ANASTACIA

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

D3.1

Initial Security Enforcement Manager Report

This deliverable presents the first results of ANASTACIA Task 3.1, which aims to manage the first stage of the enforcement of security requirements established in high-level terms, i.e. security policies, through the ANASTACIA architectural components. Such a first stage is intended to perform the policy refinement process responsible for mapping and translating security policies defined for the flows and E2E communications, to a collection of security properties to be deployed for dealing with security aspects required by objects without altering their normal operations.

Distribution level	PU
Contractual date	31.03.2018 [M15]
Delivery date	10.04.2018 [M16]
WP / Task	WP3 / T3.1
WP Leader	UMU
Authors	Alejandro Molina Zarca, Jorge Bernal Bernabé, Antonio Skarmeta (UMU), Khettab Yacine (AALTO), Belabed Dallal (THALES)
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

Index of tables

PUBLIC SUMMARY

This deliverable describes the first outcomes of task 3.1, which is in charge of designing and developing algorithms, protocols & mechanisms required for the operations of the Security Interpreter of the ANASTACIA architecture.

The main objective of the task is to carefully behold the interactions among IoT objects and the ANASTACIA architecture components in order to ensure that security requirements are met in an end-to-end fashion. Those security requirements are established in high-level terms, namely in the form of policies affecting all or a selected set of objects, or even defining a global desire to defend privacy or other security aspects. This task will also develop an inference engine to enable security policy analysis. Once a policy is defined, configured, and enforced at the control plane of the system, the inference engine will enable reasoning about the policy, including its re-adjustment when deemed appropriate by the system. The Policy Interpreter (in the past Enforcement Manager) will be in charge of mapping policies defined for the flows and E2E communications to a collection of security properties to be deployed for dealing with security aspects required by objects without altering their normal operations.

In this sense, this particular deliverable is a first report regarding the Security Policy Interpreter module of the ANASTACIA framework, including its goals, design (such as interfaces, processes, relationships within the rest of Architectural components), main features, as well as the latest advances in its development.

1 INTRODUCTION

1.1 AIMS OF THE DOCUMENT

This document is part of ANASTACIA WP3 "Policy Enforcement and Run Time Enablers" which aims to design and develop algorithms, protocols & mechanisms that form the intelligence of the Policy Interpreter component (a.k.a. Security Enforcement Manager), the ANASTACIA security orchestrator and the Security Enforcement Enablers. Provide effective co-ordination between various and heterogeneous policy nodes by specifying (within the architecture) constraints and trade-offs at the micro level, to create robustness, efficiency and performance at the policy. Explore the opportunities that NFV and SDN jointly offer in intelligently coping with security threats against IoT services and enable the orchestration of network and cloud resources in a security policy-driven fashion.

Concretely, this deliverable is scoped in Task 3.1 of WP3, which aims to carefully behold the interactions among IoT objects and the ANASTACIA architecture components in order to ensure that security requirements are met in an end-to-end fashion. Those security requirements are established in high-level terms, namely in the form of policies affecting all or a selected set of objects, or even defining a global desire to defend privacy or other security aspects. This task will also develop an inference engine to enable security policy analysis. Once a policy is defined, configured, and enforced at the control plane of the system, the inference engine will enable reasoning about the policy, including its re-adjustment when deemed appropriate by the system.

The Policy Interpreter (former Security Enforcement Manager) will be in charge of mapping policies defined for the flows and E2E communications to a collection of security properties to be deployed for dealing with security aspects required by objects, without altering their normal operations. Security policies will be enforced by the SDN controllers and the NFV MANO orchestrator at the Control Plane of the system. The management/orchestration of the security policies across the different components of the ANASTACIA architecture is carried out by the ANASTACIA Security Orchestrator, defining close coordination between this task and the other two tasks of WP3.

The main goal of this particular deliverable is to provide the first outcomes regarding the Security Policy interpreter module of the ANASTACIA framework, including the design, interfaces, features as well as the first advances in the proposed development. The final results regarding the enforcement process and the policy refinement will reported in deliverable D3.4 Final Security Enforcement Manager Report, in month 33 of the project lifetime.

This document is structured as follow: Section 2 provides a state of the art of current policy refinement solutions, techniques and related solutions. Section 3 gives an overview of the Anastacia architecture, contextualizing the policy interpreter in the ANASTACIA framework. Section 4 is the core of the deliverable, since it defines the policy enforcement process, intended to translate the high-level policy intents or "desires" to specific configurations enforceable in the underneath infrastructure (either physical or virtual system). Section 5, describes the first implementation of the policy interpreter component. Section 6 oversees the policy refinement and enforcement in one of the main identified scenario. Finally, section 7 concludes this deliverable.

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.

1.3 Revision History

Version	Date	Author	Description
0.1	05.02.2018	Alejandro Molina Zarca and Jorge Bernal Bernabé (UMU)	Table of contents
0.2	12.02.2018	Alejandro Molina Zarca and Jorge Bernal Bernabé (UMU)	First contributions across different sections
0.3	26.02.2018	Alejandro Molina Zarca and Jorge Bernal Bernabé (UMU)	Detailed definition of Section 2 and 3
0.4	07.03.2018	Khettab Yacine (AALTO)	Security Orchestrator Interfaces and implementation.
0.5	12.03.2018	Alejandro Molina Zarca and Jorge Bernal Bernabé (UMU)	Policy Interpreter Interfaces and implementation.
0.6	15.03.2018	Alejandro Molina Zarca and Jorge Bernal Bernabé (UMU)	Detailed Section 6 contribution.
0.7	21.03.2018	Belabed Dallal (THALES)	Security enabler provider interfaces and implementation.
0.8	23.03.2018	Alejandro Molina Zarca and Jorge Bernal Bernabé (UMU)	Revision and conclusions
0.9	28.03.2018	Alejandro Molina Zarca and Jorge Bernal Bernabé (UMU), Belabed Dallal (Thales)	Update from Thales. Editing process. Document ready for internal review
1.0	07.04.2018	Alejandro Molina Zarca and Jorge Bernal Bernabé (UMU)	Applied suggestions and comments from the internal review process performed by Diego Rivera (Montimage)

1.4 Acronyms and Definitions

Acronym	Meaning
MSPL	Medium-level Security Policy Language
HSPL	High-level Security Policy Language
PDP	Policy Decision Point
PEP	Policy Enforcement Point
SEC	Security Enforcement Manager
CIM	Common Information Model
SPL	Security Policy Language (SPL)
SDL	System Description Language
NSF	Network Security Functions
BMS	Building Management Systems
CPS	Cyber Physical System
CRUD	Create, Read, Update, and Delete
DSPS	Dynamic Security and Privacy Seal
IoT	Internet of Things
MANO	Management and Orchestration
MEC	Mobile (Multi-access) Edge Computing
NFV	Network Function Virtualization
SDN	Software Defined Networking
PSA	Personal Security Application
M2L	Medium to Low

2 State Of The Art in Policy refinement

This section presents a state of the art for policy refinement and main related technologies. Here it will be illustrated the main identified security policy refinement processes, focusing on how a high abstraction level policy model is refined in a medium/low abstraction level policy model.

2.1 xCIM-SDL/SPL policy refinement

Common Information Model (CIM) [1] is the main DMTF standard which provides a common definition of management-related information independent of any specification. The model defines concepts for authorization, authentication, delegation, filtering, and obligation policies. However, for an information model to be useful, it has to be mapped into some specification and for this purpose, CIM models are not suitable by themselves, due to the huge amount of classes that composes it. xCIM High-level Security Policy Language (SPL) defined in [2], allows to the administrator the definition of security policies using a friendly language, near to the spoken English. It also has an internal format which is a language for formal modelling and low-level abstraction that is oriented to developers. On the other hand, xCIM System Description Language (SDL) is a sub-model that represents the medium level abstraction representation for system description. Whereas xCIM Security Policy Language (SPL) is a sub-model of CIM that represents the medium/low level abstraction representation for security policies. Both in scope of POSITIF [3] and DESEREC [4] European projects.

Figure 1: Policy Refinement, shows the translation from the high-level specification to low-level rules specified by a language based CIM-Policy Information Model (i.e. xCIM-SPL or internal format). The tools for policy refinement and manipulation provide a Policy Console and a Policy Translation Service which permit the definition and refinement of high-level rules. These tools reduce the errors and permit additional checks. Regarding the translation process, Figure 1: Policy Refinement shows how the process is based on the direct transformation of the SPL elements to xCIM-SPL elements. Due to the lack of information provided by the natural human concepts, the authors use templates to fill the required information, generating finally the final xCIM-SPL result. The tasks Transform and Complete showed in the figure, are deployed by XML Style Sheets (XSL) transformation because all documents (i.e. templates, xCIM-SPL definitions and SPL definition) are represented by XML.

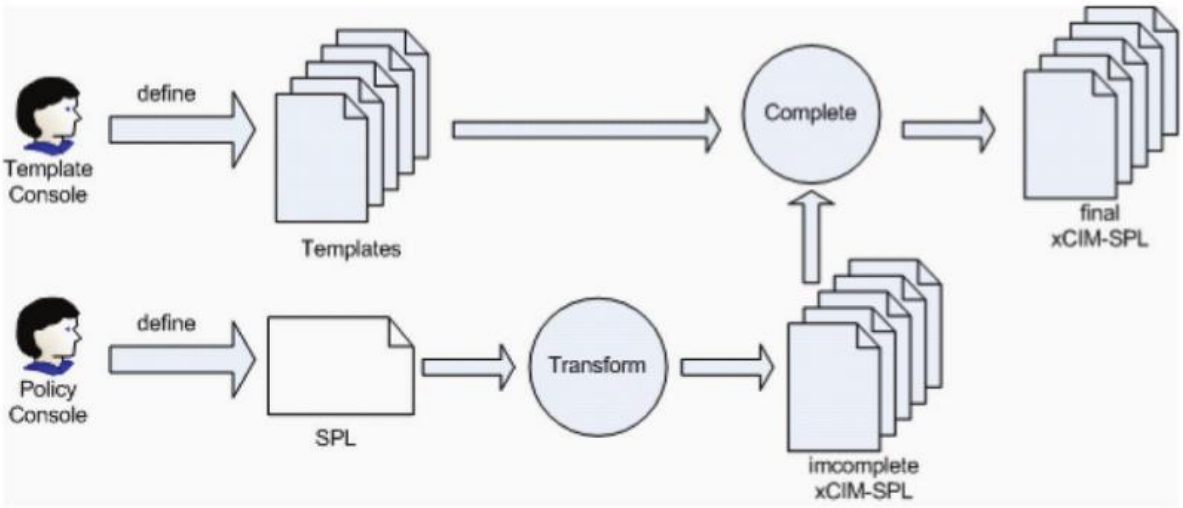

Figure 1: Policy Refinement

The authors provide also a Policy Console to ease the definition, transformation, and manipulation of security policies.

2.2 SECURED – POLICY REFINEMENT

High-level Security Policy Language (HSPL) and the Medium-level Security Policy Language (MSPL) are two abstractions defined within the European SECURED project [5] to specify security policies based on the capability concept. A capability is the ability to provide a specific security functionality by a security enabler or component. HSPL is though for coarse-graned policies, allowing to define general policies to non-technical users, being independent on the underlaying technologies. On the other hand, MSPL allows to specify information close to the implementation, but still technology independent. Thus, HSPL/MSPL extend and improve the idea exposed on xCIM-SPL/SDL of two level of device-independent languages, a lower dependent one and the use of capabilities.

2.2.1 HSPL to MSPL

Figure 2: H2M Refinement shows the main components involved in the refinement process [6]. The first step consists on identifying the required capabilities of the HSPL policy. Once identified the capabilities, it is necesary to identify the Personal Security Application (PSA). The PSA is can be defined as a hardware or software component able to enforce the identified capabilities. If there is not available any component implementing the required functionalities the process will return a non-enforzable analysis, otherwise, it will be performed a translation of an HSPL policy into MSPL policies, also including a service graph indicating which PSA could take care of which MSPL policy.

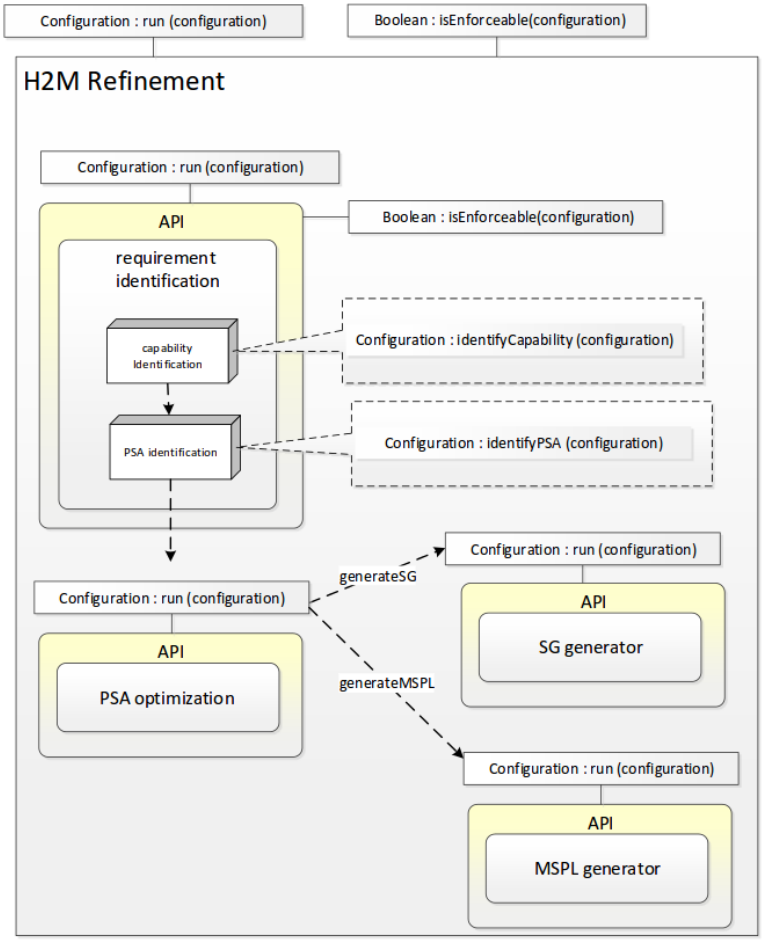

Figure 2: H2M Refinement

2.2.2 MSPL to low-level configuration refinement

Since MSPL is still device-independent, it must be translated into a specific security configuration for a specific Personal Security Application (PSA). Figure 3: M2L Refinement shows the main interconnections in order to achieve that goal. A coordinator requests a M2L translation to the M2L service, among a MSPL policy and a PSA specific configuration, indicating the PSA id. To support a wide set of low-level security controls, the translation is designed to be multi-device (e.g. netfilter/iptables or PF for a stateful firewall). The proposed approach uses a M2L (Medium-to-Low) plugin repository which contains a set of plugins that implements the translation between MSPL and the specified configuration. The plugins then, can be loaded by the M2L service in order to get the PSA configuration deppending on the PSA id specified.

