ANASTACIA

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

D3.2

Initial Security Orchestrator Report

This deliverable presents the first results 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	30.06.2018 [M18]
Delivery date	02.07.2018 [M19]
WP / Task	WP3 / T3.2
WP Leader	AALTO
Authors	T. Taleb, Y. Khettab, A. Laghrissi (AALTO), D. Rivera (MONT), R.Marín Pérez (ODINS), J. Bernal, A. Molina (UMU), D.Belabed (THALES), A.Mady, P.Sobonski, D. Mehta (UTRC),
EC Project Officer	Carmen Ifrim carmen.ifrim@ec.europa.eu
Project Coordinator	SoftecoSismatSpA 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 programmeunder 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 first results 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. It plays a major role in the ANASTACIA architecture accounting for it interactions with other components of the framework [1].

IoT devices are prone to various security attacks varying from Denial of Service (DoS) to Man-in-the-Middle attacks which are hindering their wide adoption. In this vein, SDN and NFV represent key technologies towards a novel concept of on-demand security countermeasures provisioning based on programmability and advanced virtualization technologies.

The Security Orchestrator is responsible for providing on-demand security policy enforcement on the IoT domain. This task is performed by taking in charge the transformation of the relevant security policies provided by the security policy interpreter into specific enabler configuration. It also monitors and supervises the underlying infrastructure for any potential flaws. It supports a variety of security capabilities of different categories, namely: SDN security capabilities, NFV security capabilities and IoT security controls.

In first report, we will present the different aspects of the Security Orchestrator within the ANASTACIA framework relevant to its technologies, strategies, interactions with other components and its major role in the use cases integration.

1 INTRODUCTION

This section will introduce this document by enumerating its aims, references, revision history and the different acronyms that were used.

1.1 AIMS OF THE DOCUMENT

This document is the second WP3 "Policy Enforcement and Run Time Enablers" report which focuses on the development of core enablers for the deployment and implementation phase, including the security enforcement manager, the security orchestrator, the virtual resources manager and the autonomic prediction and reconfiguration enabler.

More precisely, this deliverable tackles the Task 3.2 within WP3. This task is about the Security Orchestrator, which is a centric component of the ANASTACIA architecture. It ensures to deploy the necessary security policies either as a reaction to a detected attack or as proactive measures set by the users of the framework. To this aim, it employs an intelligent orchestration system that makes use of SDN, NFV and IoT controls to mitigate the security flaws. Once a certain security policy has been deployed, it stores relevant information about it and make it available for other components of the framework.

This document is structured 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 explain the core functionalities of the Security Orchestrator, including its main technologies, security capabilities, interactions and what it offers to the rest of the framework. In section 4, we will identify the main role of the Security Orchestrator in showcasing the ANASTACIA framework thanks to the pre-defined use cases. Finally, in section 5, we will further detail the implementation of the Security Orchestrator that was used in many of our Proof-of-Concept demos shown in the ANASTACIA plenary meetings.

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-centered Requirement Initial Analysis
- ANASTACIA deliverable D2.1 – Policy-based Definition and Policy for Orchestration, initial report
- ANASTACIA deliverable D3.1 - Initial Security Enforcement Manager Report

1.3 Revision History

Version	Date	Author	Description
0.1	7-05-2018	Yacine Khettab	Initial document structure and contributions
0.2	14-05-2018	Piotr Sobonski	Initial UTRC contributions
0.3	16-05-2018	Yacine Khettab	Content contribution for chapter 2-5
0.4	18-05-2018	Yacine Khettab	Initial D3.2 draft
0.5	27-06-2018	Laghrissi Abdelquoddouss	Final version

1.4 Acronyms and Definitions

Acronym	Meaning
MSPL	Medium-level Security Policy Language
HSPL	High-level Security Policy Language
IoT	Internet of Things
MANO	Management and Orchestration
NFV	Network Function Virtualization
SDN	Software Defined Networking
MEC	Mobile Edge Computing
OSM	Open Source Mano
ONOS	Open Network Operating System
M2L	Medium to Low
OVS	Open Virtual Switch
SO	Security Orchestrator
DPI	Deep Packet Inspection
IDS	Intrusion Detection System
SEP	Security Enforcement Plane

2 REFINEMENT OF SECURITY POLICIES FOR ENFORCEMENT

In the section, we will present the different interactions and mechanisms behind the security policy enforcement from the Security Orchestrator point of view.

2.1 INTERACTIONS WITH POLICY INTERPRETER FOR REFINING SECURITY POLICIES

This section 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 behavior from start-up. On the other hand, the reactive approach allows deploying a security policy as part of an automatic countermeasure.

2.1.1 Proactive approach

Figure 1 shows the main interactions between the Policy Interpreter and the Security Orchestrator for a security policy deployment in a proactive approach. The Policy Editor Tool is shown just in order to provide the starting point of the workflow. In this case, the following steps compound the process:

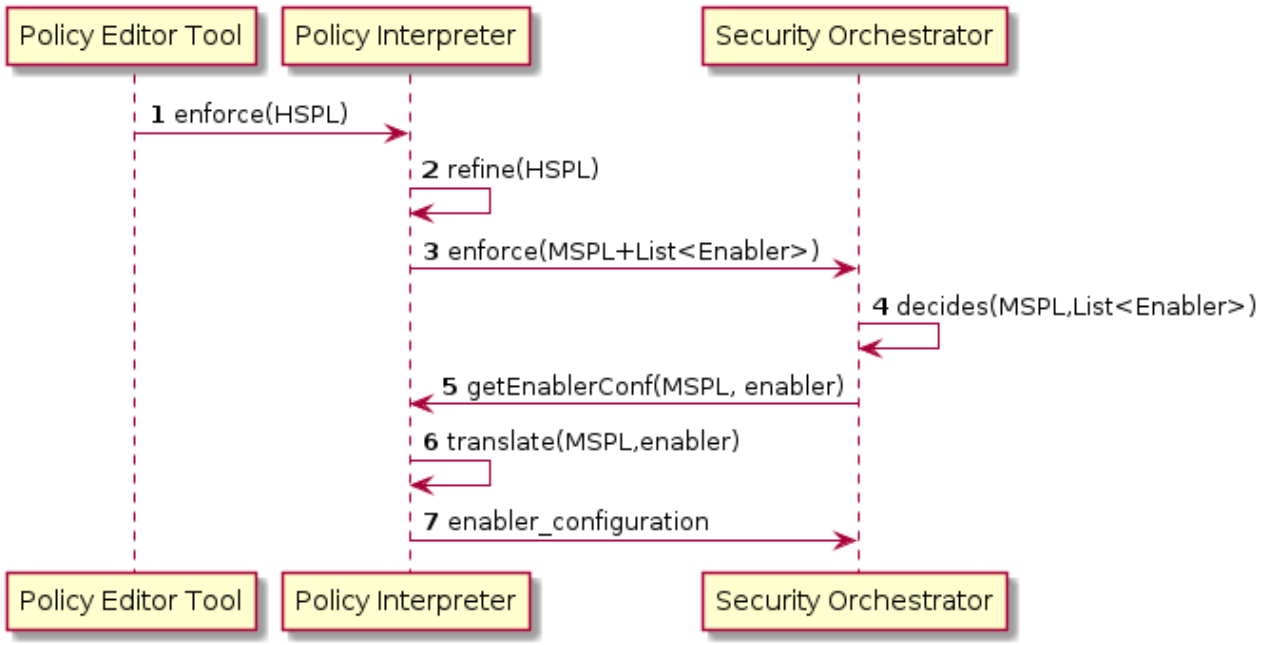

Figure 1 Main interactions between the Policy Interpreter and the Security Orchestrator in proactive approach

1. The system administrator defines a high-level policy (HSPL) through the Policy Editor Tool and he/she requests the policy enforcement.
2. The Policy Interpreter performs the High-level Security Policy Language (HSPL) to Medium-level Security Policy Language (MSPL) refinement process. This process has been simplified in the diagram, but it also interacts with the Security Enabler Provider in order to obtain a list of candidate security enablers according on the capabilities.
3. The Policy Interpreter sends the MSPL and the list of candidate security enablers to the Security Orchestrator.

4. The Security Orchestrator makes the decision regarding which Security Enabler must be used in order to enforce the security policy.
5. The Security Orchestrator requests a policy translation to the Policy Interpreter in order to get the configuration for the selected security enabler from the MSPL policy.
6. The Policy Interpreter performs the translation, using a plugin received in another interaction with the Security Enabler Provider (not shown in the figure for simplicity).
7. The Policy Interpreter sends the security enabler configuration to the Security Orchestrator.

2.1.2 Reactive approach

Figure 2 shows the main interactions between the Policy Interpreter and the Security Orchestrator for a security policy deployment in a reactive approach. The Reaction process is showed just in order to provide the starting point of the workflow. In this case, the process is compounded by the following steps:

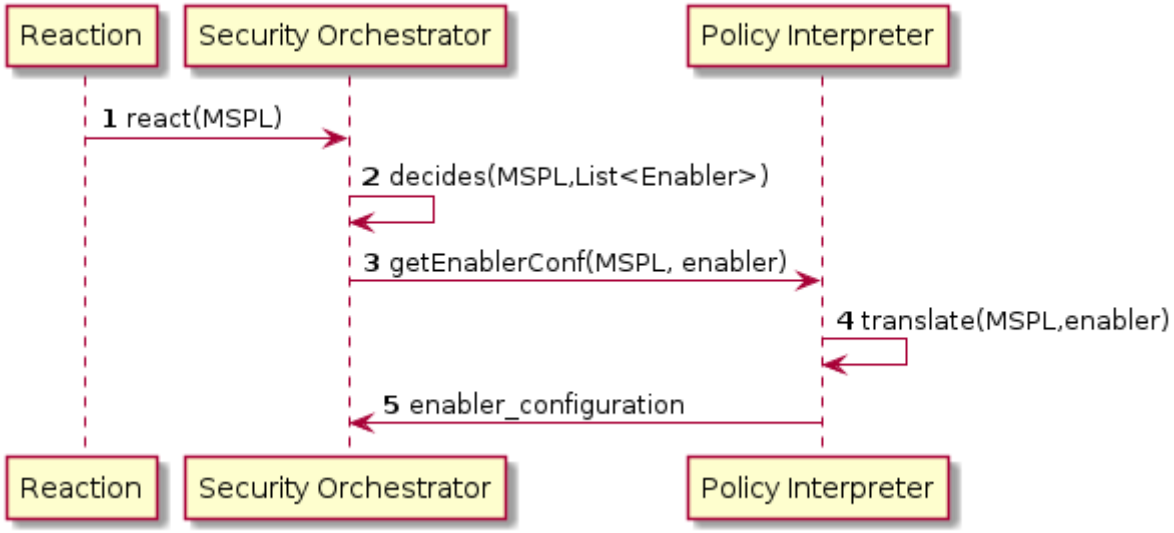

Figure 2 main interactions between the Policy Interpreter and the Security Orchestrator in reactive approach

1. The reaction module sends a MSPL policy as a reaction countermeasure.
2. The Security Orchestrator makes the decision regarding which Security Enabler must be used in order to enforce the security policy. The list of candidate security enabler has been obtained previously in an interaction with the Security Enabler Provider (not shown in the figure for simplicity).
3. The Security Orchestrator requests a policy translation to the Policy Interpreter in order to get the configuration for the selected security enabler from the MSPL policy.
4. The Policy Interpreter performs the translation using a plugin received in another interaction with the Security Enabler Provider (not shown in the figure for simplicity).
5. The Policy Interpreter sends the security enabler configuration to the Security Orchestrator.

