A coordinated framework for cyber resilient supply chain systems over complex ICT infrastructures

D4.1 Security and Certification Manager components design and implementation (IT-1)

Document Identification			
Status	Final	**Due Date**	31/05/2021
Version	1.0	**Submission Date**	18/06/2021

Related WP	WP4	**Document Reference**	D4.1
Related Deliverable(s)	NA	**Dissemination Level (*)**	PU
Lead Participant	POLITO	**Lead Author**	Cataldo Basile
Contributors	STS, ATOS, XLAB	**Reviewers**	Henrique Santos, Alexandre Santos, André Oliveira (UMinho)
			Eva Marín Tordera, Xavi Masip Bruin (UPC)

Keywords:

Enforcement and Dynamic Configuration, Security Assurance and Certification Manger, security policy models

This document is issued within the frame and for the purpose of the FHISY project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 952644. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

(*) Dissemination level: **PU**: Public, fully open, e.g. web; **CO:** Confidential, restricted under conditions set out in Model Grant Agreement; **CI:** Classified, **Int =** Internal Working Document, information as referred to in Commission Decision 2001/844/EC.

Document Information

List of Contributors	
Name	Partner
Cataldo Basile	POLITO
Daniele Canavese	POLITO
Antonio Lioy	POLITO
Grigorios Kalogiannis	STS
José Javier de Vicente Mohino	ATOS
José Francisco Ruiz	ATOS
Aleš Černivec	XLAB

Document History			
Version	Date	Change editors	Changes
0.1	26/03/2021	Cataldo Basile (POLITO)	First draft, structure defined.
0.2	03/05/2021	Cataldo Basile (POLITO)	First complete draft of the T4.1-related part.
0.2.1	07/05/2021	José F. Ruiz and José Javier de Vicente (ATOS)	Review of the v0.2 of the documents and comments.
0.3	07/05/2021	Grigorios Kalogiannis (STS)	T4.2 sections included.
0.4	10/05/2021	Cataldo Basile (POLITO)	Integrations and homogenization.
0.4.1	1/05/2021	José F. Ruiz and José Javier de Vicente (ATOS)	Review of the v0.4 of the document and minor comments.
0.5	11/05/2021	Cataldo Basile (POLITO)	Document finalized. Ready for the internal review.
0.5.1	18/05/2021	Aleš Černivec (XLAB)	Contributions added to the SACM.
0.6	23/03/2021	Cataldo Basile (POLITO)	Reviewers' comments in sections 1-3 and 5 addressed.
0.7	23/02/2021	Grigorios Kalogiannis (STS)	Reviewers' comments in sections 4 addressed.
0.8	26/05/2021	Cataldo Basile (POLITO)	Final version ready for QA
0.9	27/05/2021	Juan Alonso	Quality assessment
1.0	18/06/2021	Juan Alonso, Jose Ruiz (ATOS)	FINAL VERSION

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Cataldo Basile (POLITO)	26/05/2021
Quality manager	Alonso, Juan Andres (ATOS)	18/06/2021
Project Coordinator	Ruiz, Jose Francisco (ATOS)	18/06/2021

Table of Contents

Document Information 2
Table of Contents 4
List of Figures 6
List of Acronyms 7
Executive Summary 8
1 Introduction 10
1.1 Purpose of the document 10
1.2 Relation to other project work 11
1.3 Structure of the document 12
1.4 Glossary adopted in this document 12
2 Design of WP4 components 14
3 The Enforcement and Dynamic Configuration 17
3.1 Architecture and components 17
3.2 Workflow 18
3.3 Data models 20
3.3.1 High-level policy model 20
3.3.2 Low-level configuration settings (aka configurations) 21
3.3.3 The Security Capability model and Medium-level policies (aka abstract configurations .21
3.3.4 Landscape scenario 24
3.4 EDC Component: Controller 24
3.4.1 Architecture, Functionality, interactions 24
3.4.2 Black-box modelling and API 25
3.5 EDC Component: Register and Planner 26
3.5.1 Architecture, Functionality, interactions 26
3.5.2 Black-box modelling and API 27
3.6 EDC Component: Enforcer 27
3.6.1 Architecture, Functionality, interactions 27
3.6.2 Black-box modelling and API 28
4 The Security Assurance and Certification Management 30
4.1 Architecture and components 30
4.2 Workflow 31
4.3 Data models 32
4.3.1 Evidence Collection Engine data model 32
4.3.2 Audit data model 32
4.4 SACM Component: Security Metrics 33
4.4.1 Architecture, Functionality, interactions 33
4.4.2 Black-box modelling and API 34
4.5 SACM Component: Audit 34
4.5.1 Architecture, Functionality, interactions 34
4.5.2 Black-box modelling and API 35

4.6 SACM Component: Certification 36
4.6.1 Architecture, Functionality, interactions 36
4.6.2 Black box modelling and API 39
4.7 SACM Component: Evidence Collection Engine 39
4.7.1 Architecture, Functionality, interactions 39
4.7.2 Black-box modelling and API 40
5 Conclusions 42
6 References 43

List of Figures

Figure 1. The role of the Knowledge Base in the WP4 architecture. 14
Figure 2. EDC: high-level view of the components. 17
Figure 3. EDC Workflow (Part I) 18
Figure 4. The EDC workflow (Part II). 20
Figure 5. The current Capability model specification. 23
Figure 6. The Black-Box view of the Controller 25
Figure 7. The Black-Box view of the Register and Planner. 27
Figure 8. The Black-Box view of the Enforcer. 28
Figure 9. SACM: high-level view of the components...31
Figure 10. The SACM workflow. 31
Figure 11. The Black Box view of the Security Metrics component. 34
Figure 12. The Black-Box view of the Audit component. 35
Figure 13: Asset-based and metric-based evaluation of the compliance. 37
Figure 14. The Black-Box view of the Certification. 39
Figure 15. The Black-Box view of the Evidence Collection Engine 40

List of Acronyms

Abbreviation / acronym	Description
CIA	Confidentiality – Integrity – Availability
CRUD	Create, Read, Update, and Delete
D4.1	Deliverable number 1 belonging to WP4
EC	European Commission
EDC	Enforcement and Dynamic Configuration
EVEREST	EVEnt REaSoning Toolkit
HLP	High-Level Security Policies
I2NSF	Interface to Network Security Functions
IRO	Intent-based Resilience Orchestrator & Dashboard
MLP	Medium-Level security Policies
NFV	Network Function Virtualization
NSF	Network Security Function
REST	Representational State Transfer
SACM	Security Assurance and Certification Management
SIA	Security Infrastructure Abstraction
SPI	Security & Privacy Data Space Infrastructure
TIM	Trust and Incident Manager
WP	Work Package

Executive Summary

The WP4 provides two main functions to the overall FISHY objectives. It develops the Enforcement and Dynamic Configuration (EDC) component that keeps the security controls correctly configured using a set of high-level policies (the desired security requirements). Furthermore, WP4 provides the Security Assurance and Certification Management (SACM) component, which includes mechanisms for auditing and assessing ICT systems and an evidence-based certifiable view, which ensures and certifies that the desired security policies are correctly implemented.

