string field mac address.

Device: Provide abstraction for all different kinds of devices (VNF, PNF ..)

- type : Specify the type of a device from predefined list (IoT, FW)
- state: A choice field (UP, DOWN).

- name: A string field contains the name of the Device.
- tag: A sting field storing tag.
- model: A string field contains model
- description: A string Field contains other details related to the device.
- alias: A string field for alias

PNF: Inherits from Device contains all the information of the physical network function and IoT   devices can be related either to a room or outside location.

VNF: Inherits from Device contains all the information related to the virtual network function.

SDNSwitch: Inherits from Device contains all the information of SDN switch devices.

SDNRule: In this table stores the SDN rules applied in specified SDN switches.

- type: A string field contains the type of the rule.
- rules: A string contains the rule in text format.

Location: Abstraction of different locations used in the system.

- latitude: A string field stores the latitude.
- longitude: A string field stores the longitude.
- zone: A string field stores the zone.
- Description: : A string Field contains other details related to the location.

Outside: Since a physical device could be located in specified room or somewhere else outside the room, this table stores the location of devices outside stored rooms.

Room: Inherits from Location contains we need this to be related to the room.

Capability: stores capabilities.

- id: A string field stores a fix id providing in the creation.
- name: A string field contains the name of the capability.
- availability: A boolean field indicate the availability.

PolicyEnforcement: Stores the status of the policy enforcement process

- Id: string field stores the Policy enforcement unique id
- MSPL: String field stores the MSPL text.
- type: String field stores the enforcement type.
- status: choice field store the status of the enforcement.
- create: a datetime field stores the creation date and time.
- update: a datetime field stores the last update date and time.

Enabler: Stores enablers.

- id: A string field stores the enabler ID
- name: A string field contains the name of the enabler

Credential:

PSKCredential:

- user: a string field stores the username.
- password: a string field stores the password

Filters :

- to retrieve a device from it's IP (IPv4 or IPv6)
- filter by the room name parameter in the room_list
- specific fields as the IoT device name

6.2 COMMUNICATION THROUGH API

6.2.1 Authentication:

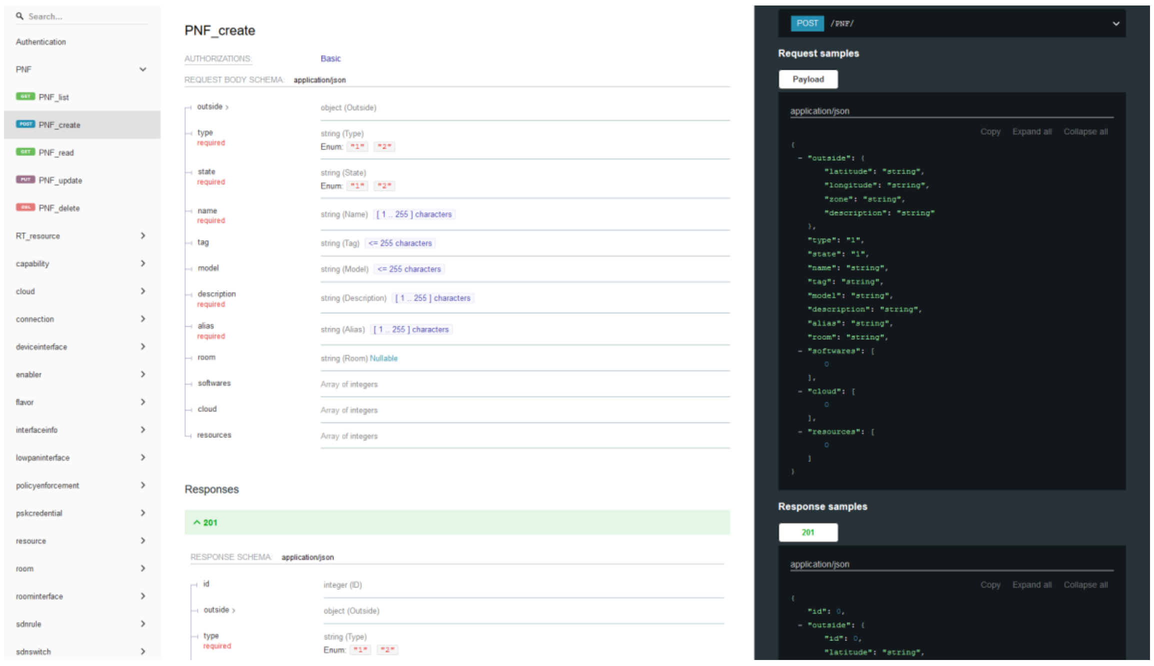
The system model provides use a simple authentication method: The basic access authentication.
Example using Curl :
curl -X GET "http://system-model-domain/api/vnf/" -H "accept: application/json" --user username:password