2.2 Interactions with Security Enabler Provider for selecting the appropriate security enablers

One of the roles of the Security Enabler Provider in the Security Orchestration Plane is to identify the security enablers that can provide specific security capabilities, in order to meet the security policies requirements. When the security orchestrator component receives the MSPL file from the reaction module, it requests the enablers list for the identified capabilities from Security Enabler Provider. The component selects the list of the enablers that fit with the sent reaction capability, as an example, for filtering capability the enablers can be “Open vSwitch (OVS), Snort, IpTable, etc”. This selection is done by requesting the “Security Enabler Repository” then the list of the suitable enablers is selected and sends to the Security Orchestrator where a sub-module in the Security Orchestrator called the Resource Planner will select the adequate enabler among the received enablers list. This planner has been implemented using python language, Falcon which is a bare-metal web API framework for Python, and Gunicorn 'Green Unicorn' which is Python WSGI HTTP Server for UNIX.

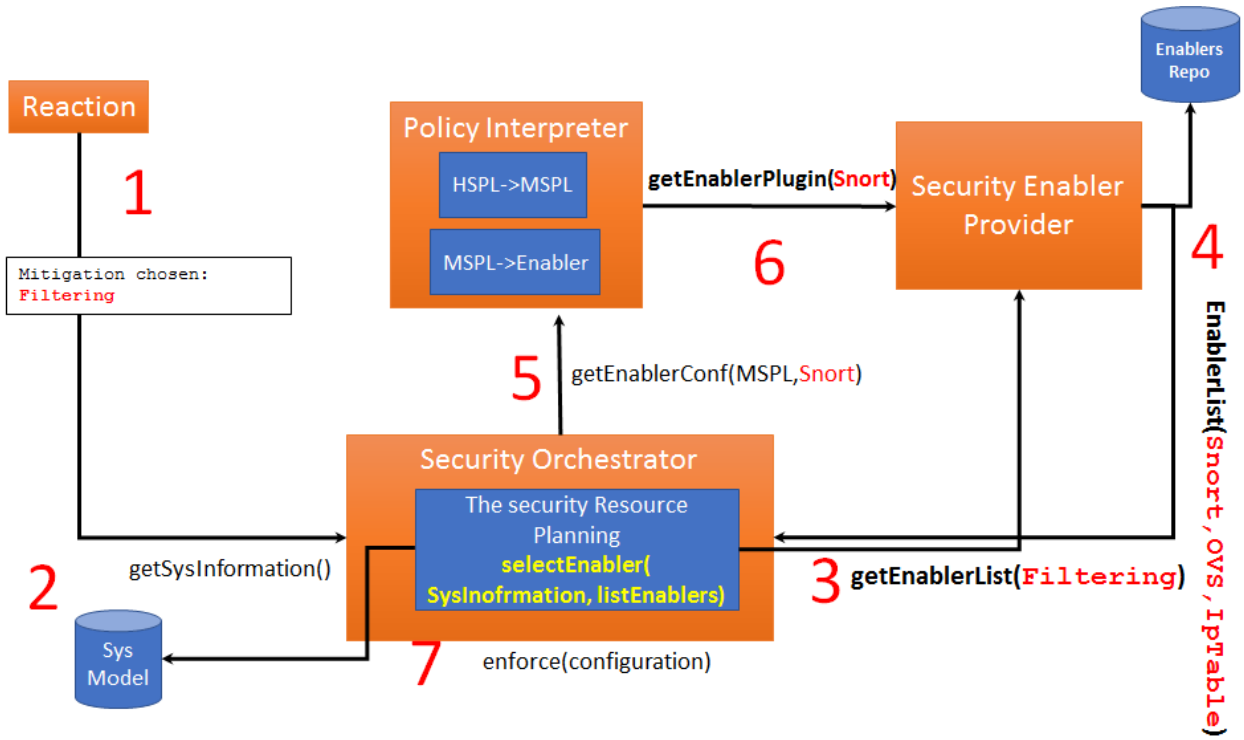

Figure 3Security Enablers Provider Interactions

2.3 SECURITY RESOURCE PLANNING MODULE

The security resource planning uses the list of the selected enablers returned to the security orchestrator by the Security Enabler Provider to decide the more adequate enabler(s) among the list to be used for enforcing the security. This selection is done through an Integer Linear Programming (***ILP***) model. The aim of the model is to select the best service (Virtual Network Function (VNF)) among the list of enablers selected previously by the selected Security Enabler Provider, in order to cope with a security attack and minimize the maximum load nodes (CPU, RAM, bandwidth) of the topology.

The different VNFs are considered as a set of enablers, each enabler is characterized by its type and resources. The security resource planning requests from the SysModel the required topology information. The set of topology nodes is also characterized by its type and its resources R. The goal of the model is minimizing the maximum load nodes to improve provider cost revenue (provider energy efficiency goal). Furthermore, we assume that

- Multiple services (VNFs) can be allocated on the same node,
- A VNF service cannot be split on multiple nodes.
- Each node can host multiple services.

The security Resource Planning Module is implemented as an autonomous plugin that receives a list of the enablers and the topology information from the SysModel. Based on this information, the resource planner runs the ***ILP*** implemented using IBM *CPLEX* Optimizer engine. It selects the needed security enablers, helps to cope with the security attacks, and obtains the nodes where these security enablers have to be installed.

2.4 INTERACTIONS WITH THE MONITORING AGENTS

The Security Orchestrator has the ability to deploy new monitoring agents in runtime if required. This might be the result of the enforcement of a security policy or the reaction against a detected attack on the Security Enforcement Plane (SEP). In both cases, the Security Orchestrator needs to interact with the new instance in order to correctly deploy it, or reconfigure the current instances on the SEP.

2.4.1 MMT-Probe Monitoring Agent

MMT-Probe is one of the monitoring agents that are integrated in the ANSTACIA Platform. A Deep Packet Inspection (DPI) tool that allows capturing packets from the monitored network, extract information from them, and test security properties that allow detecting security incidents.

The capturing capabilities of MMT-Probe are provided by the MMT-DPI library, which makes use of the libpcap or DPDK libraries to extract the packets from the network interface. The library enables to copy the packets from the network interface to the MMT pipeline for its analysis. The whole pipeline is depicted in Figure 4.

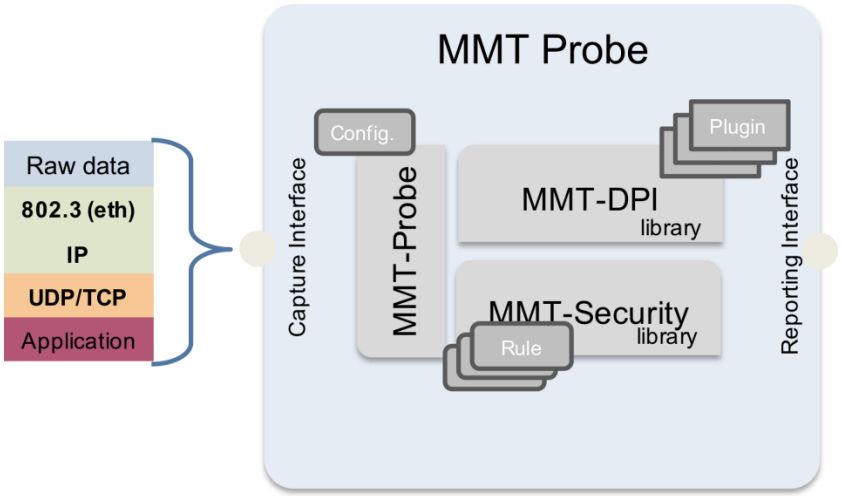

Figure 4 MMT-Probe General Pipeline

Once a packet is extracted, it is analyzed in the MMT-DPI library. This library can be easily extended thanks to his modular structure, through the development of plugins. Each plugin implements the dissection rules and the information extraction for a particular network or application protocol. Typical examples of these plugins are the implementation of the IP, TCP and UDP protocols. In spite of this and since MMT-Probe uses DPI technology to inspect the packet it is possible to implement any upper-layer protocol (such as skype or even multimedia protocols) as long as they have a fixed and recognizable structure.

As long as the dissection is performed, the MMT-DPI library extracts the information according to the format specified in the plugin. This information is then aggregated and used with two principal goals. On one hand, MMT generates a periodical statistics report about the opened connections on the network. On the other hand, the extracted information is also fed to the MMT-Security library in order to perform further security analysis with it.

The MMT-Security library is an extensible engine that can be enhanced by the implementation of new security rules. Using the information extracted with DPI, MMT-Security tests the security properties in order to detect attacks and incidents on the monitored network. The results of such evaluations are also reported in security alerts along with the statistics reports aforementioned.

The two types of reports here described represent the principal output of MMT-Probe. Both reports are published in the Kafka-Broker module that notifies the reports to the ATOS XL-SIEM module. The ATOS XL-SIEM will correlate these reports with further information coming from other sources in order to generate the verdicts about the detected incidents. It is then the XL-SIEM tool that transmits the threats detected by MMT-Probe to the Reaction Plane.