Important decisions have been already made in the first months of the WP4 activity. The analysis of the WP components has highlighted that **the EDC and the SACM are two independent components**. All the design and coordination activities have been defined accordingly. Furthermore, after careful analysis, **the FISHY Knowledge base has been adopted as the logical aggregation of all the instances of the data models** used by WP4 components to exchange information among components (and very likely, it will be adopted by all the other technical work packages). Finally, fully automatic systems are very welcomed from the research point of view but not very easy to accept in real-world scenarios. Therefore, **all the EDC operations must be explicitly authorized by the administrators**. The issues related to the automatic execution of the SACM operations have not been fully explored, however, they appeared, after a preliminary analysis, to introduce less risks.

Three abstraction levels have been identified for the security policies. The *High-Level security policies* (HLP) are used to specify security requirements without explicitly referring to the security controls. The *Medium-Level security Policies* (MLP) specify concrete configuration settings using a Network Security Function (NSF)-independent language. The *Low-Level configurations* settings specify the NSF-specific configuration settings. The *refinement* is the process that transforms HLP into abstract configurations in MLP. The *translation* is the process that transforms abstract configurations in MLP into Low-Level configurations.

The EDC is composed of three components: the **Controller** performs the refinement, the **Enforcer** performs the translation and deploys, through the WP5 Security Infrastructure Abstraction component, the obtained configurations in the target NSFs. Finally, the **Register and Planner** component manages a catalogue of all the NSF available in the domain where the FISHY infrastructure is adopted. All these components rely on the definition of a *formal model of the security capabilities*, which are an abstraction of what security controls (i.e., NSF) can do in terms of security policy enforcement. The security capability model is based on the results in the IETF I2NSF working group and, in its current form, it addresses functions related to the enforcement of Network Security Policy.

The EDC workflow starts with the Intent-based Resilience Orchestrator & Dashboard (IRO, defined in WP5) that triggers the refinement performed by the Controller, which uses the Register and Planner to get information about the available NSFs. Then the IRO triggers the Enforcer to ask the generation of the configurations and their deployment. The Enforcer also uses the Register and Planner to get information about the available NSFs. The Knowledge base notifies the IRO about the changes in any policy data models or in the landscape to solicit administrators' reactions, thus relieving individual components from passing data and triggers among each other.

The SACM is composed of four independent modules. The real-time, continuous assessment of the security posture of a target ICT system is performed by the **Evidence Collection Engine**, which aggregates from multiple sources the evidence related to the operation of components in isolation and when those should interact with other ones. The **Audit** module is responsible for initiating, coordinating, and reporting the monitoring process, which is defined using ad-hoc monitoring rules. Based on the findings of the Audit component, the **Certification** component provides evidence-based security reporting and certification tailored to the needs of the different stakeholders (senior management, external auditors, regulators). The **Security Metrics** module stores into a database several metrics addressing the Confidentiality, Integrity, Availability (CIA) principles, including custom metrics tailored to the uses cases of the FISHY pilots.

This deliverable presents the first achievements in the first iteration (IT-1) of the development of the WP4 components. All the formal models and the components of both EDC are currently in the design phase. On the other hands, a first implementation of the components of the SACM is already available and they are in the first integration phase. In the second iteration (IT-2), the initial design of both EDC and SACM will be extended according to the research results achieved in FISHY. The EDC will be implemented and integrated, the SACM will be enhanced, and these improvements will generate a second generation of the components that will be later integrated. In IT-2, integration among EDC, SACM, and the other WPs components will be explored even further in Task T4.3. Moreover, the data models will be finalized and validated. The definition of the use cases will help concentrate the effort on a more focused subset of concepts that could be adequately addressed during the project lifetime.

This deliverable also enables collaboration with WP5 for the definition of the HLP model. Finally, it allows defining, in collaboration with WP3, the interactions between the EDC of the Trust and Incident Manager, and the interactions between the Security & Privacy Data Space Infrastructure and the SACM.

1 Introduction

Helping security administrators perform their task is vital, given the recent reports providing evidence that human error contributes to more than 95% of successful attacks against company infrastructures. FISHY aims at automating several security-related operations, using the most advanced results of the research.

In this context, the WP4 provides two main functions that are part of the overall FISHY objectives.

It contributes methods to keep the security controls correctly configured from a set of high-level policies that indicate the security requirements wanted for the target ICT infrastructure. These features are provided by the Enforcement and Dynamic Configuration (EDC) component, whose design will start from supporting Network Security Controls, with a focus on the virtualized environment and software networks.

Furthermore, WP4 provides the mechanisms for auditing and assessing complex ICT systems within the scope of the FISHY project by building a real-time, cross-layer Evidence Collection Engine that measures the security based on a set of defined metrics. It also provides an evidence-based certifiable view to ensure and certify that the desired security policies are correctly implemented in the target ICT infrastructure. These features are provided by the Security Assurance and Certification Management (SACM) component.

1.1 Purpose of the document

This deliverable provides the initial design of the two WP4 components, The Enforcement and Dynamic Configuration (EDC) and the Security Assurance and Certification Management (SACM). It provides the results of the first iteration (IT-1) of the design of the architecture and workflows of the two WP4 components and the interactions with components in other technical work packages.

Furthermore, this deliverable introduces the data models needed to properly exchange data among WP4 components (and their sub-components) and with the components in the other technical WPs involved in a set of broader workflows. The design of some of these models is already in the initial definition phase. However, they will not be presented here in detail as they are not final and need to undergo a validated phase. The final version of these models will be delivered in the second iteration (IT-2) of these components.

Finally, whenever available, this document anticipates information from the preliminary studies done in T4.1 and T4.2 that will allow an effective start of the T4.3 (Integrated Security and Certification Manager) activities related to the implementation and integration of these components in the FISHY overall infrastructure.

The two components see different levels of maturity at IT-1. On the one hand, EDC is in the design phase, its features and requirements have been highlighted, its architecture, workflows, and interactions with the components in the FISHY overall architecture have clarified. EDC strongly relies on the data models that will be adopted in FISHY, which strongly depend on use cases and their requirements. Therefore, existing components from the background of the project partners are not directly usable in the FISHY environment, which is much more general and complex than the scenarios used by WP4 partners to implement their background. Effort has been made to ease the integration phases that will start with T4.3 (e.g., by studying the components APIs). However, EDC and its components are not ready to be integrated. EDC implementation will start with T4.3 but more research is needed to solve the issues related to automatic enforcement and dynamic configuration.

The risks are under control, as the partners' expertise and past experience appear able to allow a proper management of the task.