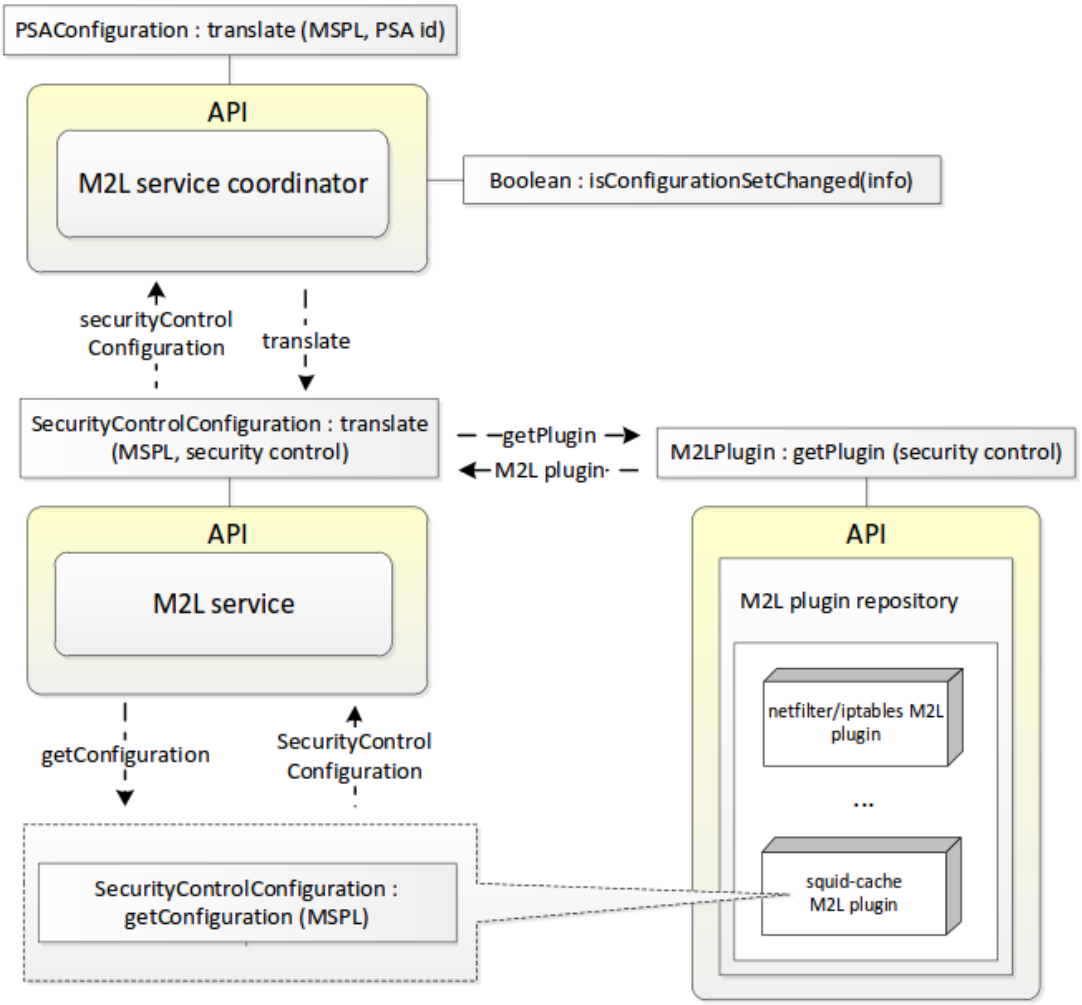

Figure 3: M2L Refinement

It is important to highlight that progress is currently being made in these research lines. For instance, the Interface to Network Security Framework (I2NSF) IETF group reuses and extends the capability concept. Actually, I2NSF is described on RFC 8329 [7] which details the framework for Interface to Network Security Functions (I2NSF) and defines a reference model (including major functional components) for I2NSF. Network Security Functions (NSFs) are packet-processing engines that inspect and optionally modify packets traversing networks, either directly or in the context of sessions to which the packet is associated. defining a reference model for I2NSF, as well as a flow-based capability negotiation concept. The RFC bases the network security function capability model in the one presented on [8].

3 POLICY INTERPRETER IN THE ANASTACIA ARCHITECTURE

The ANASTACIA system model is structured as a set of layers that provide a broad view of the framework and stand out its integration within IoT infrastructures. ANASTACIA is envisioned as a framework integrated on top of an IoT infrastructure where IoT devices, physical and virtual network elements interact in the **Data Plane**. On top of that, the **Control Plane** manages the computing, storage, and networking resources in the Data Plane by leveraging SDN controllers, NFV orchestration platforms, and IoT controllers.

*Figure 4: ANASTACIA architecture***Errore. L'origine riferimento non è stata trovata.** represents the high-level view of the ANASTACIA framework, which extends the ANASTACIA system model by expanding the functions of the ANASTACIA core. A first version of the ANASTACIA framework has been described in [9][10]. The **Autonomic Plane** includes the components that provide the ANASTACIA framework with its intelligence and dynamic behaviour. This plane can be divided into three sub-planes, which carry out specific activities within the framework:

- The **Security Orchestrator Plane** organizes the resources that support the Enforcement Plane, carrying out activities such as the transformation of security properties to configuration rules and aligning the security policies defined by the security interpreter with the provisioning of relevant security mechanisms. It has the whole vision of the underlying infrastructure and the resources and interfaces available at the Security Enforcement Plane.
- The **Security Enforcement Plane** connects the ANASTACIA core with the IoT Platform (Data and Control planes), managing the interactions among objects and components for the enforcement of the security policy defined at the User Plane. This plane supports the enforcement of configurations and reactions triggered by the Security Orchestrator Plane, in order to preserve the expected security level. At this plane, the agents that support the monitoring of IoT devices or the enforcement of reactions are instantiated, either if they are operating on remote or directly attached to the device.
- The **Monitoring and Reaction Plane** connects to the IoT Platform through the Security Enforcement Plane in order to collect security-focused information related to the system behaviour. At this plane, intelligent data-driven automated and contextual monitoring of activities at embedded devices, legacy systems and IoT devices by retrieving signals, event logs, traces, heartbeats signals, status reports or operational information. This plane also evaluates the fulfilment of the security policy by checking security models or threats signatures, detecting anomalies and creating reactions to mitigate such anomalies, in terms or reconfigurations and alerts to system administrators.

Additionally, on top of the architecture, the User Plane and the Seal Management Plane interact with the Autonomic plane:

- The **User Plane** includes interfaces, applications and tools that help system administrators to manage the IoT platform through the ANASTACIA framework. For example, at this plane system admins can edit the security policies that govern the underlying IoT platform.
- The **Seal Management Plane** oversees providing users with a real-time indicator of the overall security level.

Since this deliverable is focused on the security enforcement management, following subsections aims to illustrate a description of the main components involved on the Security Orchestration Plane, as well as their interactions and interfaces.

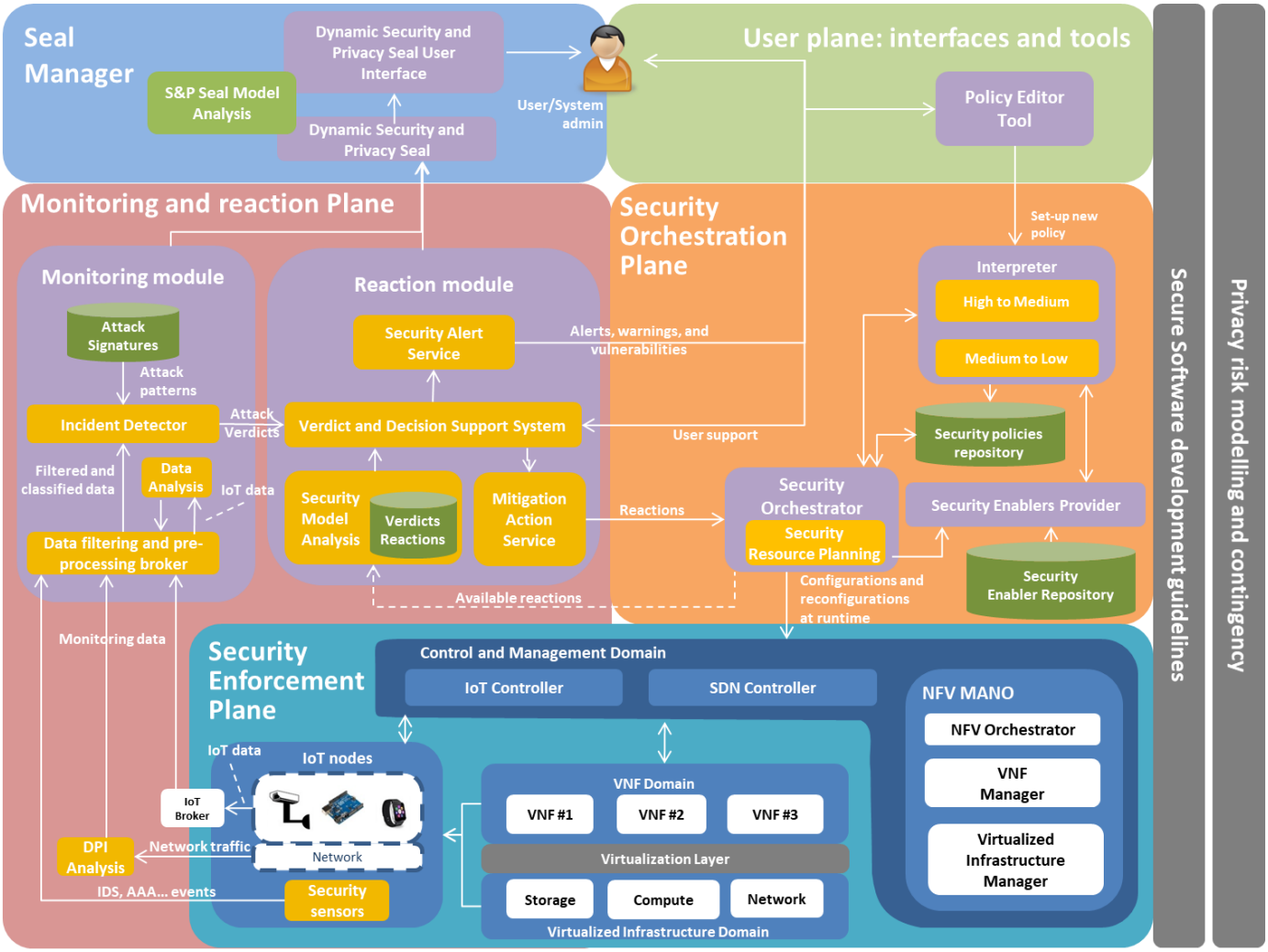

Figure 4: ANASTACIA architecture

3.1.1 ANASTACIA Security Orchestration Plane

The security orchestration Plane is being designed and implemented mainly in the scope fo WP3. The security orchestration plane contains the components in charge of interpreting the security policies set-up from the user plane as well as carrying out the enforcement of security policies and execution of reactions. Following subsections explain those main activities and the components involved on them.

3.1.1.1 Security policy set-up activity

This is the initial process triggered once a high-level/medium-level security policy has been defined by the user. In this point the policy must be interpreted by the platform in order to be enforced. The interpretation of the security policy refines a high-level security policy from a non-technical policy language to a medium-level one. In order to achieve these goals, the **Security policy set-up** activity relies on the following components:

- An **Editor** at the User plane that can be used by the User/System admin to set the Security policy to be enforced in the IoT platform.
- An **Interpreter** in the Security Orchestration Plane that will transform the Security policy (closer to a human readable policy) to a machine-readable policy that is able to represent lower configurations parameters. The security policy language and model is defined in the scope of WP2, an concretely in Task2.1.

- A **Security Enabler Provider** in the Security Orchestration Plane, that is able to identify the security enablers which can provide specific security capabilities, so to meet the security policies requirements.
- A **Security orchestrator** in the Security Orchestration Plane is responsible for selecting the security enablers to be used in the policy refinement process.

3.1.1.2 Security policy orchestration activity

Once the medium-level of abstraction policy has been obtained, it is necessary to enforce the controls specified within it. To orchestrate the selected IoT/SDN/NFV-based security enablers, appropriate interactions with the relevant management modules are required. Thus, the **Security policy orchestration** activity requires the following components:

- A **Security orchestrator** that has the whole vision of the subjacent infrastructure and is able to trigger the enforcement of the defined policies using the corresponding configurations or tasks obtained during the policy refinement process.
- The elements contained in the Security Enforcement Plane, including the **IoT controllers, NVF orchestrators, SDN controllers** and, in general, all the elements enabling the configuration of the resources offered by the IoT devices and the physical/virtual network elements in the underlying infrastructure, as shown in Figure 4: ANASTACIA architecture.

3.1.1.3 Main components

This section provides a more detailed description by the components involved on the Security Orchestration Plane. Specifically, they are the Policy Editor Tool (Table 1. Policy Editor Tool description), The Policy Interpreter (Table 2. Interpreter description), the Security Enabler Provider (Table 3. Security Enabler Provider description) and the Security Orchestrator (Table 4. Security Orchestrator description), providing an overview of functionalities, subcomponents, sources and consumers, the activities they are involved and the previous available assets.

Table 1. Policy Editor Tool description

Interpreter	
Function	The Policy Editor Tool allows administrators to model security policies with a high/medium-level of abstraction.
Subcomponent	(Not applicable)
Sources	User/Admin
Consumers	Policy Interpreter
ANASTACIA activities involved	Security Policy Set-up
Available assets	Web administration templates/xCIM Policy console.