2.4.1.1 MMT-PROBE AND VNF

The MMT technology is a Linux-based technology that can be easily integrated into VNF-enabled platforms. This flexibility is allowed by the fact that MMT is shipped as a ready-to-deploy .deb package, which enables the utilization of the software right after it is being installed. In this sense, the MMT software can be delivered in form of Virtual Machine images, ISO disk images, .deb packages (to be used with Docker) among others. This flexibility is supported by the fact that the SDN controllers usually employ a container or virtualization approach which allows emulating a complete Linux environment where MMT can easily be installed.

2.4.1.2 MMT-PROBE CONFIGURATIONS FOR VNF ENVIRONMENTS

MMT comes with comprehensive set of configurations in order to adapt the processes of sniffing, capturing, extraction, security testing and report of the raised alerts. Despite this, the configuration options can be classified into two principal groups:

- *Resources used by MMT*: This group of configurations specifies how MMT should use the resources of the machine running the software. It includes:
    - thread-nb: Indicated the number of threads MMT-Probe will use to process packets.
    - logfile: The location of the log file of MMT.
    - input-mode: This is the analysis mode of MMT. “Online” (direct capture from a network interface) and “offline” (pcap analysis) are the valid values for this field.
    - input-source: This is the source of the packets to extract. For “online” mode, the name of the interface is required. For “offline” mode, the name of the pcap file is required.
- *MMT Reports*: This section includes configurations about the reports generated by MMT. The principal options are:
    - kafka-output: This set of configurations allow publishing the generated reports in a Kafka channel. It requires a set of sub configurations.
    - enabled: “0” (false) or “1” (true) to disable or enable Kafka output.
    - hostname: The hostname of the Kafka server.
    - port: the port number of the Kafka server.
    - stats-period: This indicates to MMT that it should generate statistics report each X seconds.
    - session-report report_session: This set of configurations enables or disables the reporting of protocols that belong to a session.
    - event_report: This set of configurations is used to create customized reports based on events. Please refer to the MMT Manual for further information about this.
    - condition_report: This set of configurations is used to create customized reports based on conditions. Please refer to the MMT Manual for further information about this.
    - security2: This configuration sets the options of the security reports. It is composed of the following sub-options:

- thread-nb: Specifies in which thread the security evaluation should be performed.
- exclude-rules: A list of the rules's IDs that are excluded from the analysis.
- cpu-mem-usage: This set of options configures the periodic CPU and memory usage reports. If contains the following options:
- enable: Enables or disables these reports.
- frequency: Sets the time interval (in seconds) to generate these reports.

The aforementioned list of options is specified in a text file located in /opt/mmt/probe/conf/online.conf. This file can easily be generated on demand and implanted in a VNF instance in order to correctly configure the newly deployed DPI module.

The presented list is not extensive, and it is intended to act as an initial reference of the possible configurations of the MMT software. For further reference and an exhaustive list of the available options, please refer to the MMT manual provided with this document [2].

3 ORCHESTRATION OF SECURITY ENABLERS AND RELEVANT CONTROLS

In order to accommodate the constraints and heterogeneity of IoT systems, softwarized networks seem to be the most compelling solution. Network softwarization is a recent promising trend aiming at radically advancing telecommunication industries by embracing cloud computing technologies and software models in network services. The main pillars behind this revolution are Software Defined Networking (SDN) and Network Function Virtualization (NFV). On one hand, SDN introduces a new level of network programming by decoupling control and data plane. A logically centralized controller is in charge of supervising the network state and provides rules to the network elements for appropriately managing the traffic flows. On the other hand, NFV leverages virtualization technologies to deploy network elements as software instances, thus allowing an increased level of flexibility and elasticity in service provisioning. Furthermore, NFV can enable remarkable reduction in CAPEX/OPEX costs, by replacing dedicated expensive hardware with commodity servers able to host software-based network appliances.

The IoT paradigm is drastically enhancing our quality of life and utilization of resources to make things (i.e., home appliances, electronic devices, sensors, etc.) part of the Internet. This paradigm opens doors to innovations that will build novel type of interactions among things and humans and enables the realization of smart infrastructures, smart cities, etc. Although SDN and NFV are two separate paradigms, their joint use can further improve the potential security services offered by the network and meet the broad range of increasing requirements imposed by IoT applications.

In this section, we will present the main concepts used by the Security Orchestrator in order to enforce the relevant security policies in the IoT domain.

3.1 SDN NETWORKING

Software Defined Networking (SDN) is a relatively new paradigm, which aims to decouple the control plane and the data plane for reducing the management complexity and allowing external application to control the network's behavior. SDN offers novel capabilities to adapt, on the fly, the network flows according to the dynamic application requests. The three main components of SDN networks are switches, controllers, and communication interfaces.

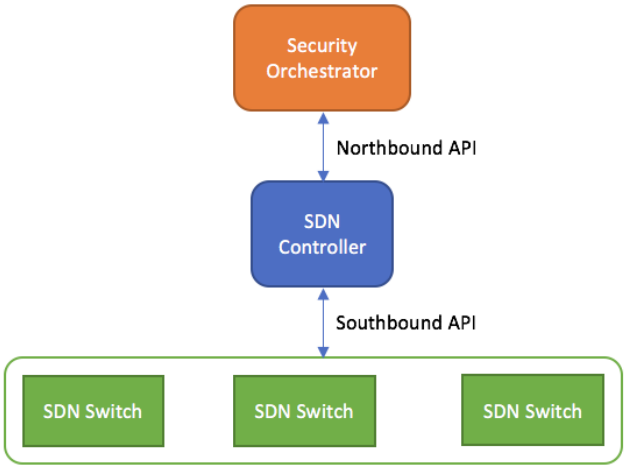

Figure 5 SDN Networking Architecture & Management

a) SDN switches: the forwarding of data traffic is the main responsibility of physical and virtual SDN switches according to the assigned network configuration. To this aim, appropriate rules are instructed by the SDN controller in the flow tables to perform packet forwarding along with a wide range of other operations.

b) SDN controller: the intelligence of an SDN-based network is centralized in its SDN controller, which maintains the state of the whole system and decides on the traffic routing by updating relevant flow rules on the switches. Its role allows it to have a global vision over the network, which makes for the most optimized and dynamic networking decisions.

c) SDN interfaces: to facilitate programming and management, the communication interfaces are fundamental to configure the network behavior. It consists of two main interfaces: southbound and northbound. While the first one manages the communication between the SDN controller and the SDN switches, the second one is in charge of the interactions between the user SDN applications and the SDN controller.

The adoption of SDN in IoT (SDN-enabled IoT systems) is considered as an essential element in the success of the Security Orchestrator. Leveraging the capabilities of SDN to route, efficiently, the traffic and optimize the utilization of the network are key enabling functions to manage the massive amounts of data flow in IoT networks and eliminate bottlenecks. This integration can be implemented at different levels of the IoT network, such as the access (where the data is generated), core and cloud networks (where the data is processed and served), which enables IoT traffic management from end-to-end.

Moreover, SDN can be also leveraged to provide advanced security mechanisms for IoT systems. For example, traffic isolation between different tenants, centralized security monitoring using the global vision of the network and traffic dropping at the edge, keeping the malicious traffic from spreading all over the network.

3.2 VIRTUAL NETWORK FUNCTIONS

Network Function Virtualization (NFV) refers to the adoption of virtualization technologies in network environments. Unlike traditional network equipment, NFV decouples the software from the hardware, bringing value added features and notable capital and operating expenditures gains. The ETSI (European Telecommunications Standards Institute) has been leading the standardization of this approach, defining novel architectures, which enable the aforementioned advantages. The ETSI NFV architecture identifies three main building blocks:

a) Virtualization Infrastructure: this layer includes all the hardware and virtualization technologies necessary to provide the desired resource abstractions for the deployment of Virtualized Network Functions (VNFs). This includes Storage, Computing and Networking resources that are usually managed by a cloud platform.

b) Virtual Network Functions: the core idea of NFV deals with replacing dedicated hardware equipment with software-based instances of network functions, i.e., the VNFs. They can be deployed and managed over multiple environments, providing scalable and cost-effective network functions.

c) Management and Orchestration: the NFV Management and Orchestration (MANO) module interacts with both the infrastructure and VNF layers in the ETSI NFV architecture. It is responsible for the management of the global resource allocation which includes: instantiating, configuring and monitoring VNFs.

Introducing virtualized network resources into the IoT ecosystem brings multiple value-added features, accounting for their heterogeneity and rapid growth. When coupled with SDN, NFV cannot only, provide advanced virtual monitoring tools such as Intrusion Detection Systems (IDSs) and Deep Packet Inspectors (DPIs), but also provision, and configure on-demand and scalable network security appliances, such as firewalls and authentication systems, in order to cope with the attacks detected by the monitoring agents.

3.3 IOT SECURITY CONTROLS

IoT security controls comprise a set of operations that can be applied over the IoT infrastructure through the IoT controller. The IoT controller is in charge of managing the command and control over the IoT domain, offering high-level of abstraction operations in order to manage the infrastructure independently of the underlying technology. It has been designed aligned with the SDN controller philosophy.