On the other hand, SACM relies on much more mature components that can be inherited from the partners' background. Therefore, IT-1 has produced the architecture and workflow of initial components. The initial integration of SACM will start with T4.3 and it will be conducted in parallel with the research that will improve the existing components and produce more advanced and useful version of the tool, which will be delivered then integrated during IT-2.

Accordingly, we have split the milestone related to WP4 M4.1-M4.4 into sub-milestones tailored for the two components.

- M4.1(MS16) FISHY Sec.&Cert. block components ready for integration (IT-1) WP4 M9 D4.1
  - M4.1.a.1 FISHY SACM design ready
  - M4.1.a.2 FISHY SACM block components ready for integration (IT-1)
  - M4.1.b FISHY EDC design ready
- M4.2(MS17) FISHY Sec.&Cert. block integrated (IT-1) WP4 M13 D4.2
  - M4.2.a FISHY SACM block integrated (IT-1)
  - M4.2.b FISHY WP4 data models ready
- M4.3(MS18) FISHY Sec.&Cert. block components ready for integration (IT-2) WP4 M26 D4.3
  - M4.3.a FISHY SACM block components ready for integration (IT-2)
  - M4.3.b FISHY EDC block components ready for integration
- M4.4(MS19) FISHY Sec.&Cert. block integrated (IT-2) WP4 M30 D4.4
  - M4.4.a FISHY SACM block integrated (IT-2)
  - M4.4.b FISHY EDC block integrated

This approach will allow a better control of the evolution of the output related to the two components of this work package and anticipate the risks in a finer grained way.

1.2 Relation to other project work

The work presented in this deliverable shows two main types of interaction with the other FISHY work: at the architecture level, with WP2 and with all the technical work packages.

At the architecture level, WP2 provides in T2.3 the design of the overall architecture, workflows, and it coordinates the interactions among all the technical work packages. All the results presented in this deliverable have been developed based on the results in this task. Moreover, the WP4 design has been validated against T2.3 results, agreed with partners in this task, and integrated into the revised FISHY general architecture.

At the technical level, the EDC strongly depends on the WP5. First of all, the high-level input policies are expected to be obtained by compiling the intents. The Intent-based Resilience Orchestrator & Dashboard (IRO) is in charge to perform this work. Moreover, after the configurations are generated for all the NSF in the target landscape (by refining the high-level security objectives), the administrator can decide to use the EDC functions to deploy them into the target security controls. This operation will be mediated by the Security Infrastructure Abstraction (SIA), which will expose API to manage the landscape. Moreover, the EDC depends on the threats detected by tools in WP3. According to the current general FISHY workflows, the report of the threats provided by the TIM will include additional information (e.g., high-level policies) that are passed to the refinement process together with the ones coming from the IRO.

Moreover, the SACM at the technical level also depends on the WP5. The monitoring mechanisms that will be developed in WP5 will form a primary base for building the Evidence Collection Engine, the core component of the SACM. Leveraging these mechanisms will forge the Evidence Collection Engine to monitor the ICT infrastructure's critical components and processes. Moreover, the SACM will publish the evidence-based, certifiable view of the security posture of the ICT system to the IRO. Data and event capturing that will guide the Evidence Collection Engine will be made through the Security & Privacy Data Space Infrastructure (SPI).

1.3 Structure of the document

This document is structured in 4 major chapters.

- **Chapter 2** presents the high-level view of the WP4 components and interactions with other work packages and the list and explanation of the major design decisions made in T4.1 and T4.2, which affect the overall FISHY architecture.
- **Chapter 3** presents the design of the EDC, the architecture and the workflow, and the design of the individual components.
- **Chapter 4** presents the design of the SACM, the architecture and the workflow, and the design of the individual components.
- **Chapter 5** draws conclusions and sketches the future steps in WP4 design and how the results in this deliverable will be used in future project activities.

1.4 Glossary adopted in this document

In this deliverable, we will use a set of terms whose meaning needs to be clarified to avoid ambiguities.

The desired security policies can be represented using different abstractions and formalisms. In WP4 we will use (note that in WP5 also intents are used):

- *High-Level Security Policies (HLP)*, the highest abstraction level in WP4, specify security requirements without explicitly referring to the security controls, i.e., the NSFs that will implement them;
- *Medium-Level security Policies* (*MLP*, also referred to as *abstract configurations*) specify the necessary NSF-specific instructions to configure a target NSF using an NSF-independent language (e.g., the iptables configuration but written for a generic filtering device that offers all and only the security features iptables offers);
- *Low-Level configurations settings* (also referred to as *configurations*) specify the NSF-specific configuration settings (i.e., that once deployed, are correctly accepted and enforced by the target NSF). Still, they may be wrapped with some FISHY-related management information. Since the EDC will start its focus from network security controls, possible extensions for application security will be evaluated for IT-2.

We will use in this document the term landscape, which is an alternative shorter form to indicate the part of the target ICT infrastructure that is managed by the WP4 components (it may be a multi-domain or single domain or a smaller portion). Therefore, the landscape is the part under the control of the refinement and assurance features offered by WP4.

The problems that WP4 addresses in general and in the FISHY ecosystem are listed below:

- The *refinement* is the process that translates high-level policies (obtained from the compilation of the intents in WP5) into medium-level policies for a set of NSF in the target landscape.
- The *translation* is the process that transforms the medium-level policies (e.g., an abstract policy for a generic packet filter) into the low-level configuration settings of the NSF in the target landscape (e.g., the iptables configuration file).

- The *remediation* is the process that, as a reaction to a security-relevant event (attack, known 0-days, etc.) or a modification in the target landscape, proposes changes to the landscape scenario (e.g., by adding the proper NSFs to the topology) or the abstract configurations of the NSFs in the landscape.
- *The enforcement* is the process that deploys the configurations into the NSFs in the landscape. It performs all the necessary operations to force the NSF to start enforcing the deployed configuration.
- *The registration* is the process that allows adding a new NSF into the FISHY Catalog of the NSF available for the security enforcement in a given landscape (or removing an existing one or modify the registered information).
- The *Security metrics* are a set of related measurements enabling qualification regarding security aspects under the scope of FISHY. The former must be compliant with standards - we will address security policies in the Confidentiality, Integrity, Availability (CIA) form - and with each use case of the FISHY project.

2 Design of WP4 components

This section presents the preliminary decision made in the first months since the WP activation, the main design principles, and the requirements that apply to the WP4 components.

The first analysis of the components and their features has proved that **the EDC and the SACM are two independent components**. Even if they are both developed within the WP4, they neither share components nor functionality. Apart from interacting with the repositories that store models and instances, these components and their workflows have no intersections. Accordingly, their design will be done in parallel. When modelling the interactions between these components, each component can see the other as a black box and call it via the API. This scenario and decision simplified the design and management of the two active tasks in WP4.