Table 2. Interpreter description

Interpreter	
Function	The Policy Interpreter is in charge of performing the refinement processes from High-level Security Policy (HSPL) to Medium-level Security Policy (MSPL) and from MSPL to Enablers/VNFs configuration or tasks.
Subcomponent	High to Medium Service (HSPL to MSPL) Medium to Lower Service (MSPL to specific configurations/tasks)
Sources	Policy Editor Tool Orchestrator Security Enabler Provider
Consumers	Orchestrator
ANASTACIA activities involved	Security Policy Set-up Security Orchestration
Available assets	SECURED FP7 Security Policy Module **Errore. L'origine riferimento non è stata trovata.**

Table 3. Security Enabler Provider description

Security Enablers Provider	
Function	The Security Enabler Provider is able to identify the list of security enablers which can provide specific security capabilities to meet the security policies requirements. Besides, this component will be endowed with an interface for delivering security M2Lplugins for M2L refinement process, which, in turn, will allow translating policies from MSPL to Low-level configurations.
Subcomponent	(Not applicable)
Sources	Security Policy Interpreter
Consumers	Security Policy Interpreter
ANASTACIA activities involved	Security Policy Set-up Security Orchestration
Available assets	Beta implementations of some few Security M2Lplugins, that allows translating from policies specified in MSPL to low level configuration, are already available in the EU SECURED project [https://github.com/SECURED-FP7/secured-spm]. Including for instance M2L-IpTables plugin to generate iptables using as baseline a filtering policy specified in MSPL.

Table 4. Security Orchestrator description

Security Orchestrator	
Function	The ANASTACIA Security Orchestrator oversees orchestrating the security enablers according to the defined security policies. To this aim, it is involved in the selection of the security enablers accounting for their security capabilities, the available resources in the underlying infrastructure, and the policies requirements. Once received the configuration of the security enablers by the Policy Interpreter, the Security Orchestrator interacts with relevant SDN/NFV/IoT control and management components, so to enforce the required features in the IoT devices and in the physical/virtual network elements of the underlying infrastructure.
Subcomponent	Security Resource Planning. This submodule is in charge of efficiently selecting the available security enablers to meet the required MSPL security policies. This submodule will define and develop appropriate strategies to enforce security by exploring the SDN, NFV, and IoT technologies, accounting for the available resources in the underlying infrastructure and the policies requirements. In this vein, it provides functions like optimal path selection, load balancing, traffic rerouting, etc.
Sources	Interpreter Security Enablers Provider Reaction
Consumers	Security Enforcement Plane (Control and Management Domain components) Interpreter Monitoring Module Reaction Module
ANASTACIA activities involved	Security Policy Set-up Security Orchestration Security Monitoring Security Reaction
Available assets	To orchestrate the configuration of the security enablers over the Security Enforcement Plane, the Security Orchestrator can leverage standardized interfaces, protocols, and available open source software libraries to interact with relevant management components: SDN controllers, such as ONOS (http://onosproject.org/) or OpenDayLight (https://www.opendaylight.org/); and NFV MANO modules, e.g., OpenBaton (https://openbaton.github.io/) or OSM (https://osm.etsi.org/).

3.2 INTERFACES FOR POLICY REFINEMENT

This section describes the main interfaces related to the Policy refinement and enforcement process which is one of the main goal of this deliveralbe. Thus, this section delves into the main interactions between the Policy Interpreter (main component of Taksk 3.1) and the rest of the components in the Orchestration Plane.

The following tables describe the interfaces involved in the policy set-up, comprising interfaces which are focused on policy refinement and policy translation:

- H2MI (Table 5. Policy Editor Tool -> Interpreter H2M (H2MI)),
- M2LI (Table 6. Policy Editor Tool -> Interpreter M2L (M2LI))

Interfaces focused on the security enablers management for policy refinement and policy translation.:

- SEPSEI (Table 7. Interpreter -> Security Enabler Provider Security Enabler (SEPSEI))
- SEPPI (Table 8. Interpreter -> Security Enabler Provider (SEPPI))

Interfaces focused on security policy management:

- H2MPRI (Table 9. Policy Interpreter -> Policy Repository Interface (H2MPRI))
- M2LPRI (Table 10. Policy Interpreter -> Policy Repository Interface (M2LPRI))

UPSPRI (

- Table 13. Security Orchestrator -> Policy Repository Interface (UPSPRI))

Interfaces strong related to the policy enforcement:

M2EI (

- Table 11. Policy Interpreter -> Security Orchestrator (M2EI))
- SMI (Table 12. Policy Interpreter -> Security Orchestrator (SMI))

Table 5. Policy Editor Tool -> Interpreter H2M (H2MI)

High to Medium interface (H2MI)		
Description	The interface allows to request a policy refinement from a High-level Security Policy (HSPL) to a Medium-level Security Policy (MSPL).	
Component providing the interface	Policy Interpreter	
	Input data	HSPL policy
	Output Data	MSPL policy
Consumer components	Policy Editor Tool	

Pre-conditions	To define the HSPL policy using a specific format. To deploy the Security Enabler Provider in order to obtain the candidates security enablers list. To deploy a system model repository, in order to define associations (for example, `device:address`). To deploy the Security Policies repository in order to maintain a registry of policy status.
Post-conditions	(Not applicable)
ANASTACIA activities involved	Security Policy Set-up

Table 6. Policy Editor Tool -> Interpreter M2L (M2LI)

Medium to Lower interface (M2LI)		
Description	The interface allows to request a policy refinement from a Medium-level Security Policy (MSPL) to a specific enabler configuration/task	
Component providing the interface	Policy Interpreter	
	Input Data	MSPL policy
	Output Data	Enabler configuration/task
Consumer components	Policy Editor Tool Security Orchestrator	
Pre-conditions	To deploy the Security Enabler Provider in order to obtain the chosen security enabler plugin.	
Post-conditions	(Not applicable)	
ANASTACIA activities involved	Security Policy Set-up Security Orchestration	

Table 7. Interpreter -> Security Enabler Provider Security Enabler (SEPSEI)

Security Enabler Provider Security Enabler Interface (SEPSEI)		
Description	The interface allows requesting the required security enablers for specific capabilities.	
Component providing the interface	Security Enabler Provider	
	Input Data	List of capabilities
	Output Data	List of candidate security enablers
Consumer	Policy Interpreter	

components	
Pre-conditions	There must be a correspondence between capabilities and security enablers.
Post-conditions	(Not applicable)
ANASTACIA activities involved	Security Policy Set-up Security Orchestration

Table 8. Interpreter -> Security Enabler Provider (SEPPI)

Security Enabler Provider Plugin Interface (SEPPI)		
Description	The interface allows to request the plugin that translates the MSPL file to low-level configuration.	
Component providing the interface	Security Enabler Provider	
	Input Data	Enabler name
	Output Data	Enabler translator plugin
Consumer components	Policy Interpreter	
Pre-conditions	There must be a correspondence between the security control name and the code location in the Security Enabler Provider.	
Post-conditions	(Not applicable)	
ANASTACIA activities involved	Security Policy Set-up Security Orchestration	

Table 9. Policy Interpreter -> Policy Repository Interface (H2MPRI)

H2M Policy Repository Interface (H2MPRI)		
Description	The interface allows to store in the policy repository the correspondence among HSPL and MSPL policies.	
Component providing the interface	Policy Repository (manager)	
	Input Data	HSPL,MSPL
	Output Data	Acknowledgement
Consumer components	Policy Interpreter (h2mservice)	
Pre-conditions	-	
Post-conditions	(Not applicable)	

ANASTACIA activities involved	Security Policy Set-up

Table 10. Policy Interpreter -> Policy Repository Interface (M2LPRI)

M2L Policy Repository Interface (M2LPRI)		
Description	The interface allows to store in the policy repository the correspondence among MSPL and specific security enabler configuration.	
Component providing the interface	Policy Repository (manager)	
	Input Data	MSPL, Enabler configuration
	Output Data	Acknowledgement
Consumer components	Policy Interpreter (m2lservice)	
Pre-conditions	-	
Post-conditions	(Not applicable)	
ANASTACIA activities involved	Security Policy Set-up Security Orchestration	

Table 11. Policy Interpreter -> Security Orchestrator (M2EI)

Security Orchestrator MSPL Enforcement Interface (M2EI)		
Description	The interface allows to request a MSPL enforcement.	
Component providing the interface	Security Orchestrator	
	Input Data	MSPL + Candidate Security Enablers
	Output Data	Enforcement result
Consumer components	Policy Interpreter Policy Editor Tool	
Pre-conditions	The Policy interpreter must be deployed in order to translate from MSPL to a selected security enabler configuration.	
Post-conditions	(Not applicable)	
ANASTACIA activities involved	Security Policy Set-up Security Orchestration	

Table 12. Policy Interpreter -> Security Orchestrator (SMI)

Security Orchestrator System Model Interface (SMI)		
Description	The interface allows to request a system model of the underlaying architecture.	
Component providing the interface	Security Orchestrator	
	Input Data	System model key name
	Output Data	System model values
Consumer components	Policy Interpreter	
Pre-conditions	It must exist a system model correspondence among element identifications and element information.	
Post-conditions	(Not applicable)	
ANASTACIA activities involved	Security Policy Set-up Security Orchestration	

Table 13. Security Orchestrator -> Policy Repository Interface (UPSPRI)

Update Policy Status Policy Repository Interface (UPSPRI)		
Description	The interface allows to update the status of the security policy.	
Component providing the interface	Policy Repository (manager)	
	Input Data	MSPL identification, Status
	Output Data	Acknowledgement
Consumer components	Security Orchestrator	
Pre-conditions	The MSPL policy must exist.	
Post-conditions	(Not applicable)	
ANASTACIA activities involved	Security Policy Set-up Security Orchestration	

4 POLICY REFINEMENT PROCESSES

ANASTACIA framework is extending and improving the HSPL and MSPL languages proposed in SECURED-FP7 project, as was defined in Anastacia deliverable D2.1. The Anastacia Policy Interpreter – comprising the two separated refinement processes – is being implemented from scratch, redisigning the original proposed translation process defined in SECURED. This change offers two well new differentiated APIs specially devised for the ANASTACIA framework: (1) a refinement process from High-level Security Language (HSPL) to Medium-level security Policy Language (MSPL), corresponding to *h2mservice* API; and (2), a translation from Medium-level Security Policy Language (MSPL) to lower-level configurationsm which will correspond to *m2lservice* API. The processes of HSPL refinement and MSPL translation are detailed below.

4.1 HSPL->MSPL REFINEMENT

Figure 5: HSPL Refinement, shows the main workflow where the *h2mservice* API receives the HSPL policies, performing the refinement in order to generate the MSPL policies. Since the refinement process could require relevant information from the current system status as well as specific Security Enabler knowledge, the figure illustrates additional information inputs. As output of the process, it will generate the correspondent MSPL policies or even a report of non-enforceability if there are not possibilities to enforce the specified policy according to the security enabler information received.

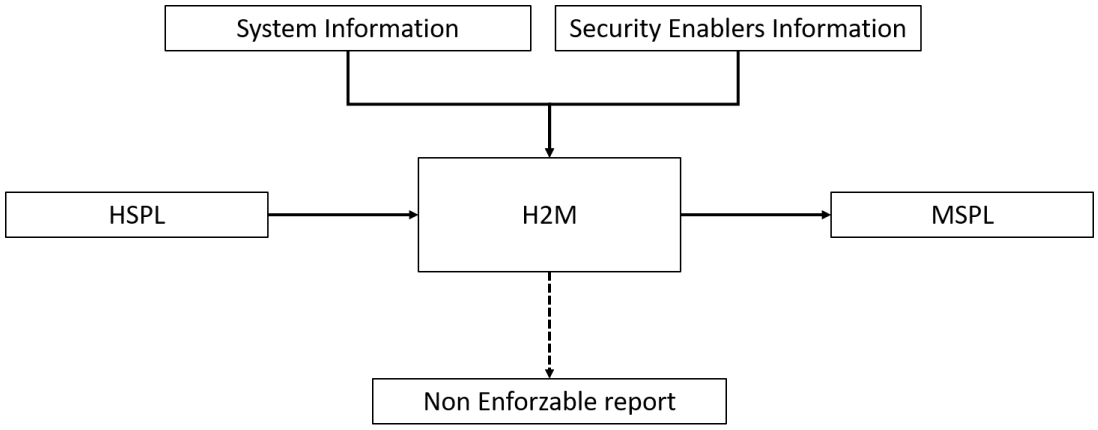

Figure 5: HSPL Refinement

The system information will provide relevant data about the whole infrastructure, providing for instance matchings between identifiers and IP addresses. On the other hand, the Security Enablers information will provide data regarding the available Security Enablers capable to enforce specific capabilities. Regarding the H2M functionality, Figure 6: HSPL Refinement Diagram, shows the workflow for the HSPL to MSPL refinement which corresponds with the following action points:

1. The user defines the HSPL policy using the Policy Editor Tool.
2. The Policy Interpreter receives a refinement request, including the defined HSPL policy.
3. The Policy Interpreter identifies the main capabilities of the policy, for instance, if the HSPL is a filtering policy, the Policy Interpreter will identify FILTERING as the main capability.
4. The Policy Interpreter requests to Security Enabler Provider the list of Security Enablers capable to enforce the security policy using the identified capabilities.
5. The Security Enabler Provider obtains the information regarding the Security Enablers from the Security Enabler repository.

6. The Security Enabler Provider populates a Security Enabler list which includes the Security Enablers capable to provide the specified capabilities.
7. The Security Enabler Provider provides the Security Enablers information to the Policy Interpreter.
8. The Policy Interpreter performs a non-enforceability analysis of the HSPL policies against the Security Enablers provided.