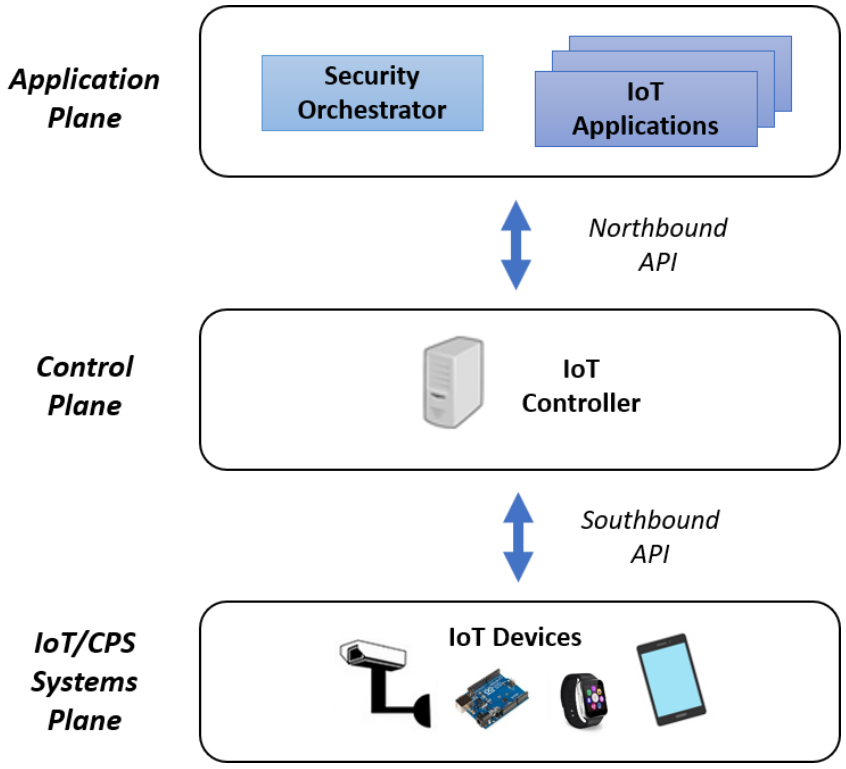

Figure 6 Different planes and APIs contemplated for the IoT controller architecture

Figure 6 shows the different planes and APIs contemplated for the IoT controller architecture, which are introduced below:

1. **Application Plane:** In this plane are allocated the applications which require performing some kind of control actions over the IoT domain. In this case, the *Security Orchestrator* could be part of this plane in order to carry out the different IoT operations according to the security policies. This plane performs the communications with the control plane through the Northbound API.
2. **Northbound API:** This API provides a high-level of abstraction API which allows receiving IoT commands and controls independently of the underlying technology, (e.g. HTTP REST API).
3. **Control Plane:** This plane is governed by the IoT Controller, which is in charge to receive command and controls from the Application plane and then to perform the specified operations over the required IoT devices using specific IoT communication protocols through the Southbound API.
4. **Southbound API:** This API provides the communication between the IoT Controller and the IoT devices using specific IoT communication protocols depending on the IoT device implementation and requirements.
5. **IoT/CPS Systems Plane:** The different IoT devices in the architecture, both physical and virtual, comprise this plane.

3.4 SECURITY ORCHESTRATOR SYSTEM MODEL

The System Model is an internal component of the Security Orchestrator which is in charge of storing data relevant to the underlying infrastructure and enforced security policies. This data is made available to all of the ANASTACIA components in order to further refine 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.

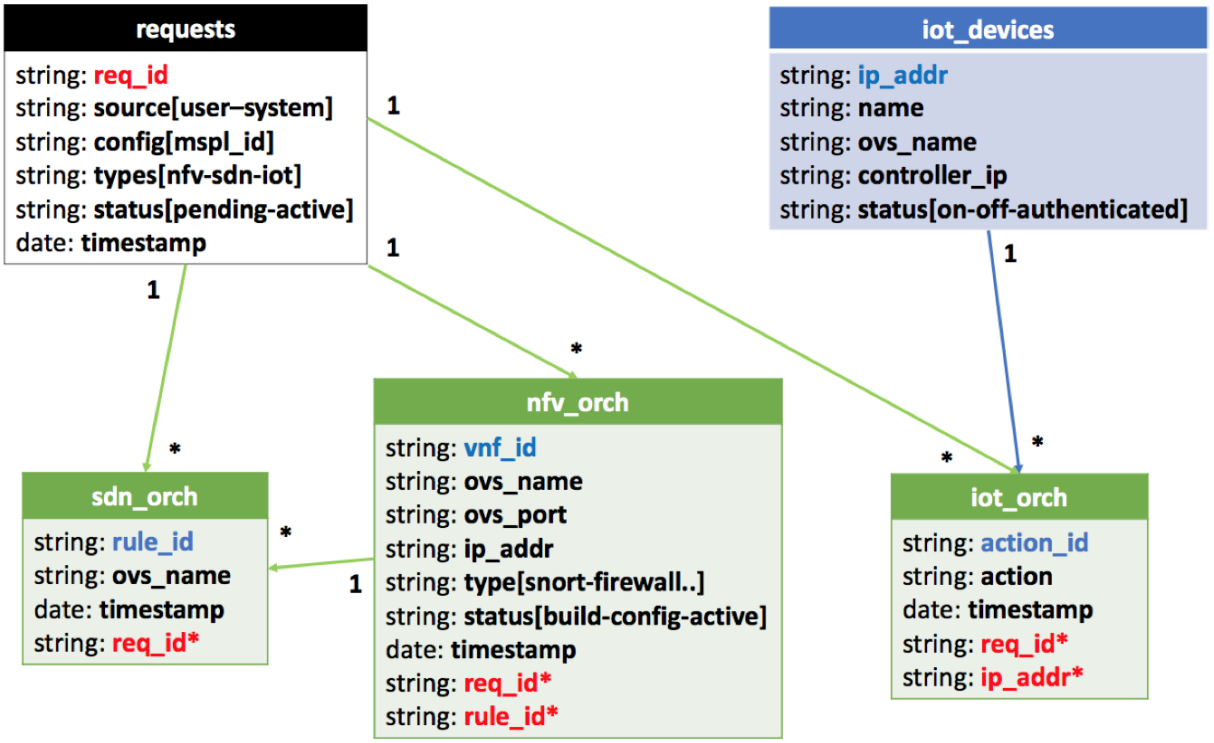

Figure 7 System Model Data Structure

As the Security Orchestrator expects to receive an MSPL file, each request with its unique ID is mapped to a class depending on the required security capabilities. An MSPL could require actions on multiple controllers (e.g., Cooja needs a VM in the OSM and filters in the ONOS). There are three capability classes: sdn_orch, nfv_orch and iot_orch. Moreover, the System Module keeps track of the existing IoT devices on the IoT domain along with their status.

- **requests (Security Orchestrator requests class):** This class stores the different requests received by the Security Orchestrator. Each request has:
    - **req_id: is** a unique ID to each request generated by the Security Orchestrator.
    - **source[user-system]: is** the source of the request. It can be either “system” which means that the request came from the Reaction Agent of ANASTACIA, or “user” which means that it came from the Policy Editor Tool as user-designed policy.
    - **config:** is the unique ID of the MSPL file which relates to the request.
    - **type:** can be either “sdn”, “nfv” or “iot”, which depends on the required capability. This attribute is used to map the requests to the correct capability class.

- **status:** reports the status of the security policy enforcement status. The default value is “pending” which is set right after receiving the request, it changes to “active” as soon as the policy has been enforced.
- **timestamp: is** the time and date of the received request.

➢ **sdn_orch (SDN orchestration class):** This class contains all the details related to the SDN configuration, which is required either through a direct SDN policy enforcement, or through part of a VNF configuration:

- **rule_id:** a unique ID of the injected SDN flow rule.
- **ovs_name:** the name of the Open Virtual Switch on which the rule has been enforced.
- **req_id*:** relevant request ID from the “requests” class.

➢ **nfv_orch (NFV orchestration class):** is the class that stores all the information concerning the security VNFs created through Open Source Mano.

- **vnf_id:** unique ID of the relevant security enabler.
- **ovs_name:** name of the OVS on which the VNF is attached.
- **ovs_port:** the port ID of the OVS on which the VNF is attached
- **ip_addr:** management IP address of the VNF.
- **type:** type of the VNF (Snort, Firewall…).
- **status [build-config-active]:**current status of the VNF (instantiating, configuring, active).
- **req_id*:** relevant request ID from the “requests” class.
- **rule_id*:** relevant SDN rule ID from the “sdn_orch” class.

➢ **iot_orch (IoT orchestration class):** this class keeps track of all the IoT actions, which were taken as mitigation actions due to a certain security policy.

- **action_id:** ID of the action to take on an IoT device.
- **action:** action to take (turn on/off).
- **req_id*:** relevant request ID from the “requests” class.
- **ip_addr*:** relevant IP address of the IoT device from the “iot_devices” class.

➢ **iot_devices (IoT devices class):** contains all the information regarding the IoT devices, including IP address mapping and status.

- **ip_addr:** IPv6 address of the IoT node.
- **name:** name of the IoT node (as provided by the user)
- **ovs_name:** the open virtual switch that is managing the IoT network.
- **controller_ip:** IP address of the IoT controller of this node.
- **status [on-off-authenticated]:**current status of the node.

3.5 ORCHESTRATION STRATEGIES

3.5.1 Orchestration Tools

In order to implement the intelligent orchestration, the Security Orchestrator interacts with three key components:

- **The IoT controller:** Used to enforce IoT-specific mitigation actions, such as IoT devices access control, authentication and power on/off. These interactions are done through Rest-API to send queries to the IoT controller depending on the security policy provided by the MSPL file.
- **The NFV MANO:** An ETSI-defined framework designed for managing and orchestrating resources in the cloud. It is used by the security orchestrator to create and configure a wide range of security enablers. It has three main functioning blocks:
    - NFV Orchestrator: Manages the registration of Network Services (NS) and Virtual Network Function (VNF) packages, lifecycle of different network services and the resources allocation requests.
    - VNF Manager: Configures and monitors each VNF after its instantiation.
    - Virtualized Infrastructure Manager (VIM): Interacts with the compute, network and storage resources (clouds) in order to provision relevant VNFs.
- **The SDN controller:** is accountable for managing network resources and enabling the programming of the underlying network. The SDN orchestration is done through the ONOS driver. This driver has been developed in order to automate the SDN management using one or multiple ONOS SDN controllers. 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.
    - Bandwidth limitation.

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 VNF 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.

Figure 8 shows the general architecture of the Security Orchestrator, which enables its security capabilities.