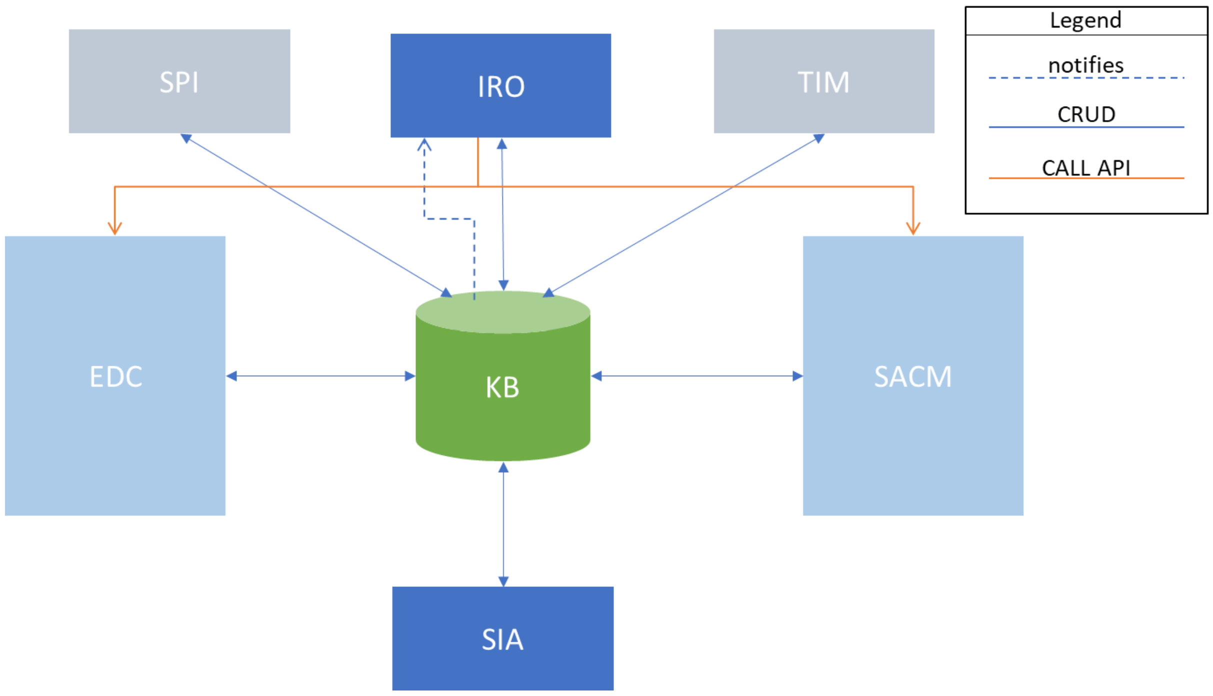


Figure 1. The role of the Knowledge Base in the WP4 architecture.

Another important decision made during the first months of the WP4 is related to the role of the repositories, which in the FISHY project have been named Knowledge Base (KB), reusing the term initially introduced in WP5 for storing data related to intents.

Figure 1 shows all the components that may have to interact with WP4 components.

- The IRO Dashboard will trigger several operations from EDC and SACM.
- The IRO Intent Compiler will produce HLP as a result of the compilation of the manually written intents.
- TIM will also produce HLP because of the threats it identifies and the reasoning processes it performs to identify mitigations. Finally,
- SIA will manage information about the ICT infrastructure it abstracts.

These components interact several times during the implementation of the FISHY workflows and they will exchange a large amount of data (e.g., sets of security policies). Therefore, using a set of component-specific storages or proposing a model where a component consumes the data it receives in input from another FISHY component did not appear appropriate. We are thus proposing in this document a central KB where specific components can perform CRUD (Create, Read, Update, and Delete) operations. In IT-2, we will evaluate if KB will be organized in sections and grant component access depending on the data models they may have to read or modify.

From the WP4 perspective, **the FISHY Knowledge base is the logical aggregation of all the instances of the data models** used by WP4 components to exchange information among components. It can be thus considered as central storage that aggregates all the data. This approach has also been extended to the interactions of WP4 component with the components developed in other work packages. It will likely be adopted also in WP3 and WP5 for the internal data exchanges. Nonetheless, it does not necessarily imply that these components will be implemented as a single centralized database. Indeed, this high-level design decision is compatible with the implementation of individual data store. During the implementation and integration phases of the project, several independent decisions can be made to fulfil the requirements of the individual tasks. In the following sections, some figures and text description may depict and refer to individual sections of the knowledge base (e.g., the NSF Catalog). However, from the logical point of view, they all belong to the central KB.

The advantages considered associated with this approach are:

- the high-level design of the FISHY is simplified, resulting in a clearer and perhaps, understandable architecture;
- all the features of KB can be uniformly defined for all the components, regardless of the WPs, also easing the definition of the KB;
- the same technology could be used to implement every one of the data model repositories;
- the components are relieved from storing the information they produce, maintaining their internal databases and history, rather a central GIT-like repository may be enough for the project purposes;
- there is no need to pass the actual data among components, instead, the references to data can be exchanged, leaving the components the responsibility to only collect the minimum and necessary data. This can be an advantage considering the potential size of the data to exchange
- KB can implement optimized query strategies (such as filters or aggregations), perform preprocessing before passing data to other components, etc.
- even the queries launched are expected to be simpler and more homogeneous amongst components.
- components may register to receive notifications of changes to the data instances they are interested in. For instance, the IRO Dashboard can be notified about all the changes in information that need to be properly presented to the users before making decisions;

This deliverable is the first iteration of the design of the WP4 components. Hence, the problems related to applying the FISHY **solutions to a multi-domain have not been fully explored yet**. The decision made is that the EDC will be deployed in a single realm for what concerns the initial phases. In the FISHY context a realm is a single domain of administration. A realm is characterized by a landscape, which describes the logical (and possibly physical) arrangement of the networked infrastructure, and a security policy. Different realm may have conflicting interests that introduce additional requirements that the FISHY project may have to address (e.g., isolation). A more precise definition of realm and how FISHY will address the additional issues is under design in WP2. EDC will be considered a service that the intra-realm IRO accesses to require performing refinement, remediation, and enforcement functions for the same realm. The separation can be enforced by forcing the EDC to access and store data from an intra-realm repository. The risk of significant design changes has been evaluated as extremely limited. Some issues may appear in case the FISHY Knowledge Base would be shared among different realms.

Nonetheless, as anticipated, solving this authorization problem cannot even be considered a research issue nowadays. In the opinion of the WP4 partners, it is an implementation issue that can be solved later, when FISHY results will be packaged as a product that works in multi-domain environments. The following months will see the refinement of the architecture that will fully address this problem as well.

The design of SACM will be based around a distributed mechanism that will collect data provably. An auditing mechanism will evaluate the former towards security metrics tailored around user cases in the supply chain. The SACM will provide a certifiable view of the latter security metrics to the dashboard, informing the end-user of violations of monitoring rules against streams of collected runtime events.