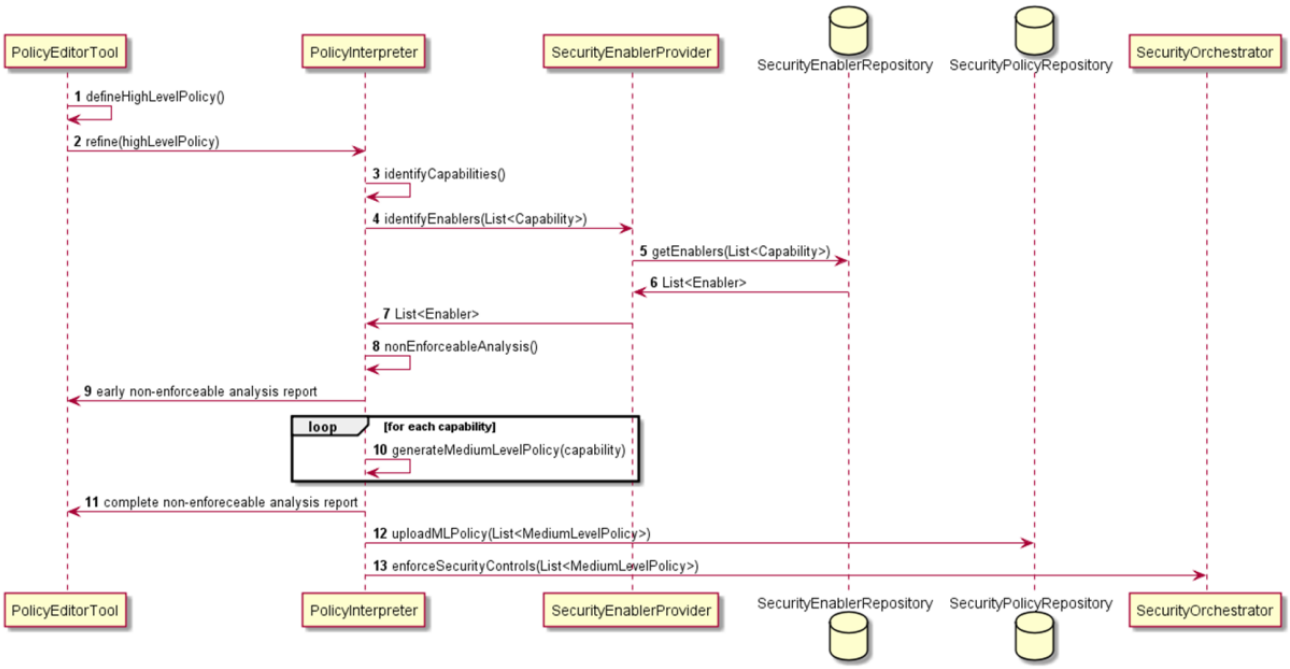

Figure 6: HSPL Refinement Diagram

9. If it is not possible to determine a viable policy enforcement, the Policy Interpreter generates a non-enforceable analysis report.
10. If the security policy can be enforced, the HSPL is refined into a MSPL policy, considering each capability.
11. If there was any problem during the refinement, the Policy Interpreter will provide a non-enforceability analysis report.
12. The refined policy is uploaded to the Security Policy Repository.
13. The refined policy is sent to the Security Orchestrator in order to start the deliberating process.

4.2 MSPL->Low level Translation

Once the Security Orchestrator has received the MSPL refined policies, it will start a deliberation process to determine when, where and how to enforce each security policy. This process will include the decision of what kind of Security Enabler will enforce which MSPL policy. When the Security Orchestrator will make the decision, it will start the process which will take care to perform the translation among MSPL policies and Enablers/VNFs configurations or tasks.

Figure 7: MSPL-Enabler translation, shows the main workflow for the MSPL to lower level translation where it is possible to discern the following points:

1. If the received message with the MSPL policy does not contain a list of candidate security enabler, the Security Orchestrator identifies the main capabilities for the MSPL policy and performs steps 2 to 5. Otherwise it performs directly the step 6 (notice that this step is not reflected in the figure for the sake of clarity).
2. The Security Orchestrator requests the available Security Enablers for those capabilities from the Security Enabler Provider.
3. The Security Enabler Provider requests the information from the repository.

4. The Security Enabler Provider receives the specific Security Enablers information.
5. The Security Enabler Provider sends the list of Security Enablers to be enforced with the requested capabilities.
6. The Security Orchestrator will make the decision regarding what kind of Security Enabler will enforce the MSPL policy.

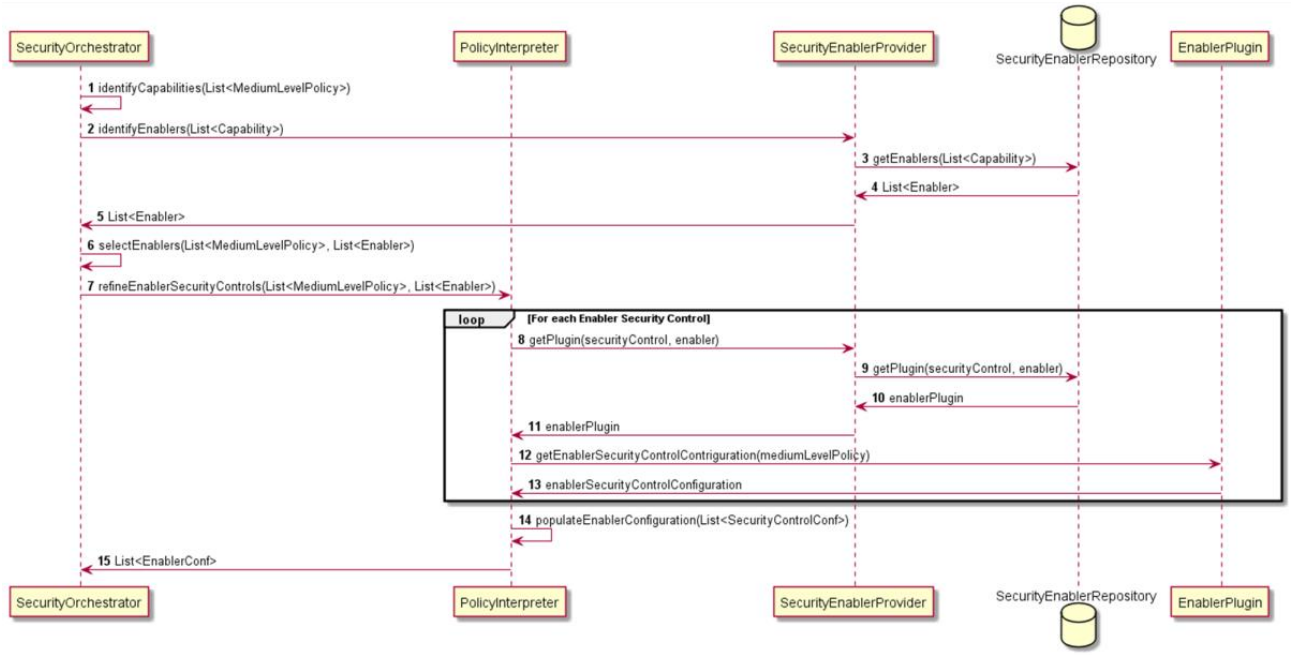

Figure 7: MSPL-Enabler translation

7. The Security Orchestrator will request the MSPL to lower translation to the Policy Interpreter, indicating the specific Security Enabler it wants to use (e.g. “sdn-onos”,”vr-iptables”…)
8. The Policy Interpreter will receive the request and it solicits to the Security Enabler Provider a proper plugin in order to perform the translation.
9. The Security Enabler Provider requests the plugin to the repository.
10. The Security Enabler Provider obtains the specific plugin.
11. The Policy Interpreter receives the plugin software.
12. The Policy Interpreter executes the plugin software, indicating the MSPL policy to translate.
13. When the plugin translation process has finished, the Policy Interpreter obtains the final configurations/tasks.
14. The Policy Interpreter prepares the configuration/tasks to be sent.
15. The Security Orchestrator receives the final configuration/tasks to be enforced.

4.3 POLICY-BASED DEPLOYMENT

This section shows the main workflow for a policy-based deployment. There are contemplated two different approaches. The proactive approach and the reactive one. In the proactive approach, the security policy can be part of a proactive measure where the administrator decides to deploy it, with the aim to prevent some issue or to establish some default behaviour from start-up. On the other hand, the reactive approach allows to deploy a security policy as part of a countermeasure.

4.3.1 Proactive Scenario

Figure 8: Policy-based - Proactive scenario shows the main workflow for a security policy deployment in a proactive scenario. In this case, the process is compounded by the following steps:

1. The Policy Editor tool requests the enforcement of an HSPL policy to the Policy Interpreter.

2. The Policy Interpreter identifies the required capabilities by the received HSPL policy, and it requests the candidate security enablers able to cope with the identified capabilities to the Security Enabler Provider.
3. The Policy Interpreter refines the HSPL policy into a MSPL policy, and it stores both in the policy repository.

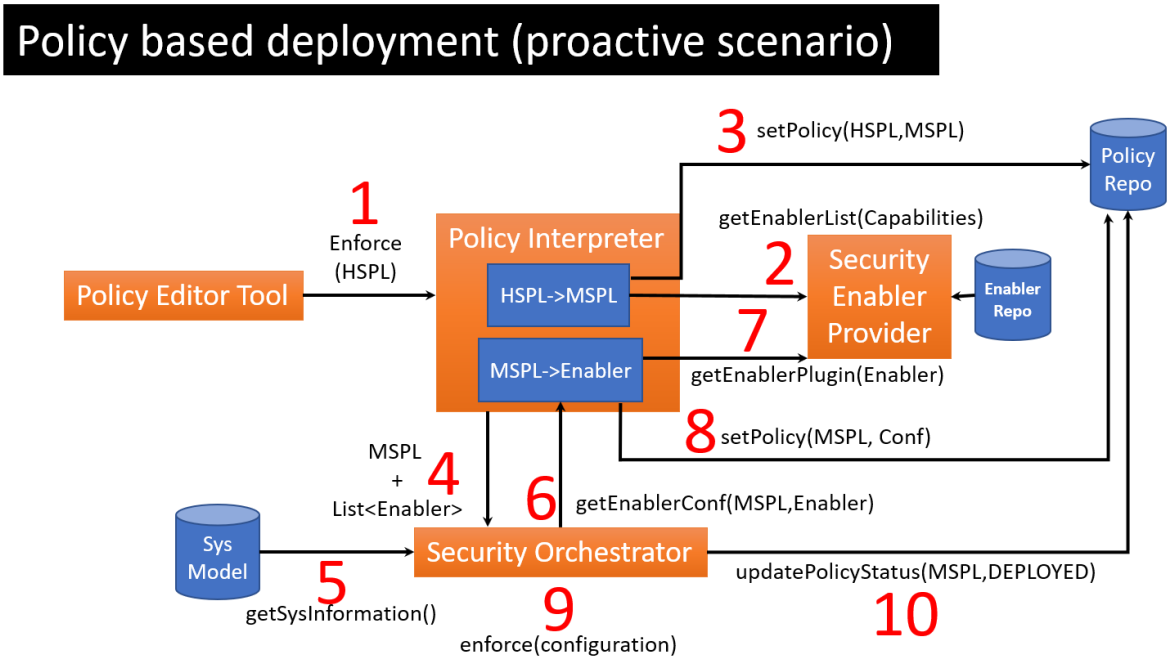

Figure 8: Policy-based - Proactive scenario

4. The Policy Interpreter sends the refined MSPL policy and the obtained candidate security enablers to the Security Orchestrator, by requesting a security enabler proper selection.
5. The Security Orchestrator gets system information in order to decide what is the best security enabler to apply the security policy.
6. The Security Orchestrator sends the MSPL and the selected security enabler to the Policy Interpreter in order to obtain a MSPL translation into specific security enabler configurations.
7. The Policy Interpreter obtains from the Security Enabler Provider the proper plugin according on the received security enabler and performs the MSPL translation, returning the security enabler configuration to the Security Orchestrator.
8. The Policy Interpreter stores the pair MSPL, enabler configuration in the policy repository.
9. The Security Orchestrator performs the policy enforcement through the selected security enabler.
10. The Security Orchestrator updates the security policy with the result obtained in the previous step.

4.3.2 Reactive Scenario

Figure 9: Policy-based - Reactive scenario shows the main workflow for a security policy deployment in a reactive scenario. In this case, the process is compounded by the following steps:

1. The Reaction module gets an MSPL template according to the reaction countermeasure.
2. The Reaction module fills the MSPL template and it sends the instantiated MSPL policy to the Security Orchestrator.
3. The Security Orchestrator obtains system model information if it is required.
4. The Security Orchestrator request a MSPL policy translation to the Policy Interpreter, indicating the chosen security enabler.
5. The Policy Interpreter obtains the proper plugin from the Security Enabler Provider and it performs the translation, returning to the Security Orchestrator the security enabler configuration.

6. The Policy interpreter stores the MSPL, security enabler configuration correspondence in the Policy Repository.

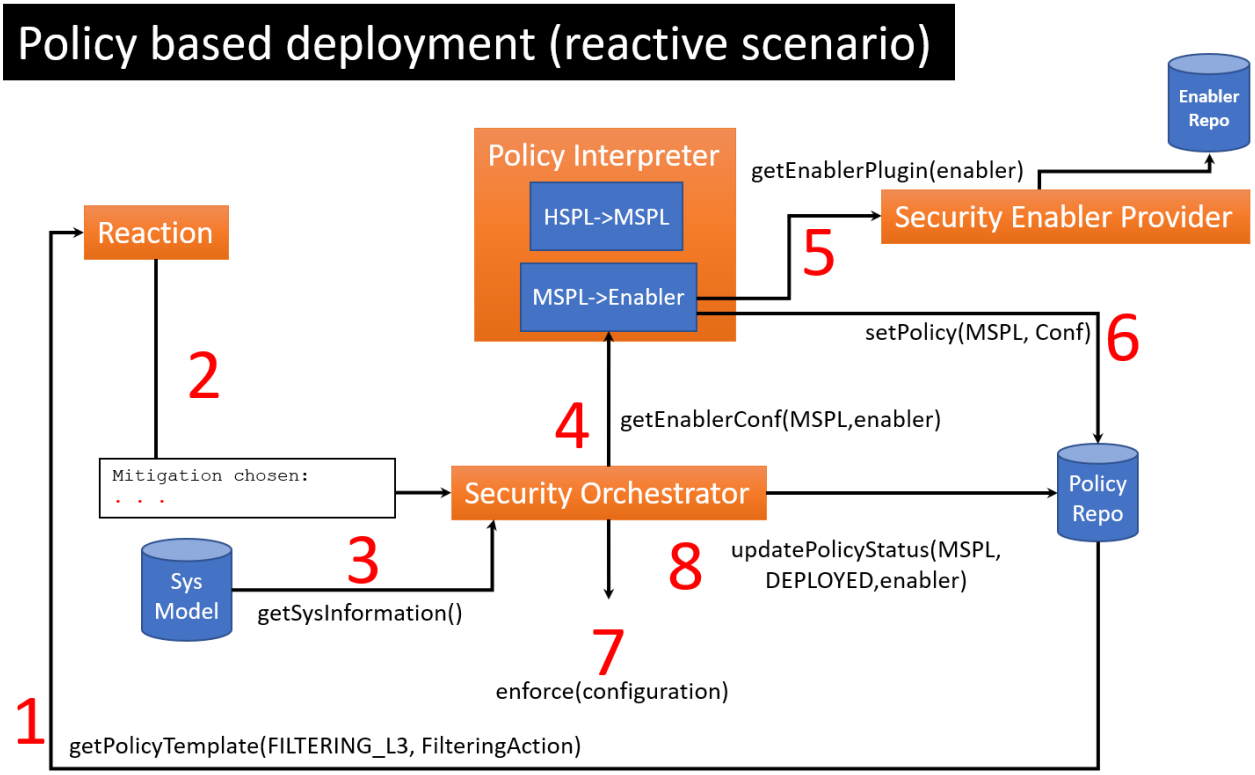

Figure 9: Policy-based - Reactive scenario

7. The Security Orchestrator enforces the received security enabler configuration.
8. The Security Orchestrator updates the policy status on the Policy repository.