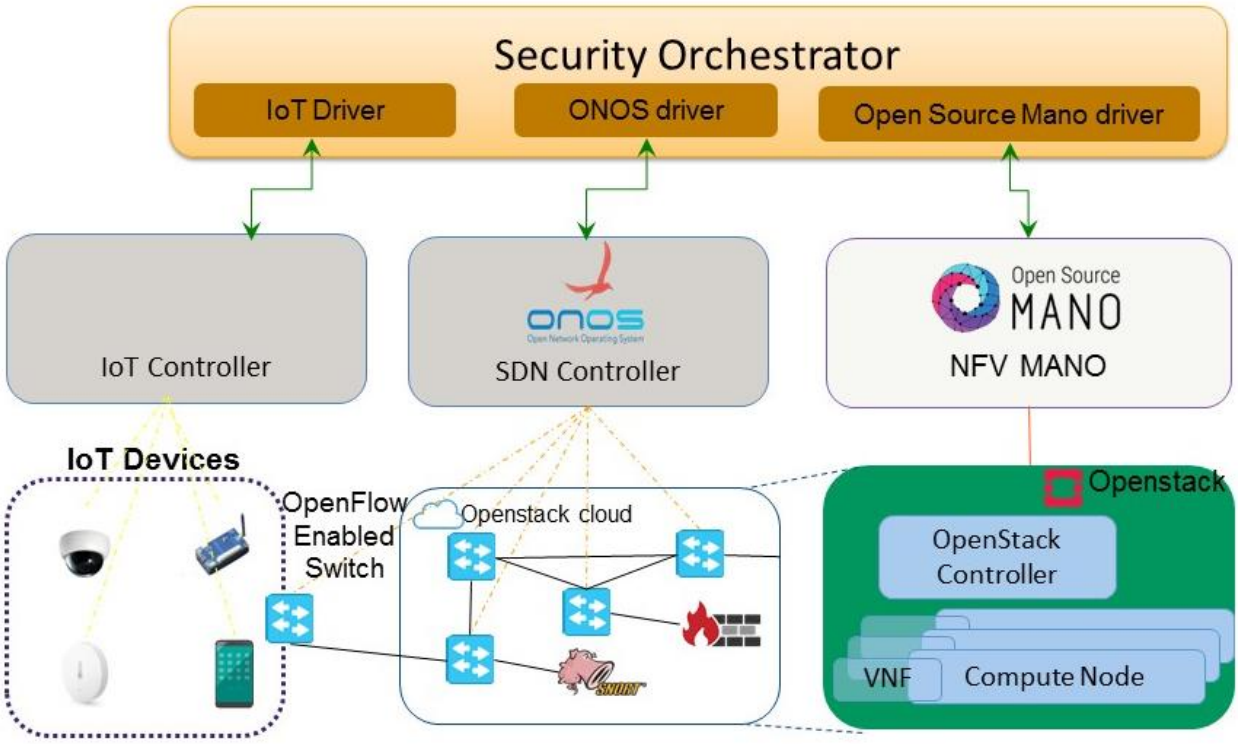

Figure 8 Security Orchestrator architecture

3.5.2 Sample Orchestration Scenario

Figure 9 shows a sequence diagram that describes the overall process of security orchestration from the processing to the enforcement.

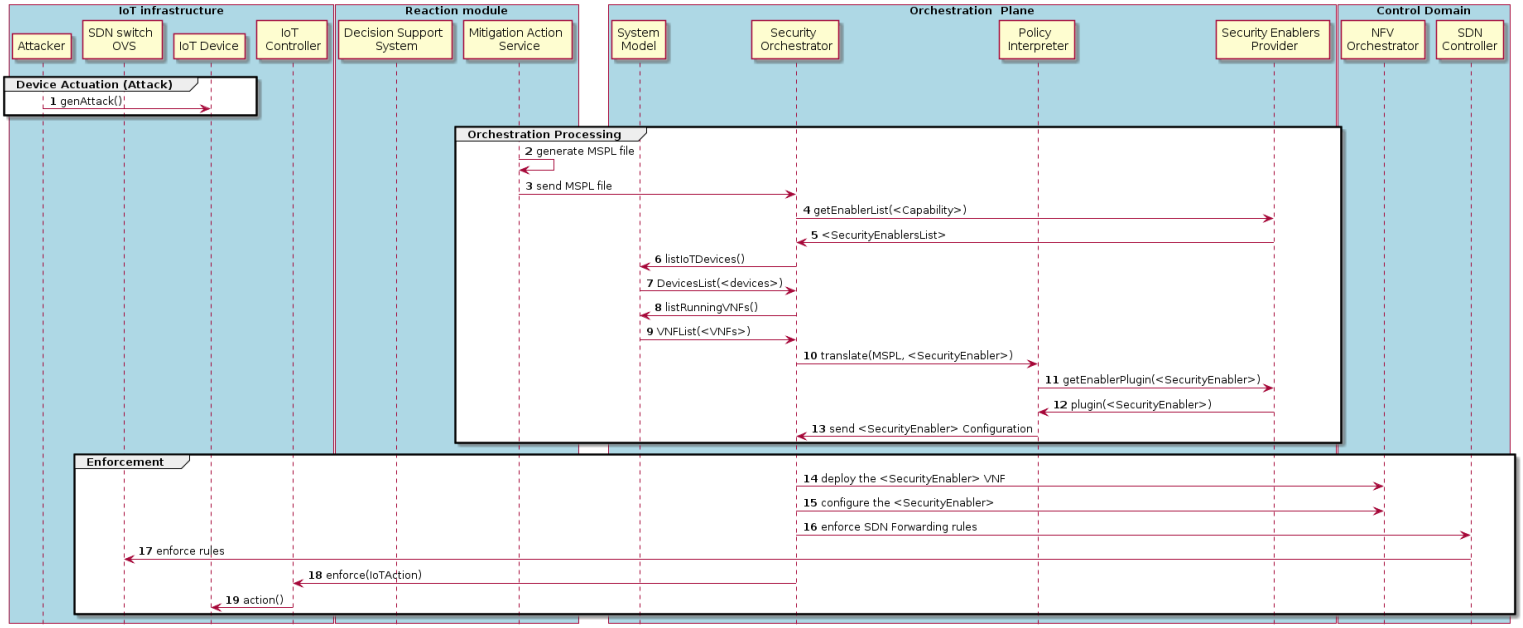

Figure 9 Sequence diagram for Security orchestration (processing and enforcement)

1. Attack generation from the Attacker to the target node.
2. The Mitigation Action Service generates an MSPL file describing the set of actions in order to mitigate the attack
3. The Mitigation Action Service sends the MSPL file to the Security Orchestrator through its Rest API interface.
4. The Security Orchestrator queries the supported security enablers from the Security Enablers Provider.
5. The Security Enablers Providers sends a list of supported security enablers which correspond to the desired security policy as instructed by the Mitigation Action Service.

6-7-8-9. The Security Orchestrator checks for the status of the underlying infrastructure by listing the different IoT devices and running VNFs by sending relevant requests to the System Model.

10. After identifying the suitable enabler for the desired security policy, the Security Orchestrator queries the low-level configuration for the relevant enabler from the Policy Interpreter.

11. The Policy Interpreter queries the enabler's plugin from the Security Enablers Provider.

12. The Security Enablers Provider sends the low-level translator plugin to the Policy Interpreter in order to generate the specific configuration.

13. The Policy Interpreter sends the relevant configuration to the Security Orchestrator.

If the Security Policy is a VNF policy:

14. The Security Orchestrator deploys the relevant VNF.

15. The Security Orchestrator configures the relevant VNF using the configuration received from the Policy Interpreter.

16. The Security Orchestrator enforces the SDN rules in order to make the VNF operational (For example: traffic mirroring in case of an IDS, traffic steering in case of a firewall).

If the Security Policy is an SDN policy:

17. The SDN controller enforces the relevant rules into the Open Virtual Switch in the IoT Infrastructure.

If the Security Policy is an IoT control policy:

18. The Security Orchestrator forwards the action request to the IoT Controller.

19. The IoT Controller enforces the required action on IoT Devices.

3.6 REST APIs AND BASIC MECHANISMS

For the different interactions, the Security Orchestrator exposes a REST API interface which enables the following functionalities: MSPL Policy Enforcement and Information about the underlying infrastructure.

1. **MSPL Policy Enforcement**: The Security receives queries in order to enforce security policies either directly from the Policy Interpreter as a "Proactive Policy" or from the Reaction Agent as a "Reactive Policy". Both can be enforced using the following Rest interface:

   URL: **http://<SecurityOrchestratorIP>/enforce**

   METHOD: **POST**

   DATA: **{**

   "policy": <MSPL_FILE_CONTENT>,

   "enablers":<List of Security Enablers>

   }

   RETURNS: **The Security Orchestrator returns a unique request ID that can be used to track the policy enforcement process.**

Optionally, a list of security enabler candidates list can be added as an argument. The IP addresses if needed would go inside the MSPL.

2. Information about the requests: The Security Orchestrator can provide information regarding the requests and their current status using the following Rest API interface:

URL: **http://<SecurityOrchestratorIP>/info/requests/<req_id>**

METHOD: **GET**

RETURNS: **The Security Orchestrator returns the VNF/SDN/IoT details relevant to the request with id: <req_id>. If no request ID was supplied, it returns the details concerning all the enforced security policies.**

Sample response:


```
{
   "requests":[
      {
         "status":"pending",
         "type":"NFV",
         "source":"user",
         "req_id":"45",
         "time":"2018-4-9, 18:2:30",
         "config":"MSPL"
      }
   ]
}
```


3. Information about the IoT devices: The Security Orchestrator can provide information regarding the IoT devices present in the network and their current status using the following Rest API interface:

URL: **http://<SecurityOrchestratorIP>/info/devices/<device_ip>**

METHOD: **GET**

RETURNS: **The Security Orchestrator returns the details relevant to the IoT device with IP: <device_ip>. If no IP address was supplied, it returns the list of all current IoT devices.**

Sample response:


```
{
   "devices":[
      {
         "status":"ON",
         "ip":"172.16.1.100",
         "name":"camera1",
         "ovs_name":"br-edge",
         "controller_ip":"172.16.1.10"
      },
      {
         "status":"OFF",
         "ip":"172.16.1.101",
         "name":"camera2",
         "ovs_name":"br-edge",
         "controller_ip":"172.16.1.10"
      }
   ]
}
```

4 USE CASES DESCRIPTION FOR THE SECURITY ORCHESTRATOR

In this section we will present the different initial use cases which aim to show the different capabilities of the ANASTACIA framework. We will briefly explain the aim of each use case, followed by the relevant sequence of events in order to mitigate different types of attacks defined in the document D6.2. [3]

4.1 BMS.2: INSIDER ATTACK ON THE FIRE SUPPRESSION SYSTEM

The main objective of this use-case is the evaluation of ANASTACIA framework to protect the system against an insider attack and avoid any damage to the building assets. In this use-case, the attacker exploits the building operations workstation to request the activation of fire alert system managed by an IoT device.