As a final consideration, the design of the WP4 components started from an important consideration coming from the real world. If we want the FISHY results used in a real environment, we must refrain from designing fully automatic systems. Said differently, human intervention is necessary and beneficial, and **all the operations must be explicitly authorized by the administrators** that have the policy enforcement responsibility in the target domain.

3 The Enforcement and Dynamic Configuration

3.1 Architecture and components

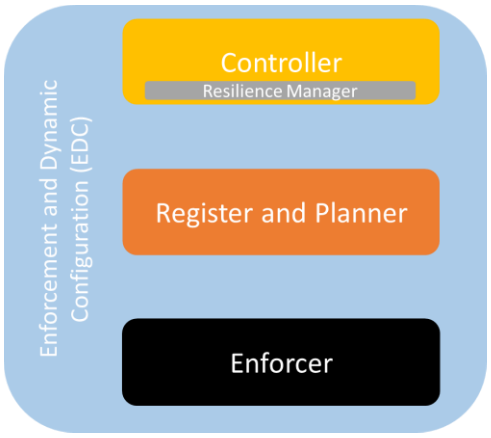


Figure 2. EDC: high-level view of the components.

The EDC is composed of three independent modules (see Figure 2).

The **Controller** is a network controller mapping from the network-specific cyber threat solution to actual NSF deployment and configuration (i.e., refines high-level policies into configurations). In short, the Controller performs the refinement of the high-level policies into medium-level configurations. Inside the Controller, the **Resilience Manager** was initially introduced to react after IRO notifications of attacks/anomalies detected by TIM. This component, which has been mentioned in the project proposal, has been removed from the FISHY EDC architecture as its features were absorbed, already in the last version of the proposal, by the Controller (see Section 3.4 below).

The Register and Planner (or simply **Planner**) is the component that manages the information related to the NSF available in a given domain (e.g., the list of available firewalls, VPN gateways, etc.). It implements the functions that allow the NSFs to register (i.e., providing a formal description of their security capabilities, which can be used to enforce security policies) using open standard interfaces. Therefore, it will manage a catalogue of NSF and implement functions to optimally query, compare, and select NSFs based on their capabilities (see Section 3.5).

The **Enforcer** is the lower-level block of the EDC. It continuously reconfigures the ICT system via the existing NSFs based on the available capabilities (see Section 3.6). In IT-1 we have considered the network part of the ICT system. In IT-2, we will consider extending it to application security. It implements the translation functions, i.e., it converts medium-level policies in the knowledge base into device-specific configuration settings. Moreover, it uses SIA to connect WP4 policy functions with the actual infrastructure. It pushes the configurations it translates by interacting with SIA through the means provided by this WP5 tool. It may also request SIA to implement changes in the ICT in case policies are not enforceable or to remediate to risk identified by any of the WP3 threat intelligence tools.

3.2 Workflow

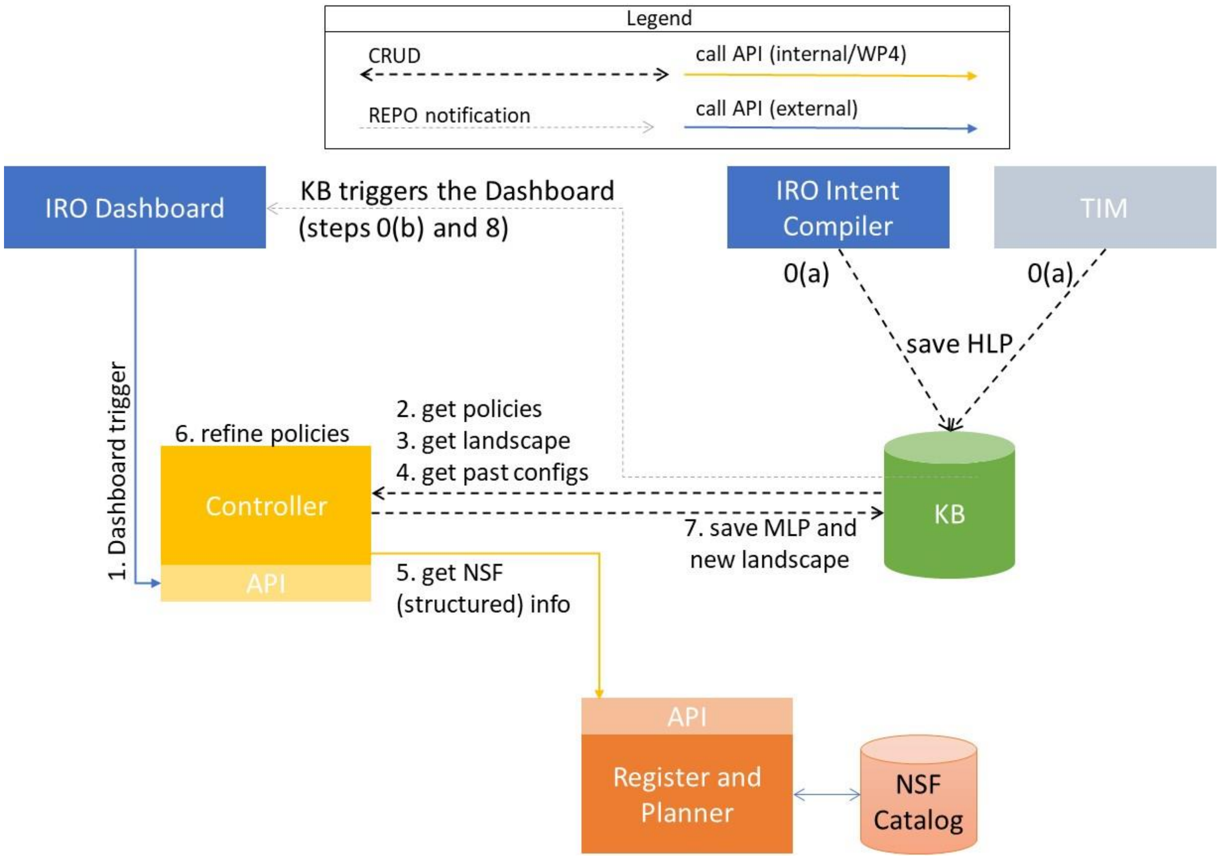


Figure 3. EDC Workflow (Part I)

The workflow performed by the EDC is described below, each item of the next numbered list corresponds to a workflow step, which is depicted in Figure 3 (steps 1-8) and Figure 4 (steps 9-19). The steps 6 and 14, since they are computations that performed internally at a single component, have not been showed explicitly in the figures.