5 FIRST POLICY ENFORCEMENT IMPLEMENTATION

5.1 POLICY INTERPRETER IMPLEMENTATION

In order to provide the 2-level policy refinement as designed, a policy interpreter has been developed in python, which is capable of refining HSPL policies into MSPL policies and translating MSPL policies into specific configuration according with the selected technology. The policy interpreter provides two well-diferentiated services that are in charge of each policiy level.

5.1.1 H2MService

The h2mservice implementation is in charge to refine HSPL policies into MSPL policies. Figure 10: h2mservice implementation, shows the main inputs and outputs for the service in charge of the HSPL to MSPL refinement.

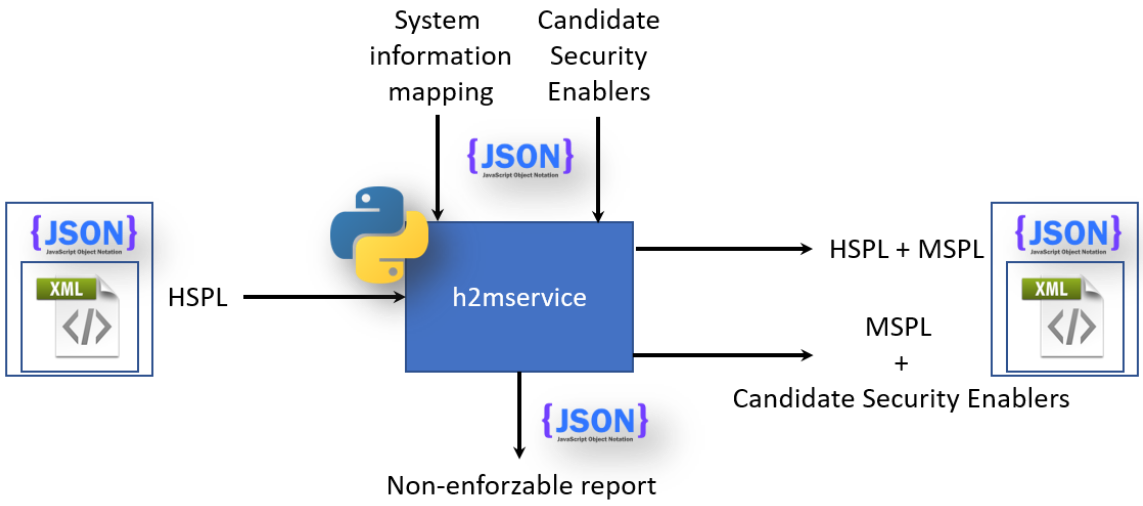

Figure 10: h2mservice implementation

This service provides a *h2mservice REST API* which, through a HTTP POST method, receives a list of HSPL policies in XML, being this list enclosed in a JSON file. In this way we separate the specific parameters of the API from the security policy model. Figure 11: h2mservice input example shows an example of how the HSPL list codified in XML is enclosed into a JSON file, specifiying an *hspl_list* parameter.

```
{
"hspl_list":{ "<?xml…<tns:hspl_list></tns:hspl_list>"
}
```

Figure 11: h2mservice input example

Once the service receives the POST request, it reads the HSPL list from the JSON file and calls the *h2mrefiner* module. The h2mrefiner module maps the policy to a python class using a correspondance generated from the HSPL scheme. In this way, it is possible to go through the HSPL policies as if they were composed by python objects. Then, the module identifies the capabilites for each HSPL policy, infering them from the action and the object of the HSPL policy. Once the capabilities have been identified, the module obtains a list of candidate security enablers from the Security Enabler Provider, according to the identified capabilities. If the candidate security enablers list is not enough to cover the aforementioned capabilities, the module returns a non-enforzable report, indicating which HSPL policy can not be enforced

with the available security enablers. On the other hand, the policy refinement process starts. Since some information can be provided at very high level, the module obtains the correspondence among the high level information and the physical one (e.g. the correspondence among the device name and its ip address). Once the required information has been obtained, and in order to perform the whole policy refinement, the *h2mrefiner* module uses specific refinement methods for each object involved on each capability. These methods are executed automatically during the parsing of the HSPL input depending on the identified capability. For instance, an HSPL filtering policy will trigger the methods related filtering actions or filtering conditions. When these kind of methods are executed, it is performed a HSPL to MSPL refinement for that specific section of the HSPL policy. Finally, when all elements have been refined, the module returns the full refinement result.

```
{
"mspl_list":{
     "mspl": "<?xml…<ITResource></ITResource>…,
     "cadidate_security_enablers": ["enabler1","enabler2"]
   }
}
```

Figure 12: h2mservice API output example

Figure 12: h2mservice API output example, shows how the MSPL list is enclosed in the JSON result, also indicating for each policy the candidate security enablers to enforce it.

5.1.2 M2LService

The m2lservice implementation is in charge to translate MSPL policies into specific security enabler configuration. Figure 13: m2lservice implementation, shows the main inputs and outputs involved on the translation process.

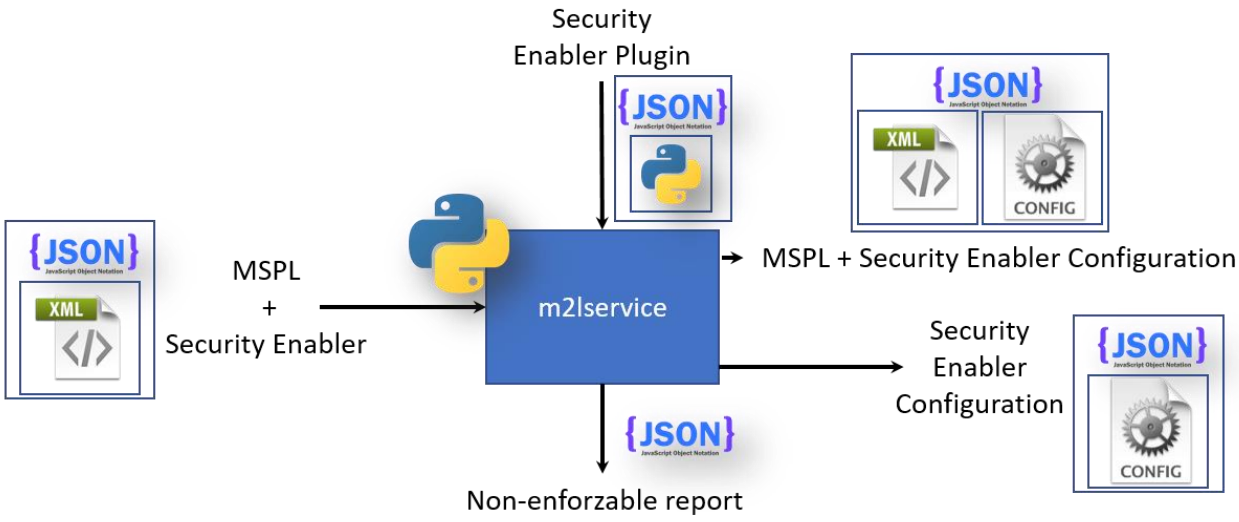

Figure 13: m2lservice implementation

In the same way as in the previous case, the service is providing a *m2lservice REST API.* In this case, it is receiving a list of MSPL policies through HTTP POST method, including a security enabler to use for each one. Figure 14: *m2lservice* API input example, shows how the MSPL list is enclosed into the JSON file. In this case, there is not a set of candidate security enablers, but the selected one.

```
{
"mspl_list":{
     "mspl": "<?xml…<ITResource></ITResource>…,
     "security_enabler": "enabler1"
   }
```

```
}
```

Figure 14: *m2lservice* API input example

Once the request has been received, the service downloads a plugin implementation from Security Enabler Provider for each MSPL policy. Each downloaded plugin is capable of translating a MSPL policy in the configuration for the specified security enabler. Thus, the service executes a specific method (i.e. *get_configuration*) on the plugin, using as parameter the MSPL policy. Figure 15: *m2lservice* API output example, shows how an ordered list is created for each security enabler which contain the specific configurations to enforce.

```
{
"enablers_configurations": {
    "enabler1": ["conf1","conf2"],
    "enabler2": ["conf1","conf2","conf3"]
    }
}
```

Figure 15: *m2lservice* API output example

Regarding the processes involved in the refinement and translations, we have identified two different steps:

- Input parser: Parse the MSPL policy which is modelled in XML.
- MSPL translation/refinement: Translates the main MSPL policy using the main fields into a specific security enabler configuration.

We have developed the aforementioned processes on python language, using a python tool called "**pyxb"** for the input parser process. This tool uses the HSPL/MSPL XML schemes to generate python code which represents the XML elements as python objects. Once the document is parsed into python objects, each relevant field or sentence is translated into, either, a specific MSPL from HSPL, or in a specific security enabler configuration from MSPL. To this aim, each plugin should implement an interface, and namely, the "***get_configuration***" method, which receives the security policy as a XML string and provide the specific refinement or translation.

```
class M2LPlugin:
        'IPTABLES Medium to low plugin implementation'
        def get_configuration(self,mspl_source):
                'Return the IPTABLES configuration from the mspl_source'
                # Customize the pyxb generated classes with our own behavior
                global mspl
                mspl.ITResourceType._SetSupersedingClass(ITResourceType)
                mspl.FilteringAction._SetSupersedingClass(FilteringAction)
                mspl.FilteringConfigurationCondition._SetSupersedingClass(FilteringConfigurationCondition)
                #Load the xml
                # Start the parser process using the xml root element
                return it_resource.get_configuration()
```

Figure 16: M2LPlugin example

The Figure 16: M2LPlugin example shows an example of the M2LPlugin module and more specifically the "***get_configuration***" method, which establishes which classes will be redefined in order to perform the translation. In addition, this method will load the MSPL policy and start the translation process from the main ITResource element, being it the root element for an MSPL policy.

5.2 IMPLEMENTATION OF THE ORCHESTRATOR

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.

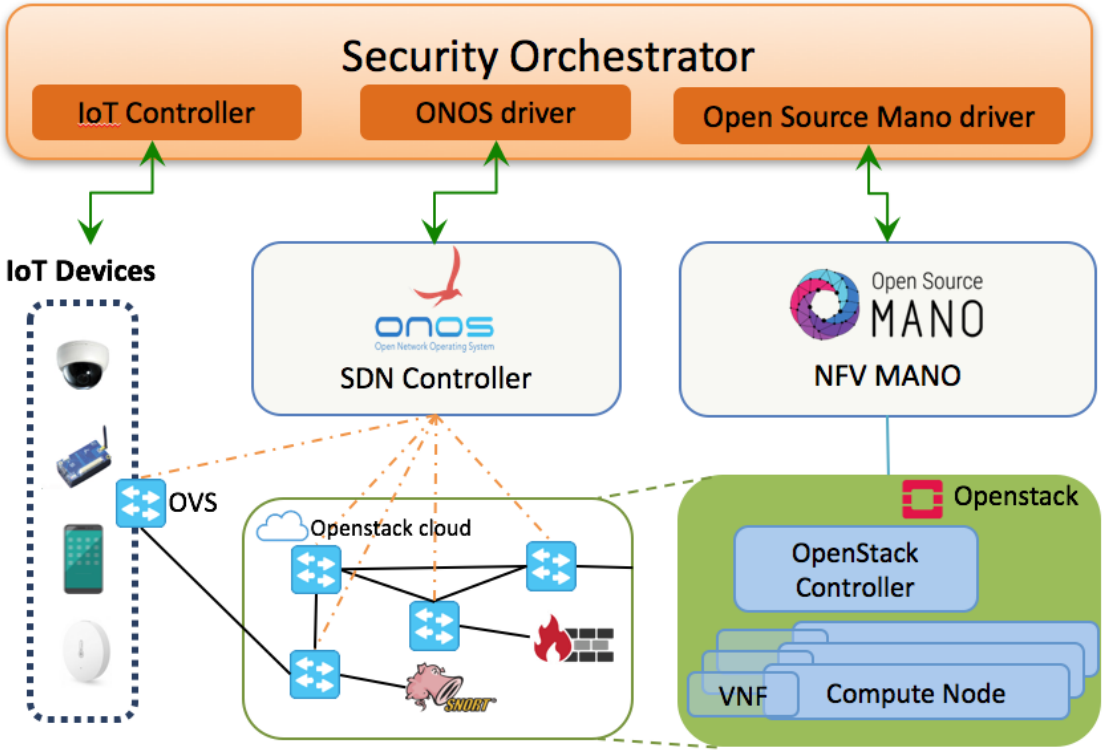

Figure 17: Security Orchestrator Implementation Architecture

To this aim, 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. This interaction is 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 programmability 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 analysed, for example: mirroring the traffic to a monitoring agent or steering the traffic through a firewall.

5.3 IMPLEMENTATION OF THE ENABLERS PROVIDER

The Security Enabler provider has two main roles; it provides the list of the available enablers and it provides a plugin that translates the policies from MSPL to Low-level configurations. The first role is implemented as a piece of software that from specific capabilities given as an input it will provide the more accurate enablers. The second role is also implemented as piece of software capable to translate MSPL policies into specific configuration/tasks rules according to a concrete security enabler.