6.2.2 The API endpoints:

- "software": "http://system-model-domain/api/software/",
- "resource": "http://system-model-domain/api/resource/",
- "cloud": "http://system-model-domain/api/cloud/",
- "deviceinterface": "http://system-model-domain/api/deviceinterface/",
- "roominterface": "http://system-model-domain/api/roominterface/",
- "interfaceinfo": "http://system-model-domain/api/interfaceinfo/",
- "connection": "http://system-model-domain/api/connection/",
- "lowpaninterface": "http://system-model-domain/api/lowpaninterface/",
- "sdnrule": "http://system-model-domain/api/sdnrule/",
- "sdnswitch": "http://system-model-domain/api/sdnswitch/",
- "vnf": "http://system-model-domain/api/vnf/",
- "PNF": "http://system-model-domain/api/PNF/",
- "room": "http://system-model-domain/api/room/",
- "policyenforcement": "http://system-model-domain/api/policyenforcement/",
- "capability": "http://system-model-domain/api/capability/",
- "enabler": "http://system-model-domain/api/enabler/",
- "RT_resource": "http://system-model-domain/api/RT_resource/",
- "flavor": "http://system-model-domain/api/flavor/",
- "pskcredential": "http://system-model-domain/api/pskcredential/"

Swagger doc:

- "swagger doc": "http://system-model-domain/redoc"
- "swagger-ui": "http://system-model-domain/swagger/"



PNF_create

AUTHORIZATIONS: Basic

REQUEST BODY SCHEMA: application/json

outside ›	object (Outside)
type required	string (Type) Enum: "1" "2"
state required	string (State) Enum: "1" "2"
name required	string (Name) [1 .. 255] characters
tag	string (Tag) <= 255 characters
model	string (Model) <= 255 characters
description required	string (Description) [1 .. 255] characters
alias required	string (Alias) [1 .. 255] characters
room	string (Room) Nullable
softwares	Array of integers
cloud	Array of integers
resources	Array of integers

Responses

^ 201

RESPONSE SCHEMA: application/json

id	integer (ID)
outside ›	object (Outside)
type required	string (Type) Enum: "1" "2"

POST /PNF/

Request samples

Payload

application/json

```
{
  - "outside": {
        "latitude": "string",
        "longitude": "string",
        "zone": "string",
        "description": "string"
    },
    "type": "1",
    "state": "1",
    "name": "string",
    "tag": "string",
    "model": "string",
    "description": "string",
    "alias": "string",
    "room": "string",
  - "softwares": [
        0
    ],
  - "cloud": [
        0
    ],
  - "resources": [
        0
    ]
}
```