0. The Dashboard is triggered by KB (step 0(b)). This trigger happens when a new HLP policy is available in KB, e.g., because a new intent has been written then compiled by the IRO or because TIM has added a new HLP as a complement to a threat report (steps 0(a)).)).
1. The Controller is triggered by the Dashboard, using the Controller API. We expect this to be an operation requiring action from an administrator, that is, to be manually triggered. However, as an alternative, this trigger can be automatically generated if, for example, an administrator has explicitly authorized this automatic task (e.g., by configuring some Dashboard automation features) in advance. The Dashboard provides the reference to the set of instances of HLP to be considered during the refinement. When not clear from the scenario, the Dashboard may add information about the domain where the policy needs to be refined (step 1).
2. The Controller queries KB to retrieve all the HLP instances the Dashboard referred to with its call.
3. The Controller queries KB to get information about the current ICT system, i.e., the current landscape, including both standard nodes (VNF) and security controls (NSF)
4. The Controller may query KB to get information about the MLP currently implemented by the NSFs in the current landscape.

5. The Controller queries the Planner to collect information about all the NSFs available in the domain. Instead of directly querying KB, the Planner is queried because we expect that capability data will require to be prepared, formatted, and interpreted. Moreover, the Planner may implement higher-level features like comparisons, filters, etc.
6. The Controller refines the HLP into MLP. If some non-enforceability issue manifest, the Controller also proposes a new landscape logical layout (e.g., adding new NSFs to compensate for lack of capabilities or redirecting a flow to pass into additional security controls).
7. The Controller stores the refined MLP (and the new landscape layout, if any) into KB. The MLP contains a reference to the set of HLP from which they have been derived, and each MLP refers to one or more specific NSF instances.
8. KB notifies the Dashboard about changes in the stored MLP objects and, possibly, the new landscape logical layout proposed by the Controller.
9. The Enforcer is triggered by the Dashboard, using the Enforcer API. We expect this to be an operation requiring action from an administrator, that is, to be manually triggered. However, as an alternative, this trigger can be automatically generated if, for example, an administrator has explicitly authorized this automatic task (e.g., by configuring some Dashboard automation features) in advance. The Dashboard provides the reference to the MLP instances that need to be translated.
10. The Enforcer queries KB to retrieve all the MLP instances the Dashboard referred to with the API call.
11. The Enforcer queries KB to get information about the current ICT system, i.e., the current landscape, including both standard nodes (VNF) and network security controls (NSF)
12. The Enforcer may query KB to get information about the MLP currently implemented by the NSFs in the current landscape.
13. The Enforcer queries the Planner to collect information about all the NSFs referred to by the MLP. In this case, instead of directly querying KB, the Planner is queried because of the higher-level functions it implements.
14. The Enforcer translates the MLP, that is, converts MLP into low-level configuration settings of the NSF.
15. The Enforcer stores the generated configurations into KB. The configurations provide reference to the original MLP and the target NSF instances.
16. KB notifies the Dashboard about changes in the stored Configuration objects.
17. The Enforcer is triggered by the Dashboard, using the Enforcer API. The Dashboard provides the reference to the configurations that need to be pushed. We expect this is an operation manually triggered by an administrator. Alternatively, this trigger can be automatically generated, provided an administrator has explicitly authorized this automatic task (e.g., by configuring some Dashboard automation features) in advance. During the design, the need for two explicit automatic triggers emerged. The first type of trigger will be raised if only new configurations need to be pushed (i.e., there are no changes in the landscape proposed by the Controller). The second type will be triggered in case of changes to be implemented to the landscape. The latter has been evaluated as a very sensitive operation that may also affect performance and should not be automated.
18. The Enforcer calls the SIA to enforce the changes in the landscape (if any), pushes the configurations in the proper NSFs, reconfigures and restarts. According to the current status of the design of the SIA, this operation should be performed by accessing the SIA-exposed API.
19. The Enforcer saves into KB the information obtained by the SIA about the actual status of the configuration of the NSFs (e.g., that configurations have been properly deployed and the security control restarted with the new configurations).

Alternatively, instead of performing the last two configuration steps (step 18 and 19), the SIA may be triggered by the IRO with the information about the new landscape and configurations to push. This hypothesis has not yet discarded. It will be evaluated during the definition of the overall FISHY workflow and features. However, it is worth mentioning that this decision does not affect the WP4 components described in this deliverable.

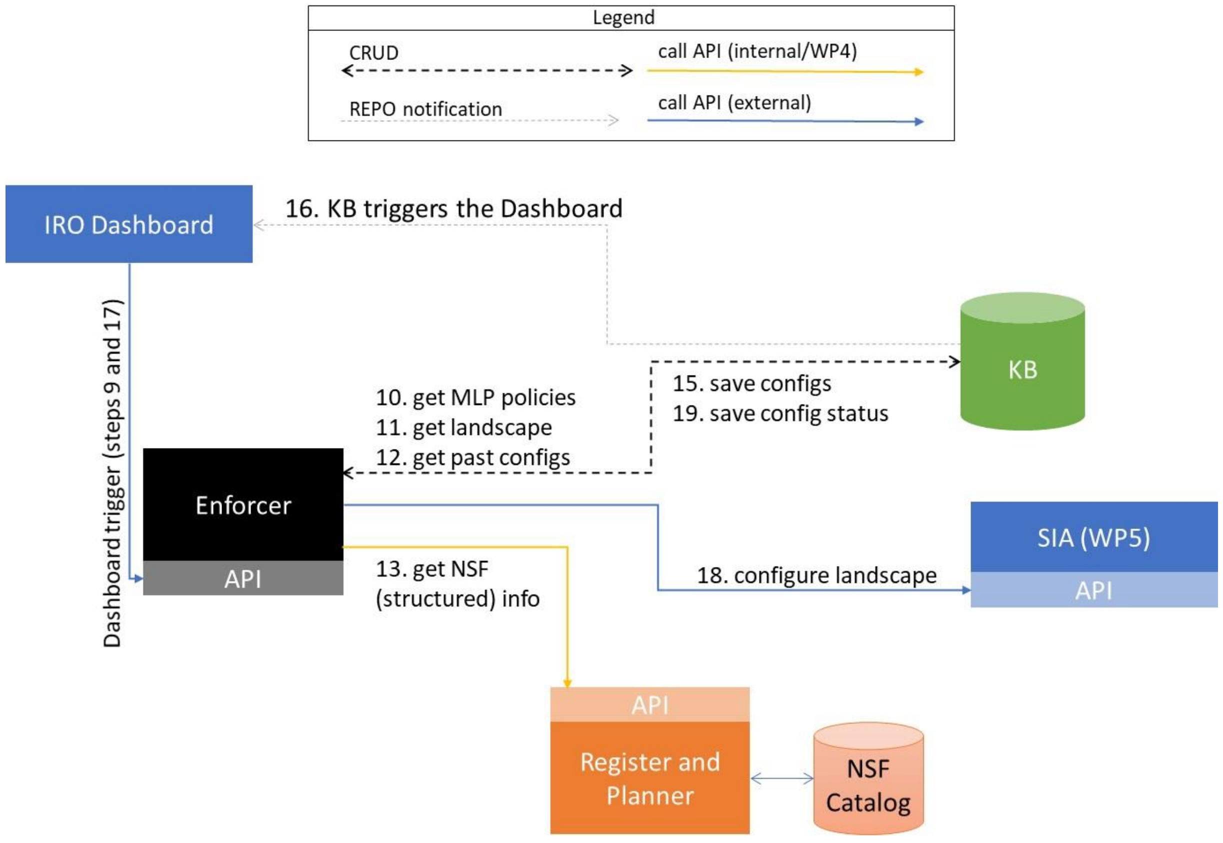


Figure 4. The EDC workflow (Part II).