The following figure shows the main interactions of the Security Orchestrator for a security policy deployment in a reactive approach. The figure shows the messages exchanged by the ANASTACIA components in the Orchestration Plane and the Enforcement Plane. The Reaction module is showed just in order to provide the starting point of the workflow. In this case, the process is compounded by the following steps:

1. The reaction module sends an MSPL policy as a reaction measure.
2. The Security Orchestrator obtains the list of candidate security enablers in an interaction with the Security Enabler Provider.
3. The Security Orchestrator obtains the list of deployed IoT-devices in an interaction with the System Model.
4. The Security Orchestrator obtains the list of running VNFs in an interaction with the System Model.
5. The Security Orchestrator makes the decision regarding which Security Enabler must be used in order to enforce the security MSPL policy.
6. The Security Orchestrator requests a policy translation to the Policy Interpreter in order to get the configuration for the selected security enabler which is Cooja Honeynet for the use case of BMS.2.
7. The Policy Interpreter performs the translation, using a plugin received in another interaction with the Security Enabler Provider (not shown in the figure for simplicity).
8. The Policy Interpreter sends the Cooja-Honeynet configuration to the Security Orchestrator.
9. The Security Orchestrator requests to VNF Controller for the deployment of a Honeynet with virtual IoT devices using Cooja emulator.
10. The Security Orchestrator requests to SDN Controller for the traffic forwarding from the real IoT network towards the virtual Honeynet in VNF deployment.

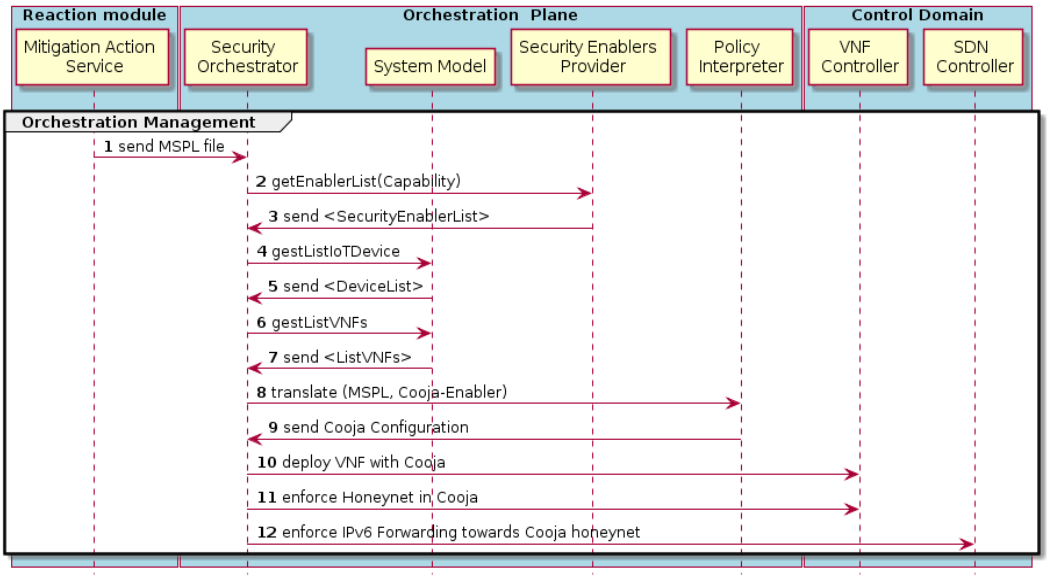

Figure 10 Sequence diagram for use case BMS.2

4.2 MEC.3: DoS/DDoS ATTACKS USING SMART CAMERAS AND IoT DEVICES

The MEC.3 use case aims to show how ANASTACIA can cope with one of the most well-known IoT attacks. Indeed, DoS (Denial of Service) attacks are hindering, until this day, the wide adoption of IoT. The disruption of services induced by this type or security flaws can be catastrophic, especially in industrial and health care environments.

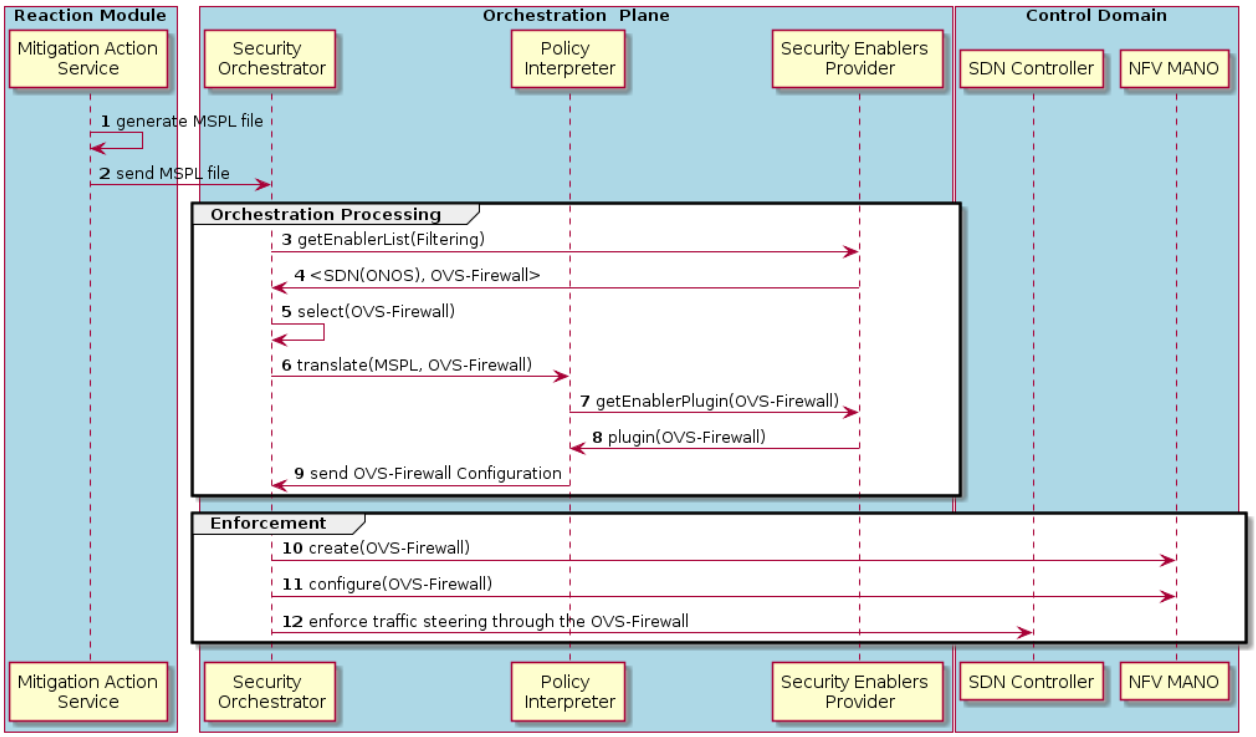

Figure 11 Sequence diagram for MEC.3 use case interactions with Security Orchestrator

In this scenario, the attacker targets an IoT device by flooding the network with ICMP packets, disrupting its functionalities. After the detection of the attack, the framework goes through the following sequence of events:

1. The Reaction module generates the MSPL file describing the set of mitigation actions to be taken in order to patch the security flaw.
2. The Mitigation Action Service sends the MSPL Filtering policy file to the Security Orchestrator.
3. After receiving the desired policy, the Security Orchestrator queries the list of the filtering security enablers from the Security Enablers Provider.
4. The Security Enablers Provider returns a list containing all the supported security enablers which can enable traffic filtering.
5. The Security Orchestrator picks the most appropriate enabler depending on the cost and status of the underlying infrastructure.
6. The Security Orchestrator chooses the OVS-Firewall as a filtering security enabler, then it queries the low-level configuration which corresponds to the MSPL file. It sends the relevant request to the Policy Interpreter
7. The Policy Interpreter requests the OVS-Firewall translation plugin.
8. The Security Enablers Provider returns the OVS-Firewall plugin that will be then used by the Policy interpreter to perform the low-level translation.
9. The Policy Interpreter sends the relevant low-level configuration to the Security Orchestrator.
10. After gathering all the required inputs for the Enforcement of the security policy, the Security Orchestrator creates the OVS-Firewall VNF using the NFV Orchestrator.
11. As soon as the VNF is accessible, the Security Orchestrator refines the default configuration of the VNF to adapt it to the current security threat.
12. As a final step, using SDN networking, the Orchestrator enforces the necessary rules in the Open Virtual Switches that will alter the default routing mechanisms and make all the packets go through the newly deployed firewall. This way, only the malicious traffic is stopped, and all the services provided but the IoT devices are kept.

4.3 BMS.3: REMOTE ATTACK ON THE BUILDING ENERGY MICROGRID

This use case will show how ANASTACIA can cope with a remote attack on the building energy microgrid. The attack is a SQL injection that targets the backend database server. The MMT monitoring tool is capable of detecting such threats in order to enable the ANASTACIA mitigation of such attacks.