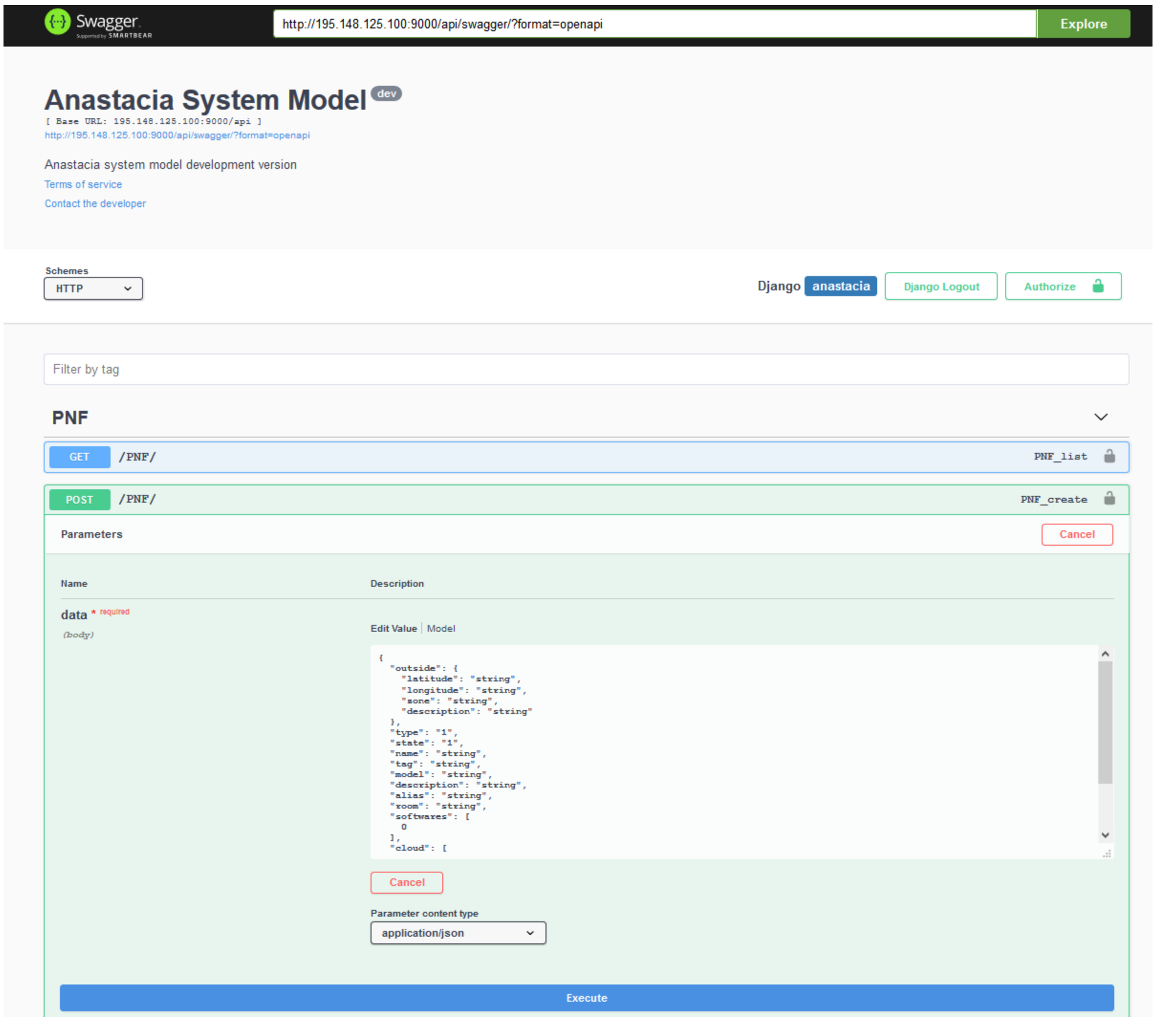
Response samples

201

application/json

```
{
    "id": 0,
  - "outside": {
        "id": 0,
        "latitude": "string",
```

Figure 11 System model swagger documentation



http://195.148.125.100:9000/api/swagger/?format=openapi

Anastacia System Model dev

[Base URL: 195.148.125.100:9000/api]

http://195.148.125.100:9000/api/swagger/?format=openapi

Anastacia system model development version

Terms of service

Contact the developer

Schemes HTTP

Django anastacia Django Logout Authorize

Filter by tag

PNF

GET /PNF/ PNF_list

POST /PNF/ PNF_create

Parameters Cancel

Name	Description
data * required (body)	Edit Value | Model

```
{
  "outside": {
    "latitude": "string",
    "longitude": "string",
    "zone": "string",
    "description": "string"
  },
  "type": "1",
  "state": "1",
  "name": "string",
  "tag": "string",
  "model": "string",
  "description": "string",
  "alias": "string",
  "room": "string",
  "softwares": [
    0
  ],
  "cloud": [
```

Cancel

Parameter content type: application/json

Execute

Figure 12 System model swagger interface

7 CONCLUSIONS

This document provides an overview of the Security Orchestrator component that we have designed and developed within the ANASTACIA framework. It thoroughly describes the inner core functionalities of the orchestrator and its different interactions with other components. We have given a brief description on the developed components, such as security resource planning, policy interpreter. Moreover, we have described the new proposed components that include: *i)* Security orchestrator optimizer (SOO); *ii)* Resource and QoS monitoring component (RQM); *iii)* Data analytics (DA) component. Also, in this deliverables, we have discussed about different 5G enablers that include SDN, NFV and IoT orchestration strategies, which are supported by the security orchestrator. Finally, we discussed the technical tools that allow the Security Orchestrator to provide an on-demand optimized security policy enforcement service called Security as a Service (SECaaS) for the ANASTACIA framework.

8 References

[1] I. F. e. al., “Towards Provisioning of SDN/NFV-based Security,” in *IEEE Conference on Standards for Communications and Networking (CSCN)*, Helsinki, 2017.

[2] M. B. D. L. C. D. T. T. a. N. T. Y. Khettab, “Virtual security as a service for 5G verticals,” in *IEEE Wireless Communications and Networking Conference (WCNC)*, Barcelona, 2018.

[3] [Online]. Available: https://osm.etsi.org.

[4] [Online]. Available: https://osm.etsi.org/wikipub/index.php/Creating_your_VNF_Charm.

[5] [Online]. Available: https://onosproject.org/.

[6] A. M. Z. e. al., “Security Management Architecture for NFV/SDN-Aware IoT Systems,” *IEEE Internet of Things Journal,* vol. 6, no. 5, pp. 8005-8020, Oct. 2019.

[7] T. T. a. A. D. S. Lal, “NFV: Security Threats and Best Practices,” *IEEE Communications Magazine,* vol. 55, no. 8, pp. 211-217, Aug. 2017.

[8] A. K. I. O. K. A. a. T. T. S. Lal, “Securing VNF communication in NFVI,” in *IEEE Conference on Standards for Communications and Networking (CSCN)*, Helsinki, 2017.

[9] [Online]. Available: https://www.nagios.org/projects/nagios-core/.

[10] [Online]. Available: https://github.com/Icinga/icinga2.