5.3.1 The list of the available enablers

The plugin implementation that provide the list of the available enablers, is done as follows. When, the security enabler provider received the list of capabilities, it must ask the Enabler repository and parse an XML file that contains the list of the available enablers. Each enabler that matches with the capabilities is added to the list. Finally, the list is returned to the security orchestrator, where the security resource planning will use it to decide the more adequate enabler(s) among the list to be used to enforce the security.

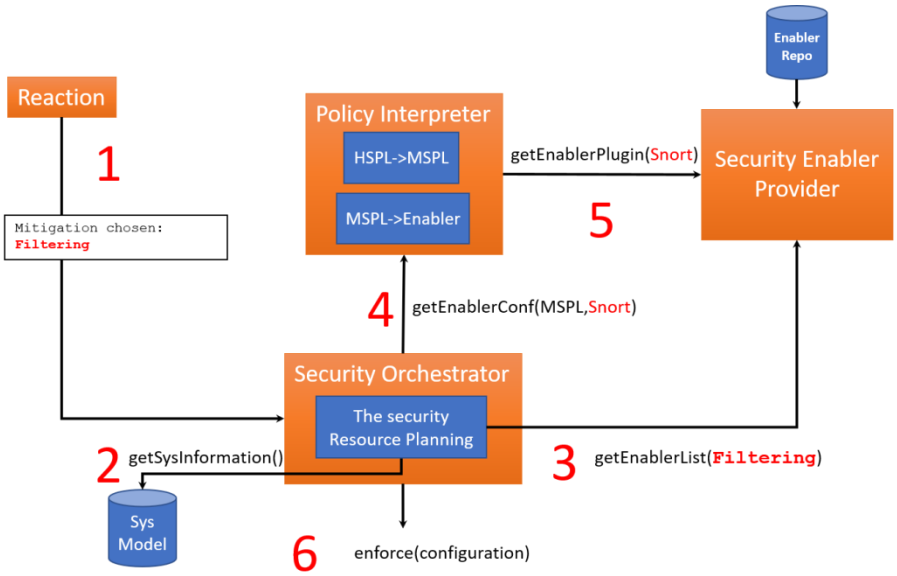

Figure 18: Security Enabler Provider steps

Figure 18 shows an example of the security enabler provider step by step. (1) Under a security attack the reaction component has chosen as a mitigation strategy: the filtering capability Figure 18: Security Enabler Provider steps. (2) The security orchestrator requests the System Model data base to obtain an up-to-date representation of the underlying system infrastructure model. (3) The aforementioned capability will be sent to the security enabler provider by calling "*getEnablerList(Filtering)*" Figure 18: Security Enabler Provider steps. The security enabler provider will ask the Enabler repository to get the list of available

enablers as “snort, iptables, firewall ….”. The previous list will be used by the security resource planning, and based on the information given by “getSysInformation()” step, it will make the proper enabler selection.

Enabler Plugin

When the Security Resource Planning subcomponent selects the enabler, the orchestrator will send the selected enabler identification, along with an MSPL file that encloses the appropriate configuration, to the Policy Interpreter. In the example shown in Figure 18: Security Enabler Provider steps in step (4), the chosen enabler is “*Snort*” Figure 18: Security Enabler Provider steps. In step (5), the request is sent from the Policy Interpreter to the Security Enabler provider, which will provide the specific plugin for the selected security enabler. Then, the Policy Interpreter executes the plugin software to perform the M2L process. Once the policy interpreter has performed the translation using the plugin, the orchestrator will receive the final configuration that will be enforced by launching/configuring the enabler, as shown in step (6) in Figure 18: Security Enabler Provider steps.

Regarding the security enabler plugins, the implementation inherits and extends the concept from the SECURED project. Nonetheless, the plugins are being implemented from scratch in python in ANASTACIA to comply with the new requirements, new kind of technologies and enablers, and to be able to cope with our new MSPL policies. The plugin consists on a piece of software which will define well-known callable methods which will take in charge to perform the translation among the MSPL policy and the specific configuration or task. The main goal of this plugin approach is to provide a plugin repository where the developers can contribute with their own plugins and solutions. Actually, it is envisaged the same policy could be enforced using different plugins. For instance, the same filtering policy can be enforced through SDN using different plugins for each SDN controllers, or by using a virtual router by configuring IPTABLES as firewall. Inside of ANASTACIA scope, it is envisaged to provide the following plugins, in order to be able to enforce the main identified security policies:

1. **SDN ONOS Plugin:** It will take in charge the MSPL policy translation onto a set of rules understandable by the ONOS controller.
2. **SDN ODL Plugin:** It will take in charge the MSPL policy translation onto a set of rules understandable by the OpenDayLight controller.
3. **IPTABLES Plugin:** It will be able to translate the MSPL policy onto iptables rules.
4. **OpenWRT Plugin:** It will be in charge to generate OpenWRT router configuration from an MSPL policy.
5. **Quagga Plugin:** It will be in charge to generate Quagga router configuration from an MSPL policy.
6. **Open Virtual Switch Plugin:** It will generate OVS rules according to the specified policy.
7. **PfSense Plugin:** It will take in charge to generate the PfSense firewall configuration.
8. **SNORT Plugin:** It will translate the MSPL policy onto a SNORT monitoring rules.
9. **DTLS/TLS Proxy Plugin:** It will obtain the proxy configuration for DTLS/TLS secure connections.
10. **VPN Plugin:** It will generate the OpenVPN configuration from the secure policies parameters.
11. **Kippo Plugin:** It will perform the translation from MSPL to Kippo honeypot.
12. **Cooja Plugin:** It will translate the physical IoT system model provided to a Cooja virtual one.
13. **vAAA XACML Plugin:** It will generate the required configuration for the AAA infrastructure from the MSPL policy.
14. **Data Aggregator Proxy Plugin:** It will translate the security policy to a configuration of a data aggregator proxy in order to provide pseudonimity.
15. **IoT Control Plugin:** This plugin will be in charge of translating the operational IoT security policies to the specific IoT Controller rules.

6 POLICY-BASED DEPLOYMENT EXAMPLE: CASCADE ATTACK ON A MEGATALL BUILDING

6.1 NARRATIVE DESCRIPTION

FoulGame is a notorious group of criminal hackers who specialize in attacks on internet-connected services of global brands. They have set their eyes to destroy the brand name of Hilltop Group who owns many iconic hotels worldwide. FoulGame intends to use internet-connectivity of the buildings operations to create an emergency in a mega-tall hotel building. They hope that the emergency will generate panic, trap the guests in escape elevators, activate fire-suppression sprinklers, confuse first-responders, etc.
FoulGame wants to exploit a zero day vulnerability of the HVAC system network that allows an external service such as an internet-service or original equipment manufacturer (OEM) to set default values (e.g., -40 ºC) to temperature sensors. For practical reasons, HVAC zonal temperatures are also monitored by the fire safety systems as a precaution. But if the temperature exceeds a threshold (e.g., +80 ºC), an emergency is activated. This could cascade to alarms and sprinklers activating, air-handlers stopping, elevators becoming disabled, fire-doors and corridors closing, etc. Risk to lives of occupants due to activation of fire-suppression systems, depletion of oxygen in the air, and rush and stampede in the stairwells will be catastrophic.
Hilltop Group can use ANASTACIA to identify and rate cyber-security security vulnerabilities automatically for the entire building. ANASTACIA will use system design and operational data to discover dependencies between cyber-physical systems and operations for the entire megatall structure. Hilltop Group will use ANASTACIA to predict potential security consequences of interacting operations between subsystems and generate threat isolation strategies. ANASTACIA will continuously enforce access and security policies and resilient control strategies comprehensively at various cyber-physical levels, viz.,the temperature sensors, fire-panels, elevator system managers, air-handling unit controllers, fire-suppression sprinkler systems, etc.

6.2 INVOLVED ACTORS

The actors are:

- The building **manager**, responsible for the building security and safety.
- The **ANASTACIA platform** installed and used in order to ensure the safety of the building.
- A criminal hacker, attacking the building by modifying the temperature sensor data.

6.3 USE CASE STEPS

The use case has the following steps.

- The criminal hacker exploits zero-day vulnerability for internet-connected temperature sensor.
- The hacker uses this sensor vulnerability to connect to the sensor whenever it is online and manipulates the temperature value by increasing it up to 80C. This data tempering will trigger the fire alarm, the evacuation alarm, deactivation of elevators and HVAC heat exchangers.

- The ANASTACIA platform monitors the physical and cyber behaviour correlate, such as temperature value, with other sensing and actuation values in the same zone and adjacent zones.
- The ANASTACIA platform detects outlies, which can be due to an intrusion or malicious activities.
- The ANASTACIA platform analyses the detected abnormalities and outliers and evaluates the severity of the situation.
- The ANASTACIA platform activates predictive mechanisms to ensure that the rest of the building operations system continues as normal:
    - The ANASTACIA platform sends an alert to the building manager
    - The ANASTACIA platform activates safe-mode of operations.
- The ANASTACIA platform enables the manager evaluates all components in the entire hierarchy of the building operations.
- The administrator reacts to the alert:
    - The manager identifies the attack as a real intrusion.
    - The manager accepts the changes suggested by the ANASTACIA platform.

6.4 USE CASE FORMALIZATION

Table 14: Use case formalization, shows a more formalized approach of the use case, specifying fields like the use case identification, actors, description or the pre-conditions.

USE CASE BMS.4		
A	**Use Case ID**	*UC_BMS.4*
B	**Use Case Name**	*Cascade attack on a megatall building*
	Primary actors	ANASTACIA platform
	Supporting actors	building manager, criminal hacker
	Description	The ANASTACIA platform, installed to protect a megatall building, reacts to a criminal hacker remote data tempering
	Stakeholders' interests	Protect the system from a remote data tempering for sensitive sensor and actuation data.
	Triggers	A criminal hacker plans to gain control over critical temperature sensor to manipulate the temperature value and hence triggering the fire and evacuation alarms.
	Pre-conditions	A building automation system was installed within a megatall building and the ANASTACIA platform was deployed and configured to protect all the sensitive data points.
	Normal flow	[*UC_BMS.4*]

		Course of Actions 1. The ANASTACIA platform monitors cyber and physical signals and correlate their behaviours in building operational subsystems. 2. The ANASTACIA platform detects an intrusion. 3. The ANASTACIA platform analyses the detected abnormalities and outliers and evaluates the severity of the situation. 4. The ANASTACIA platform activates predictive mechanisms to ensure that the rest of the plant operations system continues as normal: a. The ANASTACIA platform sends an alert to the building manager b. The ANASTACIA platform activates resilient and safe-mode of operations. 5. The ANASTACIA platform enables the manager evaluates all components in the entire hierarchy of the building operations. 6. The administrator reacts to the alert: a. The manager identifies the attack as a real intrusion. b. The manager accepts the changes suggested by the ANASTACIA platform.
	Post-conditions	The state of the system has been re-established, the criminal hacker attack has been avoided, the device has been isolated, and the device vulnerability has been reported.

Table 14: Use case formalization

6.5 SECURITY POLICIES INSTANTIATION

To accomplish the current use case example, we asume the reaction module or the administrator aims to apply the following security policies in order to recover the affected sensor:

- IoT Control – Power Management.
  - Reset the IoT device.
- Filtering – Attacker IP address.

If the sensor continues showing an abnormal behaviour a more drastical solition will be taken:

- IoT Control – Power Management.
  - Turn off the IoT device
- Filtering – All traffic from/to affected IoT device.

Below section illustrates definition, refinement and translation examples for the identified security policies.

6.5.1.1 Policy definition

If the countermeasure is provided by the security administrator, it could be provided at High-level Security Policy Language (HSPL), in order to ease the policy definition, or even directly in MSPL.

```
<hspl_list>
   <hspl id="Attacker-filtering" subject="Attacker">
      <action>no_authorise_access</action>
             <objectH>AllTraffic</objectH>
       </hspl>
       <hspl id="SensorA-2" subject="SensorA">
             <action>enable</action>
             <objectH>resource</objectH>
             <fields>
                    <tns:resource_values>
```

```
                        <tns:name_resources>Reset</tns:name_resources>
                    </tns:resource_values>
                </fields>
        </hspl>
</hspl_list>
```

Figure 19: Use case HSPL example

Figure 19: Use case HSPL example shows an example of HSPL policies list definition which is compounded by two different policies. The first one indicates that the subject, the identified ***attacker***, is ***not authorised to access any*** kind of ***traffic***. The second one indicates that it is necessary to ***enable*** the resource ***Reset*** over the ***SensorA***. In case the sensor continues showing an abnormal behaviour it can apply more severe measures like turn off the IoT device and filter all traffic from/to the IoT device, preventing the IoT control to execute commands that are not working properly. To this aim, the HSPL policies will be quite similar, being required to change just the subject and the specific resource.