3.3 Data models

3.3.1 High-level policy model

The high-level policies (HLP) are security requirements that administrators want to be enforced in their ICT system.

Being the output of the IRO Intent Compiler and the TIM, and the input to the refinement processes, HLP have:

- On the one hand, HLP should be abstract enough to allow a proper representation of threats and map the result of the compilation of the intents.
- On the other hand, however, they should be concrete enough to make it possible to refine them into MLPs. They should be formally modelled and contain enough semantics to allow a process to understand what security controls are needed to use and how to configure and enforce them.

Currently, the approach that the FISHY project is exploring is to model this type of policies as authorization statements, one of the most used approaches for describing high-level policies, which has already been favourably used in the EC-funded project SECURED[1].

The idea of the HPL is to define a policy by using high-level security requirements (e.g., ``do not download any malware'', ``do not connect to illegal sites'').

HPL is composed of statements with the following structure:

[subject] action object [(field_type,value) ... (field_type,value)]

where:

- ***subject*** is the user who needs to access or perform some operation on an object (e.g., employee, family member) and may be omitted if the policy is applied to the user that defines the HPL;
- ***action*** is the operation performed on the object (e.g., protect, permit access, enable);
- ***object*** is the entity (i.e., a resource such as e-mail scanning, Internet traffic, P2P traffic) target of the action (e.g., authorize access);
- ***(field_type,value)*** is an optional condition that adds specific constraints to the action (e.g., time, content type, traffic type). The value part is a string with a specific format depending on the field type.

On the one hand, this model is very general and accommodates almost all the types of policies that we may want to support in FISHY. On the other hand, this set of unlimited possibilities will be characterized by the project's use cases. In IT-1, HLP will mainly cover network security policies.

3.3.2 Low-level configuration settings (aka configurations)

The low-level configuration settings have a primary requirement. They must be understood and **enforceable by the target NSF**.

The enforceability is guaranteed because they will be obtained as the translation of MLP that are aware of the features (i.e., the capability) owned by the NSFs. Therefore, in FISHY, we aim at properly wrapping the raw configuration settings, which can be pushed without further processing into the NSF, with essential management information.

A very simple example of what can be a useful wrapping of configurations is presented below:

```
<llconf nsf=”iptables” format = ASCII-text” filename=”iptables.rules4”>
        -A FORWARD -m state -m state --state ESTABLISHED,RELATED -j ACCEPT
        -A FORWARD -p tcp =d 192.168.0.1/24 -dport 80,443 -j ACCEPT
[..]
</llconf>
<llconf nsf=”fw1” format = “b64encoded-binary” filename=”bin.rules”>
        bajh234jsdf02nsdkf[..]shdjf==
</llconf>
```

3.3.3 The Security Capability model and Medium-level policies (aka abstract configurations

The security capabilities are an abstraction of what a (network) security control can do in terms of security policy enforcement. Security capabilities have a central role in several EDC operations.

[1] https://www.secured-fp7.eu/

First, in FISHY, we have decided that being coherent with international standard and models is a method to increase the visibility and impact of our research and results. Therefore, we decided that the security capability model must follow the initial design of a capability model provided by the IETF I2NSF (Interface to Network Security Functions) Working Group[2]. Quoting from a draft produced by this working group:

> *"Security Capabilities are independent of the actual security control mechanisms that will implement them. Every NSF should be described with the set of capabilities it offers. Security Capabilities enable security functionality to be described in a vendor-neutral manner. That is, it is not needed to refer to a specific product or technology when designing the network; rather, the functions characterized by their capabilities are considered. Security Capabilities are a market enabler, providing a way to define customized security protection by unambiguously describing the security features offered by a given NSF."*

The I2NSF proposed an Information Model intended to provide:

> *"a formal description of NSF functionality. Capabilities enable unambiguous specification of the security capabilities available in a (virtualized) networking environment and their automatic processing by means of computer-based techniques. This includes enabling the security controller to properly identify and manage NSFs, and allow NSFs to properly declare their functionality so that they can be used in the correct way."*

The capability model focuses on six main concepts derived from the analysis of the generic features security controls offer to specify configurations, as already proposed in the I2NSF WG draft. The first four concepts describe the creation of a rule.

- The *conditions* describe the features available at the NSF to identify the element (e.g., the traffic) to which to enforce (e.g., conditions on the IP addresses or regex on the HTTP MIME type).
- The *actions* are the operations that an NSF can perform on individual elements (e.g., deny packets or encrypt flows).
- Finally, the model describes the *events* that allow triggering the rules' evaluation for specific classes of security controls. Currently, the need for events has not been highlighted in the FISHY use cases.
- The last field is related to the *condition clause*. It serves to express that a rule is activated when all the conditions evaluate to true (DNF, Disjunctive Normal Form), or just one condition is satisfied (CNF, Conjunctive Normal Form).

The following two concepts explain how to build the desired security policy from individual rules.

- The *resolution strategies* describe how the control behaves when more than one rule applies to the same entity. For instance, the first matching rule applies the rule's action at the highest priority when more rules apply.
- Support for *default actions* allows determining how the control behaves when no rule matches the element. For instance, it is essential to know if the control allows specifying the "deny all" default action, as implemented by most firewalls, or not.

Figure 5 describes the current status of the design of the security capability model: the yellow classes represent the original classes defined in the I2NSF draft, and the light blue ones are the new classes we have introduced in FISHY to support the refinement and translation processes.

[2] https://datatracker.ietf.org/wg/i2nsf/about/
https://tools.ietf.org/html/draft-ietf-i2nsf-capability-05

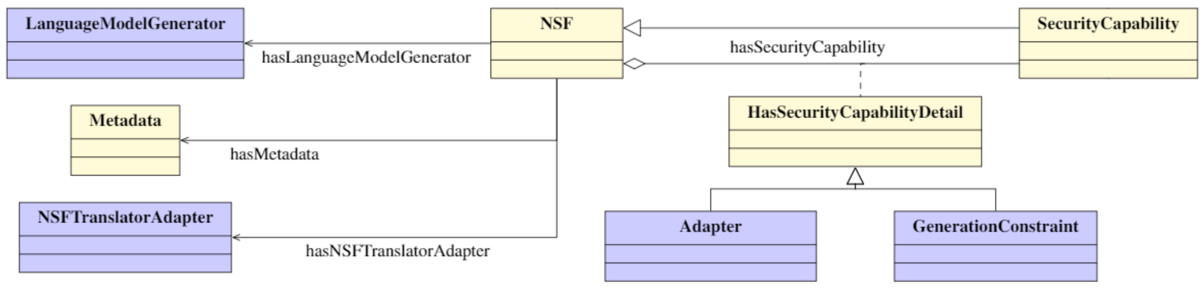


Figure 5. The current Capability model specification.