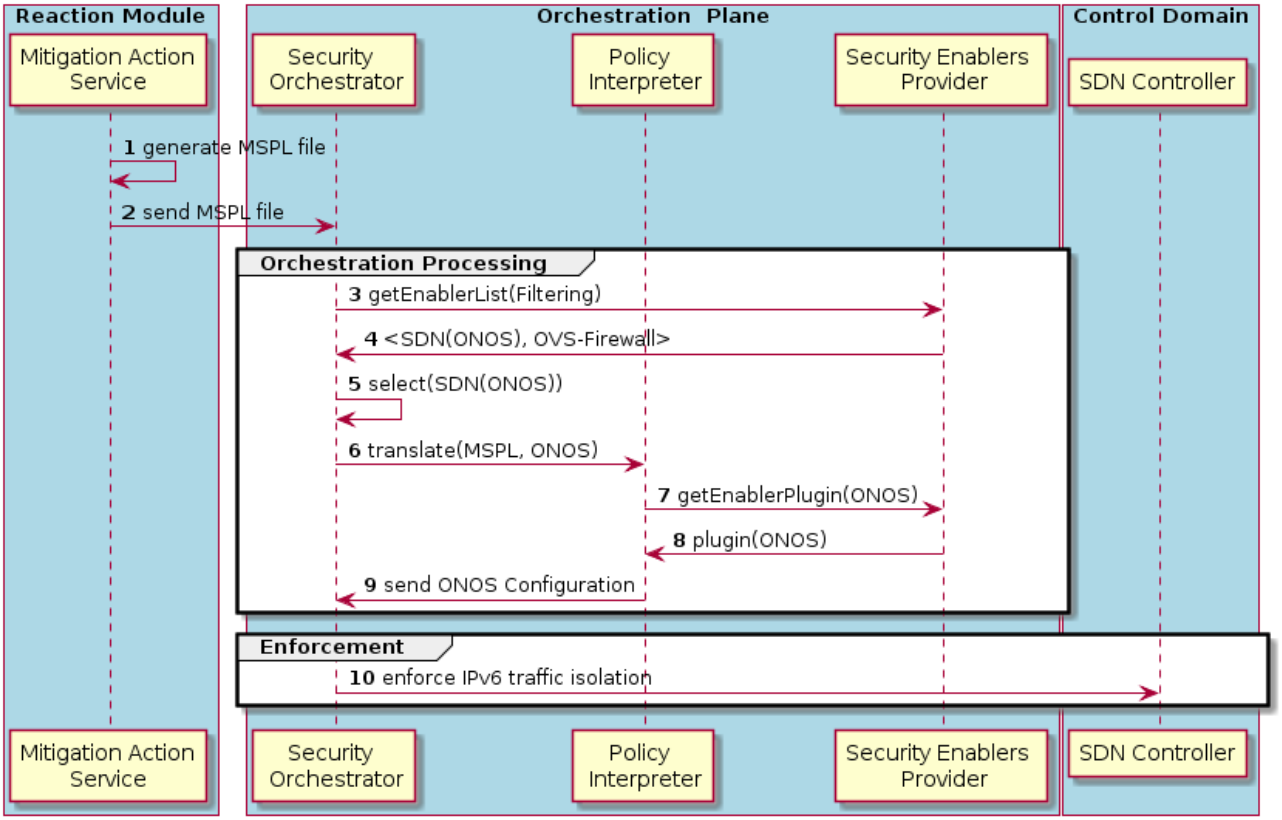

Figure 12 Sequence diagram for the BMS.3 use case

After the detection of the attack, the ANASTACIA framework goes through the following sequence of events:

1. The Reaction module generates the MSPL file describing the set of mitigation actions to be taken in order to patch the security flaw.
2. The Mitigation Action Service sends the MSPL Filtering policy file to the Security Orchestrator.
3. After receiving the desired policy, the Security Orchestrator queries the list of the filtering security enablers from the Security Enablers Provider.
4. The Security Enablers Provider returns a list containing all the supported security enablers that can enable traffic filtering.
5. The Security Orchestrator picks the most appropriate enabler depending on the cost and status of the underlying infrastructure.
6. The Security Orchestrator chooses ONOS as a filtering security enabler, then it queries the low-level configuration which corresponds to the MSPL file. It sends the relevant request to the Policy Interpreter
7. The Policy Interpreter requests the ONOS policies translation plugin.
8. The Security Enablers Provider returns the ONOS plugin, which will be then used by the Policy interpreter to perform the low-level translation.
9. The Policy Interpreter sends the relevant low-level configuration to the Security Orchestrator.
10. After gathering all the required inputs for the Enforcement of the security policy, the Security Orchestrator, using the SDN driver, instructs the SDN controller to isolate the malicious traffic on the edge OVS.

4.4 BMS.4: CASCADE ATTACK ON A MEGATALL BUILDING

BMS.4 use case demonstrates ANASTACIA capability of detecting attack on sensor network by changing temperature of given set of them and then triggering fake fire and evacuation alarms causing harm in a tall building infrastructure, panic among personnel and wasted productivity time. Figure 13 illustrates interactions between Security Orchestrator and other ANASTACIA components in use case BMS.4. Internal SO interactions have been depicted in section 4.1.

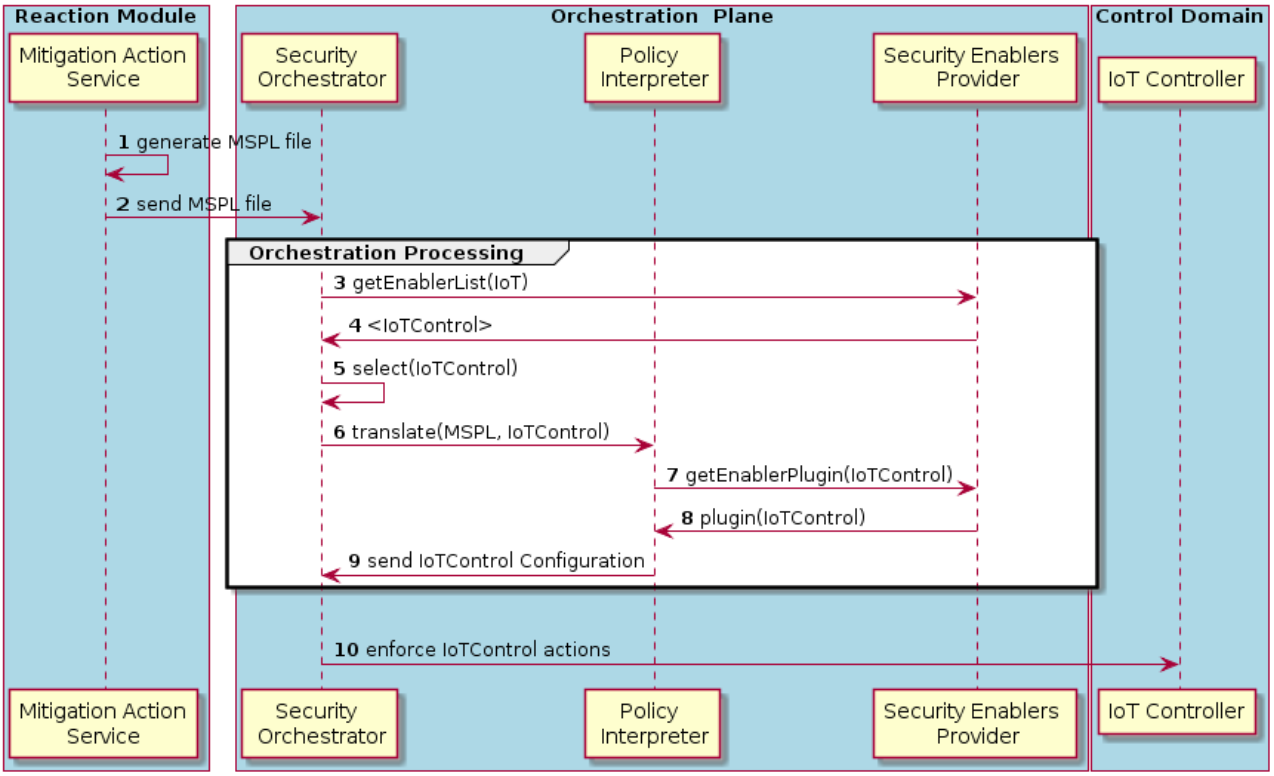

Figure 13 Sequence diagram for the BMS.4 use case

ANASTACIA is able to detect abnormal IoT device behavior thanks to the "Data Analysis" component within the Monitoring Module. From the Security Orchestrator point of view, the defined mitigation action is to turn off the infected temperature sensor. The security enforcement process is as follows:

1. The Reaction module generates the MSPL file describing the set of mitigation actions to be taken in order to patch the security flaw.
2. The Mitigation Action Service sends the MSPL Filtering policy file to the Security Orchestrator.
3. After receiving the desired policy, the Security Orchestrator queries the list of the security enablers, which can enforce IoT specific mitigation actions for the capability "IoT_control".
4. The Security Enablers Provider returns the supported security enablers accordingly. In this case, it returns "IoT Controller".
5. Since the Security Orchestrator received only one security enabler in this use case, it selects this security enabler.
6. The Security Orchestrator then queries the low-level configuration that corresponds to the MSPL file. It sends the relevant request to the Policy Interpreter.

7. The Policy Interpreter requests the IoTController policy translation plugin.
8. The Security Enablers Provider returns the IoTController plugin, which will be then used by the Policy interpreter to perform the low-level translation.
9. The Policy Interpreter sends the relevant low-level configuration to the Security Orchestrator.
10. After gathering all the required inputs for the Enforcement of the security policy, the Security Orchestrator enforces the mitigation action through the Rest API interface of the IoT controller.

5 Implementation of the Security Orchestrator

This section presents the details regarding the implementation of the Security Orchestrator.

5.1 Security Orchestrator Code Base

The Security Orchestrator is divided into 4 main sub-projects: **IoT Orchestration**, **Network Orchestration**, **NFV Orchestration** and **System Model**. Each query received via the Rest API interface is forwarded to the **Security Manager** that will be responsible for the orchestration part.

Figure 14 depicts the structure of the project.