6.5.1.2 Policy refinement

The policy refinement process takes as input HSPL policies and it refines them into a Medium-level Security Policiy (MSPL). This process is not mandatory since an experimented user could define the security policy directly using the MSPL model, instead of the HSPL one.

```
<ITResource>
        <configuration xsi:type="RuleSetConfiguration">
                <capability  xsi:type="FilteringCapability">
                        <Name>Filtering_L3</Name>
                </capability>
                <configurationRule>
                        <configurationRuleActionxsi:type="FilteringAction">
                                <FilteringActionType>DENY</FilteringActionType>
                        </configurationRuleAction>
                        <configurationConditionxsi:type="FilteringConfigurationCondition">
                                <isCNF>false</isCNF>
                                <packetFilterCondition>
                                        <SourceAddress>aaaa::2</SourceAddress>
                                </packetFilterCondition>
                        </configurationCondition>
                        <Name>Attacker-filtering-1</Name>
                        <isCNF>false</isCNF>
                </configurationRule>
                <Name>Attacker-filtering</Name>
        </configuration>
</ITResource>
```

Figure 20: Use case MSPL filtering example

Figure 20: Use case MSPL filtering example, shows an example of filtering HSPL policy refinement. As can be observed, in this case the Policy Interpreter has identified as main capability ***Filtering_L3*** according to the combination of the action and the object in the received HSPL policy, providing also as specific rule action a ***filtering action***. This specific filtering action has inferred from the action of the HSPL policy, generating then a filtering action indicating that it must be applied the action ***DENY*** over the filtering condition. Finally, the filtering conditions are generated considering subject, target and object HSPL fields. At this point, it can be observed how the attacker´s name has been translated to attacker´s ip address. This translation occurs thanks to an interaction with the Security Orchestrator in order to get system model information.

```
<ITResource>
        <configuration xsi:type="RuleSetConfiguration">
                <capability>
                        <Name>IoT_control</Name>
                </capability>
                <configurationRule>
                        <configurationRuleAction xsi:type='PowerMgmtAction'>
```

```
                        <PowerMgmtActionType>RESET</PowerMgmtActionType>
                  </configurationRuleAction>
                  <configurationCondition xsi:type='FilteringConfigurationCondition'>
                        <isCNF>false</isCNF>
                        <packetFilterCondition>
                              <SourceAddress>aaaa::2</SourceAddress>
                        </packetFilterCondition>
                        <appLayerCondition xsi:type='IoTApplicationLayerCondition'>
                              <URL>.reset</URL>
                              <method>PUT</method>
                        </applicationLayerCondition>
                  </configurationCondition>
            </configurationRule>
            <Name>SensorDisableConfiguration</Name>
      </configuration>
</ITResource>
```

Figure 21: Use case MSPL IoT control example

Figure 21: Use case MSPL IoT control example, shows an example of IoT control HSPL policy refinement. As can be observed, in this case the Policy Interpreter has identified as main capability ***IoT_control*** according to the combination of the action and the object in the received HSPL policy, providing also as specific rule action a ***power management action***. This specific IoT action has been inferred from the action of the HSPL policy, generating then a power management action indicating ***OFF*** as power management type. Finally, a filtering configuration condition is generated, considering subject, target and object HSPL fields. Specifically, it has been generated an IoT application condition in order to specify the URL and method available to perform the specified action. At this point, it can be observed how the IoT device´s name has been translated to IoT device´s ip address. This translation occurs thanks to an interaction with the Security Orchestrator in order to get system model information, in the same way of the previous one.

6.5.1.3 Policy translation

The policy translation process takes as input MSPL policies, including the security enabler will oversee the policy enforcement, and then it refines them into the specified security enabler configuration. Below are showed the different configurations generated depending on the chosen security enabler.

```
FORWARD -s aaaa::2 -j DROP
```

Figure 22: Use case iptables filtering example

Figure 22: Use case iptables filtering example, shows an example of the generated configuration for the previous MSPL filtering policy, choosing as security enabler iptables. In this case, it is being dropped all packets which contain as source ip address the attacker´s ip address.

```
{
    "priority": 60000,
    "timeout": 0,
    "isPermanent": true,
    "deviceId": "[DEVICE_ID (e.g. of:000008002768a30a)]",
    "treatment": {
        "instructions": [
            {
                "type": "NOACTION"
            }
        ]
    },
    "selector": {
        "criteria": [
            {
                "type": "ETH_TYPE",
                "ethType": "0x86dd"
            },
            {
                "type": "IPV6_SRC",
```

```
                "ip": "aaaa::2"
            }
        ]
    }
}
```

Figure 23: Use case ONOS Northbound filtering example

Figure 23: Use case ONOS Northbound filtering example, shows an example of the generated configuration for the same MSPL filtering policy, but in this case, choosing as security enabler the Northbound API of the ONOS SDN controller. The configuration is also indicating the priority of the rule, even the switch SDN-enabled where the rule must be enforced.

```
{
	"id": "Rule0",
	"priority": 60000,
	"table_id": 0,
	"flow-name": "Rule0",
	"instructions": {
		"instruction": [{
			"order": 0,
			"apply-actions": {
				"action": [{
					"order": 0,
					"drop-action": {}
				}]
			}
		}]
	},
	"match": {
		"ethernet-match": {
			"ethernet-type": {
				"type": "34525"
			}
		},
		"ipv6-source": "aaaa::2"
	}
}
```

Figure 24: Use case ODL Northbound filtering example

Figure 24: Use case ODL Northbound filtering example, shows an example of the generated configuration for the same MSPL filtering policy, but in this case, choosing as security enabler the Northbound API of the Opendaylight SDN controller. As can be observed, the configuration is quite similar for both SDN controllers, differing basically on the field names.

```
{
	"target": "aaaa::2",
	"method": "PUT",
	"resource":".reset"
	"payload": 1
}
```

Figure 25: Use case IoT Controller Northbound example

Finally, Figure 25: Use case IoT Controller Northbound example, shows an example of the generated configuration for the IoT control MSPL policy. In this case the MSPL is being translated to a JSON file configuration which will be processed by the IoT Controller.

6.5.1.4 Policy Deployment Process

This section shows the policy enforcement process for the instantiation of the security policies in the example use case. Since the policy enforcement can be triggered from the Reaction Module or from the Policy Editor Tool, below are showed and explained these two different approaches.

6.5.2 Proactive Policy Deployment

Figure 26: Proactive filtering deployment, shows an example of filtering policy deployment in a proactive way, which will follow below steps:

1. The system administrator models an HSPL policy indicating that the identified attacker is not authorised to access any kind of traffic. Once the filtering policy has been modelled, the system administrator can request the HSPL policy enforcement to the framework.
2. The Policy Interpreter identifies the HSPL capabilities, being in this case FILTERING_L3, and it requests a list of candidate security enablers which implements FILTERING_L3 capability, to the Security Enabler Provider.

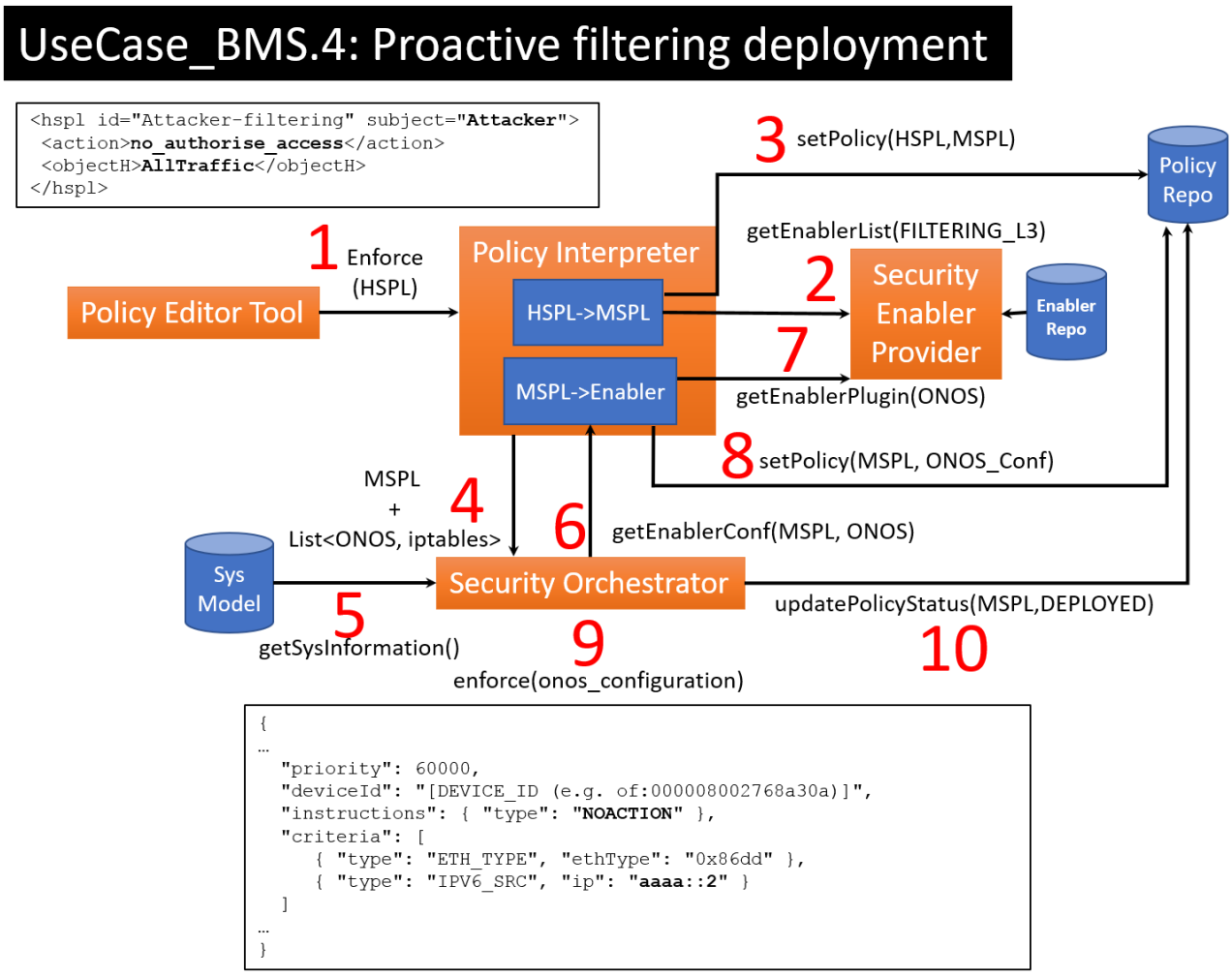

Figure 26: Proactive filtering deployment

3. The Policy Interpreter performs the HSPL to MSPL policy refinement and stores the result in the Policy repository.
4. The Policy Interpreter sends the generated MSPL and the list of candidate security enablers, which in this case are *ONOS* and iptables.
5. The Security Orchestrator takes information of the underlaying architecture and it decides which is the best enabler to use. In this case, it decides to use *ONOS*.

6. The Security Orchestrator requests a MSPL policy translation to specific *ONOS* configuration.
7. The Policy Interpreter requests the translator plugin for *ONOS*. Once has been obtained, executes the *get_configuration* method in the plugin, using as parameter the MSPL policy. The plugin then performs the translation, generating an *ONOS* configuration.
8. The Policy Interpreter stores the correspondence among the MSPL policy and the *ONOS* configuration in the Policy repository.
9. The Security Orchestrator receives the *ONOS* configuration and it starts the policy enforcement through the *ONOS* SDN controller.
10. The Security Orchestrator updates the security policy status, storing the result of the policy enforcement in the Policy Repository. If the enforcement has been done successfully, it will set the security policy as *DEPLOYED*.

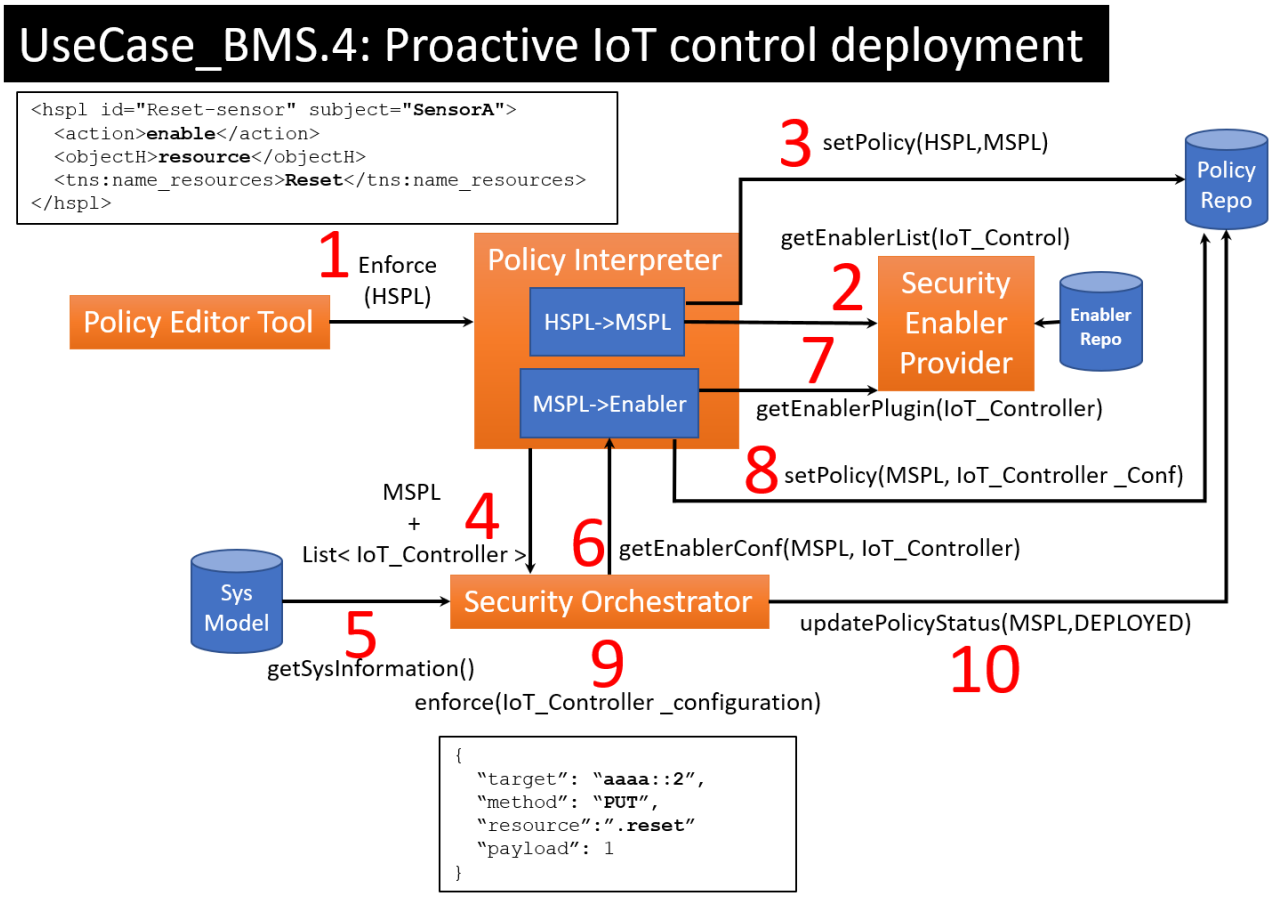

Figure 27: Proactive IoT control deployment

Figure 27: Proactive IoT control deployment, shows an example of IoT control policy deployment in a proactive way, which will follow below steps:

1. The system administrator models an HSPL policy indicating that the *SensorA* must be restarted. Once the filtering policy has been modelled, the system administrator can request the HSPL policy enforcement to the framework.
2. The Policy Interpreter identifies the HSPL capabilities, being in this case *IoT_Control*, and it requests a list of candidate security enablers which implements *IoT_Control* capability, to the Security Enabler Provider.
3. The Policy Interpreter performs the HSPL to MSPL policy refinement and stores the result in the Policy repository.
4. The Policy Interpreter sends the generated MSPL and the list of candidate security enablers. In this case, there is only one plugin, the *IoT_Controller* plugin.