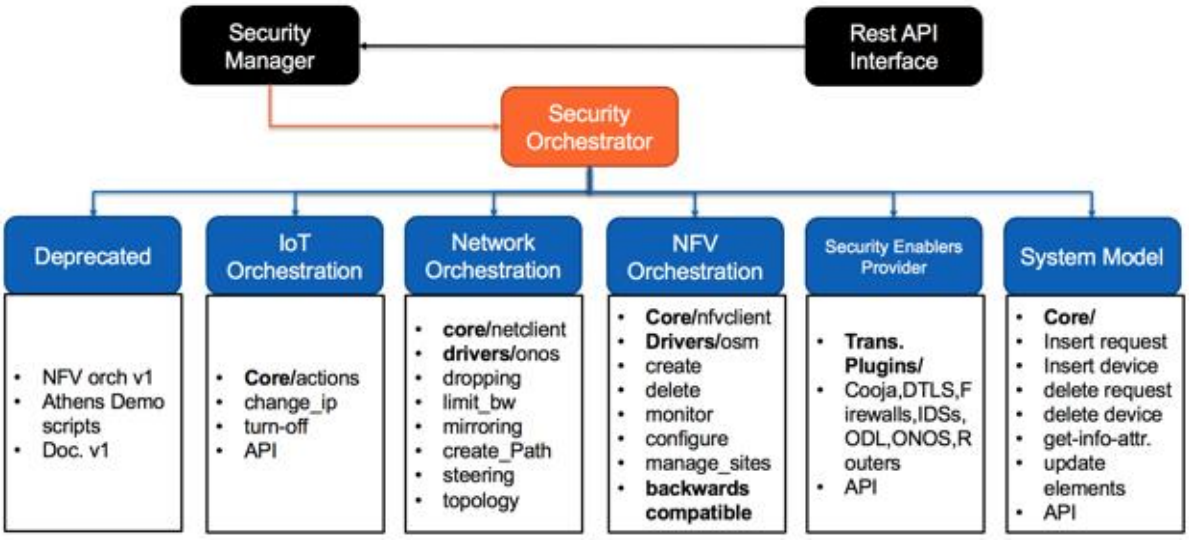

Figure 14 Structure of the Security Orchestrator Code Base

- **IoT Orchestration:** Contains the libraries that enable the Security Orchestrator to execute remotely actions on the IoT devices through the IoT controller's Rest API interface.
- **Network Orchestration:** This subproject is comprised of a library, which adds an extra abstraction layer on top of the SDN Northbound API, in order to enable SDN management functionalities that are relevant to policy enforcement, such as traffic isolation, traffic mirroring, and traffic steering and path creation.
- **NFV Orchestration:** This part of the project takes care of instantiating and configuring security VNFs using the NFV orchestrator Open Source Mano.

- **System Model:** Is comprised of a database and library to manage it. This database contains all of the information regarding the received policy enforcement requests, IoT devices, SDN rules and security VNFs.

5.2 Open Source MANO

OSM is a NFV Orchestrator officially launched at the World Mobile Congress (WMC) in 2016, founded by Mirantis, Telefonica, BT, Canonical, Intel, RIFT.io, Telekom Austria Group and Telenor. It is compliant with the ETSI NFV MANO reference architecture and offers support for multi cloud and SDN vendors support (Openstack, AWS, ONOS and Opendaylight). [4]

It is comprised of three basic components:

a) **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.
b) **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.
c) **The VNF Configuration and Abstraction (VCA):** performs the initial VNF configuration and constant monitoring using Juju Charms LXD containers.

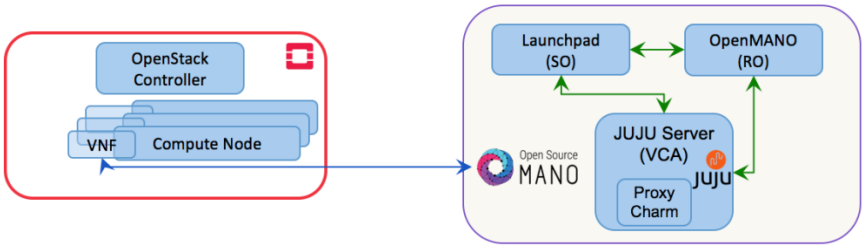

Figure 15Overview of Open Source Mano components

5.3 ONOS SDN Controller

ONOS (Open Network Operating System) is an open source project that aims to create an SDN operating system for communications and service providers [5]. It is well known for its high performance, scalability and high availability. It uses standard protocols, such as OpenFlow and NetConf in order to expose advanced traffic manipulation functions through its applications. These applications provide a high level of abstractions, while giving detailed information about the network, such as existing nodes, number of packets of a certain traffic and existing links, making application development much simpler.

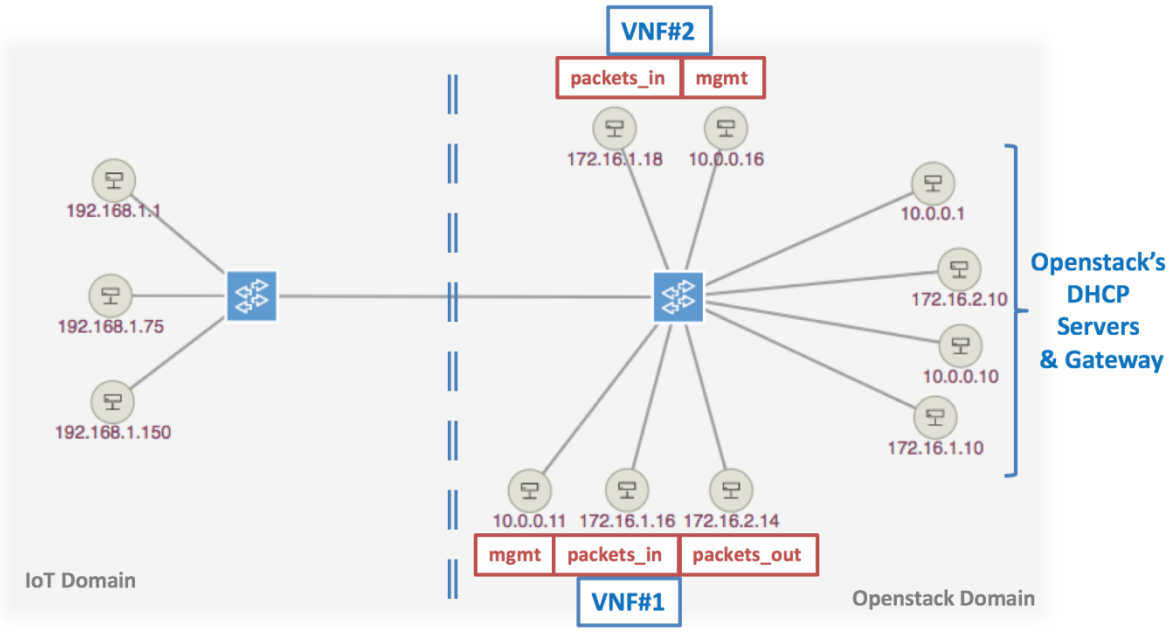

Figure 16 SDN Topology View

In our current setup, we have integrated the ONOS SDN controller with networking component of Openstack in order to enable a high level of flexibility in terms of L2 traffic management between different VNFs. Figure 16 shows the SDN topology view.

5.4 IoT Controller

The IoT controller implementation is in charge to provide IoT operations at high-level of abstraction in order to carry them out over the IoT devices through specific IoT communication protocols. Figure 17 shows the main elements used in the current implementation. In this case, the implementation has been performed in *python* from scratch.

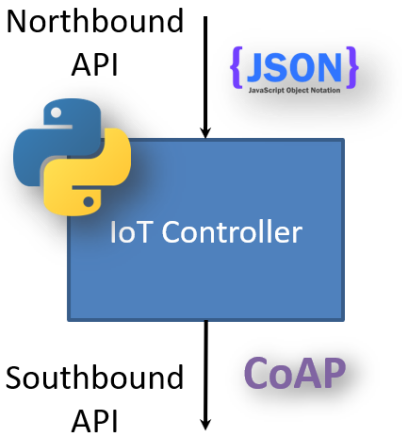

Figure 17 IoT controller implementation elements

The *Northbound REST API* provides different *HTTP-JSON* endpoints in order to manage different aspects of the IoT devices like:

- **IoT devices meta-information:** It provides information regarding the IoT devices managed by the IoT controller, (e.g. Network, operating system, resources…).
- **IoT device management:** It allows performing command and control over the IoT devices like power management operations or authentication. By the time being, it is envisaged to allow performing the following operations:
    - **Turn off:** The IoT device could be turned off remotely by the IoT controller.
    - **Reset:** The IoT device could be reset remotely by the IoT controller.
    - **Bootstrapping:** The IoT controller could request a re-authentication process to the IoT device.

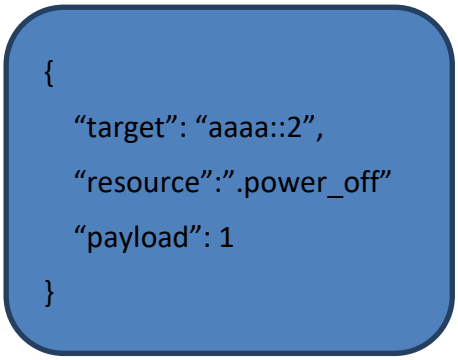

Figure 18 Northbound API JSON input example

Figure 18 provides an example of the *JSON* input for the *Northbound API* device management endpoint. As can be observed, it is specifying a concrete IoT device as target, identified by the ipv6 address, as well as the resource to activate and a payload usually used to include extra information like flags. Once the IoT controller receives the command and control request, it parses the request and it carry out the IoT control over the IoT device through the *Southbound API*.

The *Southbound API* protocol is in charge of performing the communication between the IoT controller and the IoT devices using specific IoT protocols, depending on the IoT target. To this aim, the specific data parsed in the request is transformed into an interaction according to the specific IoT protocol. In this case the implementation employs *CoAP* as IoT protocol with COAP-based devices, (but other IoT protocols could be added in the future). The queries are transformed on CoAP GET/PUT messages in order to request/modify some IoT information or behavior in COAP-based devices.

6 CONCLUSIONS

This document provides a first overview of the Security Orchestrator component within the ANASTACIA framework. It thoroughly describes the inner core functionalities of the orchestrator and its different interactions with other components. Then we discussed the different SDN, NFV and IoT orchestration strategies, which are supported by the Security Orchestrator.

These security enforcement capabilities are planned to be shown in a set of use cases defined in D6.2. So far, from the Security Orchestrator point of view, the MEC.3 use case has been already enabled. The rest of the use cases are soon to be deployed and tested.

Finally, we discussed the technical tools that allow the Security Orchestrator to provide an on-demand security policy enforcement service called Security as a Service (SECaaS) for the ANASTACIA framework.

7 REFERENCES

[1] ANASTACIA D1.1: https://seafile.gruppoitaleaf.com/f/8d46d4741f/

[2] MMT Manual https://seafile.gruppoitaleaf.com/f/ee76a0594d/

[3] ANASTACIA D6.2: https://seafile.gruppoitaleaf.com/f/cc3750b504/

[4] Open Source Mano Wiki: https://osm.etsi.org/wikipub/index.php/OSM_Release_THREE

[5] ONOS Website: https://onosproject.org/