5. The Security Orchestrator takes information of the underlaying architecture. Since there is only one candidate security enabler, it is selected.
6. The Security Orchestrator requests a MSPL policy translation to specific IoT Controller configuration.
7. The Policy Interpreter requests the translator plugin for IoT Controller. Once has been obtained, executes the *get_configuration* method in the plugin, using as parameter the MSPL policy. The plugin then performs the translation, generating an IoT Controller configuration.
8. The Policy Interpreter stores the correspondence among the MSPL policy and the IoT Controller configuration in the Policy repository.
9. The Security Orchestrator receives the IoT Controller configuration and it starts the policy enforcement through the IoT Controller.
10. The Security Orchestrator updates the security policy status, storing the result of the policy enforcement in the Policy Repository. If the enforcement has been done successfully, it will set the security policy as *DEPLOYED*.

6.5.3 Reactive Policy Deployment

Figure 28: Reactive filtering deployment, shows an example of filtering policy deployment in a reactive way, which will follow below steps:

1. The Reaction module gets a MSPL policy template according with the required reaction, specifying as capability *FILTERING_L3* and the action *FilteringAction*.
2. The Reaction module fills properly the MSPL template, adding the *FilteringActionType* and the *SourceAddress* fields. Then it is sent to the Security Orchestrator to start the policy enforcement.

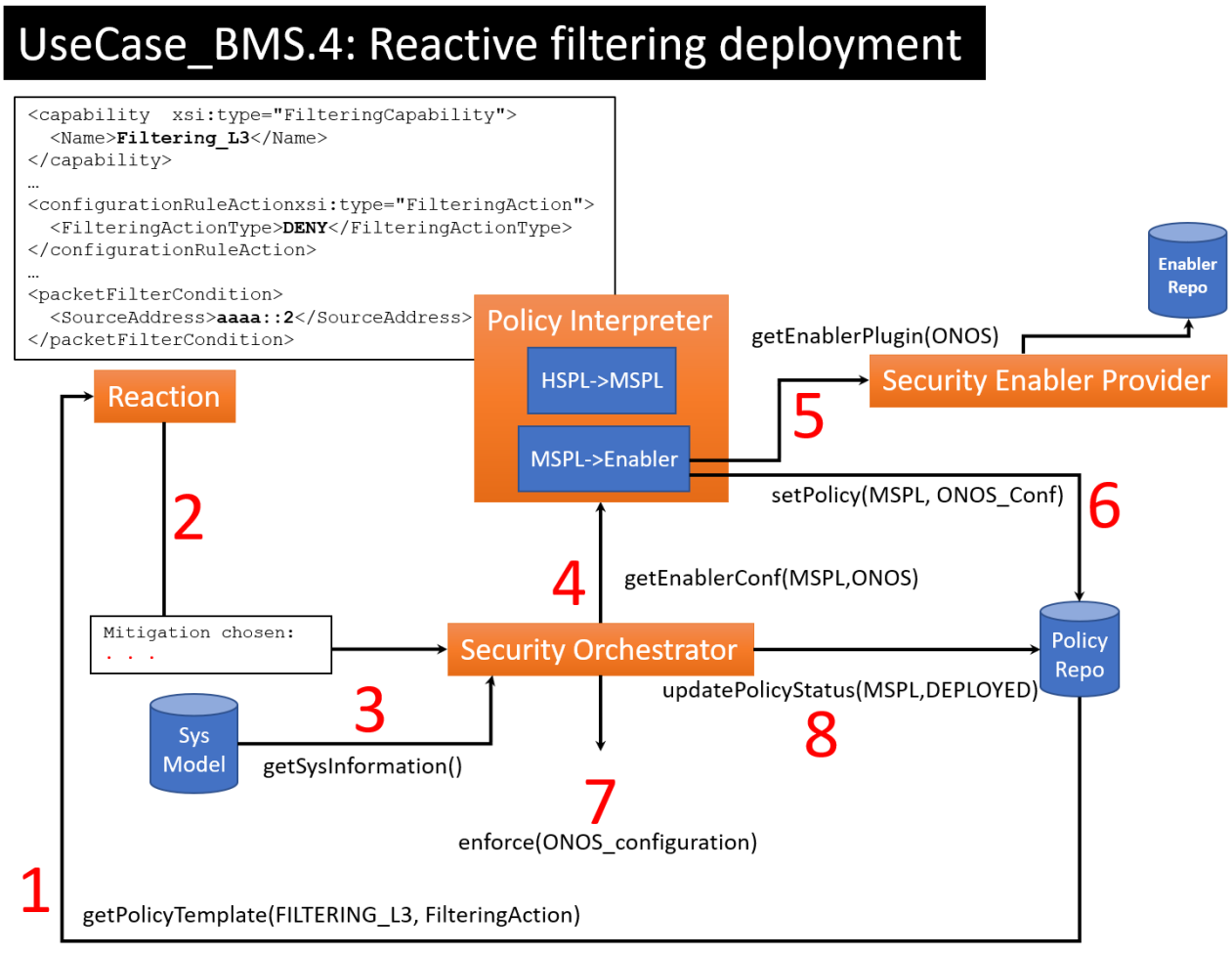

Figure 28: Reactive filtering deployment

3. The Security Orchestrator takes information of the underlaying architecture and it decides which is the best enabler to use. In this case, it decides to use ONOS.
4. The Security Orchestrator requests a MSPL policy translation to specific ONOS configuration.
5. The Policy Interpreter requests the translator plugin for ONOS. Once has been obtained, executes the *get_configuration* method in the plugin, using as parameter the MSPL policy. The plugin then performs the translation, generating an ONOS configuration.
6. The Policy Interpreter stores the correspondence among the MSPL policy and the ONOS configuration in the Policy repository.
7. The Security Orchestrator receives the ONOS configuration and it starts the policy enforcement through the ONOS SDN controller.
8. The Security Orchestrator updates the security policy status, storing the result of the policy enforcement in the Policy Repository. If the enforcement has been done successfully, it will set the security policy as *DEPLOYED*.

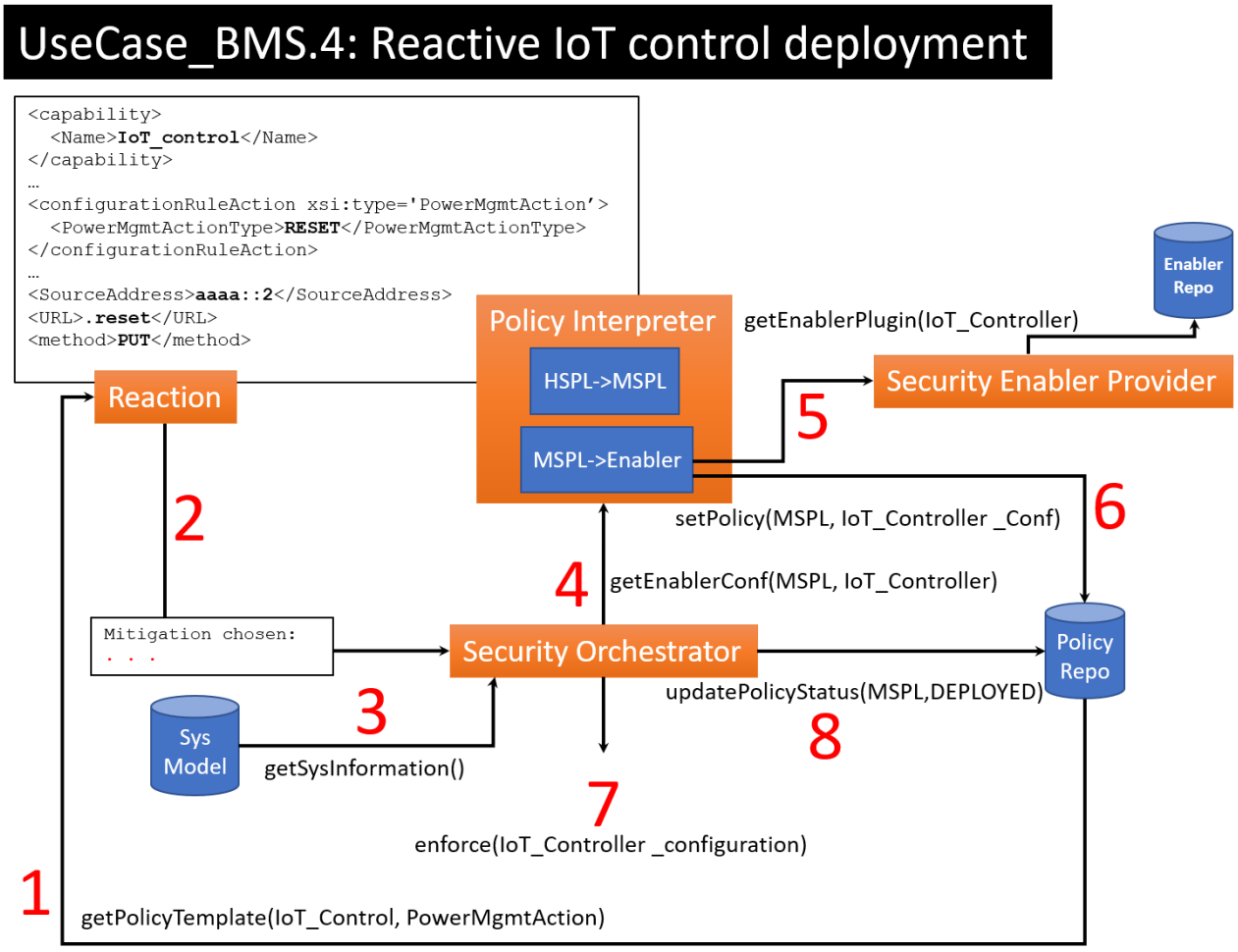

Figure 29: Reactive IoT control deployment

Figure 29: Reactive IoT control deployment, shows an example of filtering policy deployment in a reactive way, which will follow below steps:

1. The Reaction module gets a MSPL policy template according with the required reaction, specifying as capability *IoT_Control* and the action *PowerMgmtAction*.
2. The Reaction module fills properly the MSPL template, adding the *PowerMgmtAction*, the *SourceAddress*, the *URL* and the *method* will be used. Then it is sent to the Security Orchestrator in order to start the policy enforcement.

3. The Security Orchestrator takes information of the underlaying architecture and it decides which is the best enabler to use. In this case, it decides to use IoT Controller.
4. The Security Orchestrator requests a MSPL policy translation to specific IoT Controller configuration.
5. The Policy Interpreter requests the translator plugin for IoT Controller. Once has been obtained, executes the *get_configuration* method in the plugin, using as parameter the MSPL policy. The plugin then performs the translation, generating an IoT Controller configuration.
6. The Policy Interpreter stores the correspondence among the MSPL policy and the IoT Controller configuration in the Policy repository.
7. The Security Orchestrator receives the IoT Controller configuration and it starts the policy enforcement through the IoT Controller.
8. The Security Orchestrator updates the security policy status, storing the result of the policy enforcement in the Policy Repository. If the enforcement has been done successfully, it will set the security policy as *DEPLOYED*.

7 CONCLUSIONS

This document provides the first report about the policy refinement and translation processes being devised and implemented in WP3, and concretely in Task 3.1. In this regard, the document has described the Policy Interpreter component of the Anastacia framework, as main component in charge of performing those taks. The report delves into the state of art regarding the policy refinement techniques and technologies. The document defines the relationships and interfaces between the Policy interpreter and the rest of components. Besides, it has been detailed the policy refinement process from high-level security policy language to the medium-level security policy language, as well as the translation process from the medium-level security policy language to the specific security enabler configurations.

Once it has been ilustrated at design level, it has been provided an explanation regarding the current first implementation and integration of the main components and services involved on the policy enforcement, that is, the Policy Interpreter, the Security Orchestrator and the Security Enabler Provider.

Finally, it has been instantiated a specific use case, providing security policy examples at different abstraction levels, and showing how the use case could be covered through the security policies deployment in two distinct ways; a proactive one, where the system administrator is in charge to model and requests the policy enforcement, and a reactive one, whereby the Reaction module is in charge if starting the policy enforcemnt process.

8 REFERENCES

[1] Common Information Model (CIM), DMTF Standard.

[2] Jorge Bernal Bernabe, Juan M. Marin Perez, Jose M. Alcaraz Calero, Jesus D. Jimenez Re, Felix J. Garcia Clemente, Gregorio Martinez Perez, Antonio F. Gomez Skarmeta, **"Security Policy Specification"**, Network and Traffic Engineering in Emerging Distributed Computing Applications, IGI Global, pp. 66-93, 2012.

[3] Policy-Based Security Tools and Framework (POSITIF), EU project, FP6, IST-2002-002314

[4] Dependable Security by Enhanced Reconfigurability (DESEREC), IST-2004-026600, EU project, Framework Programme 6

[5] SECURED EU FP7 project, deliverable D4.1: **Policy specification**.

[6] SECURED EU FP7 project, deliverable D4.2: **Policy transformation and optimization techniques**.

[7] Diego Lopez et al. **I2NSF Framework for Interface to Network Security Functions**. RFC 8329, IETF. Feb 2017. I2NSF Working Group

[8] L. Xia et al. **Information Model of NSFs Capabilities**. Internet-Draft, IETF. December 2017.

[9] I. Farris et al., "Towards provisioning of SDN/NFV-based security enablers for integrated protection of IoT systems," 2017 IEEE Conference on Standards for Communications and Networking (CSCN), Helsinki, 2017, pp. 169-174.doi: 10.1109/CSCN.2017.8088617

[10] S. Ziegler, A. Skarmeta, J. Bernal, E. E. Kim and S. Bianchi, "ANASTACIA: Advanced networked agents for security and trust assessment in CPS IoT architectures," 2017 Global Internet of Things Summit (GIoTS), Geneva, 2017, pp. 1-6.doi: 10.1109/GIOTS.2017.8016285