The security capability model is composed of two classes (*NSF* and *SecurityCapability*), which are abstractions of a Network Security Function and a generic security capability, and an association class (*HasSecurityCapabilityDetail*), to link them according to the *decorator pattern* [1]. This pattern is also helpful to provide additional information (as an association class) when capabilities are "attached" to the decorated objects and are important to define how capabilities map to device-specific settings.

This pattern may be used to add capabilities both statically, when an NSF is described, and dynamically, during network operations. Thanks to the decorator pattern, the model will ease the description of the capabilities by supporting templates, allowing merging them and individually adding and subtracting features from a previous description. For instance, the description of the capabilities of iptables can be determined by extending the description of a generic packet filter and a generic filter on TCP states, i.e., by adding more features to these generic templates. Furthermore, the v2.0 of an NSF can be obtained from the description of the v1.0 of the NSF by adding or removing individual capabilities.

In FISHY, we plan to exploit the capability model to build the **formal model of the abstract configurations** and automate the **translation of abstract configurations** represented in MLP into the low-level configurations settings (i.e., to implement some of the functionalities of the Enforcer). To this purpose, the I2NSF information model has been enriched to include classes that allow the generation of the abstract language of an NSF described using its capabilities and translating MLP into low-level configuration settings.

As an additional requirement, we have selected the EDC design to minimize the semantics that the refinement and translation processes have to know to work with NSFs capabilities. In particular, the only hardcoded semantics must be the one associated with the previously mentioned 6-tuple. The internal implementations of the Controller and Enforcer must only be aware of the six categories of capabilities that have been inherited from the I2NSF WG approach to provide refinement and translation services.

Starting from the description of the capabilities owned by an NSF, the extensions we are proposing to the I2NSF model allow generating an abstract language that can be used to specify valid configurations for the described NSF. As a requirement for the work in FISHY, this generation must not rely on the NSF specific code. As reported before, the MLP are **vendor-independent specifications of the policies** that a given NSF must enforce. In short, MLP expresses configuration directives for the NSF but use a generic language instead of the NSF-specific language and settings. Indeed, the security capabilities describe the features a security control offers in terms of security policy enforcement. Thus, capabilities can be mapped to the parameters, fields, and options that can be enabled and configured by means of a set of configuration directives in the target NSF language. Therefore, in FISHY, we have investigated the possibility of generating an NSF's abstract language from the description of its capabilities. The current design includes the *GenerationConstraint* class, a subclass of the association class *HasSecurityCapabilityDetail* to customize the generation of the language (e.g., to customize enumerates, limit the available options). The *LanguageGenerator* class is a singleton that, starting from an NSF instance, generates its abstract language. It also interprets the *GenerationConstraint* instances that have been used to associate the capabilities to the NSF.

The next idea concerns the possibility to build an MLP translator on the capability model. In practice, when associating capabilities to an NSF, the modeler can also specify how the capabilities are expressed in the NSF-specific language. For instance, the modeler could explain that iptables supports conditions on destination ports, and they are written using the command *--sport.* Moreover, the modeler is allowed to specify other dependencies related to aa specific capability. In the past example, the modeler can explain that, to use the *--sport* options, the conditions *-p tcp* or *-p udp* need to be also explicitly specified written. Hence, an automatic tool can perform the translation automatically without writing additional code. Therefore, given the MLP of an NSF written in its abstract language, the extensions to the I2NSF model proposed in FISHY allows translating configurations into the NSF-specific configuration settings. This translation must be implemented to support the addition of new NSFs without requiring the having to write new code. The current design includes the *Adapter* class, a subclass of the association class *HasSecurityCapabilityDetail,* to customize the translation phase. The *NSFTranslatorAdapter* class is a singleton. It starts from an NSF instance and generates the configurations based on the information associated using *Adapter* instances during the specification of the capabilities of the NSF.

Both the language generation and translation operations adopt the *adapter pattern* to allow NSFs to be uniformly accessed to perform the translation into low-level settings, even though they may have different capabilities, languages, and interfaces. This pattern allows employing a unique implementation of the translator without the need for NSF-specific code.

3.3.4 Landscape scenario

A formal description of the ICT system where enforcing policies is needed by several tasks in the FISHY project. We need to represent individual network nodes (e.g., physical, virtual machines, lightweight VMs like dockers), the services they provide, and the global topology and interconnections among the nodes.

We are currently investigating if the graph-based representation used by the Network Function Virtualization (NFV) orchestrators used in the project is expressive enough to represent the landscape. Otherwise, they will be complemented with the additional information not available in these models.

For this reason, the requirements related to this data model are investigated and formalized. The decision of the actual model is currently less urgent and has been delayed until the design of SIA will be finished.

3.4 EDC Component: Controller

3.4.1 Architecture, Functionality, interactions

The Controller implements the following functions:

- refine HLP into abstract configurations for a fixed (static) landscape, including services and NSFs (i.e., it cannot propose modifications to the landscape) and report about non-enforceability issues (i.e., when the desired policy cannot be enforced with the included landscape),
- refine HLP into abstract configurations for a landscape with full control on the NSFs it can use. For instance, the Controller can select the NSFs that best implement the policies and decide where to place them according to the devices to protect. It can:

  - start from a landscape where only services and other non-security nodes are described and propose all the NSF to use to enforce the policy,
- start from a landscape where services and NSFs are included and
  a. configure the included NSF and
  b. propose some additional NSF to remediate non-enforceability issues. It will perform non-enforceability remediation, i.e., propose changes to the landscape and to allow enforcing the desired HLP.
- (optionally) optimize the placement of the NSFs needed to enforce an input HLP.

We will investigate in FISHY approaches that, starting from an already configured ICT infrastructure, perform an "incremental" refinement. These approaches, after a change in the landscape or the HLP, minimize (according to ad hoc objective functions) the changes in the landscape and configurations (e.g., minimize the impact on the landscape, minimize the number of NSFs that require reconfigurations, or other service and performance indicators). This research objective has proven to be a very hard problem, which is very interesting from the research point of view but not essential for FISHY to be successful. Therefore, the adoption in FISHY will be evaluated in IT-2.

The research issues to be addressed strongly depend on the design and the successful development of the security capability model. The capability model is the foundation of a set of refinement models able to manage different security requirements and support different types of security capability (e.g., conditions and actions from a variety of existing preventive security controls, including packet filters, stateful firewalls, proxies, channel protection, data storage, VPN gateways, …).

3.4.2 Black-box modelling and API

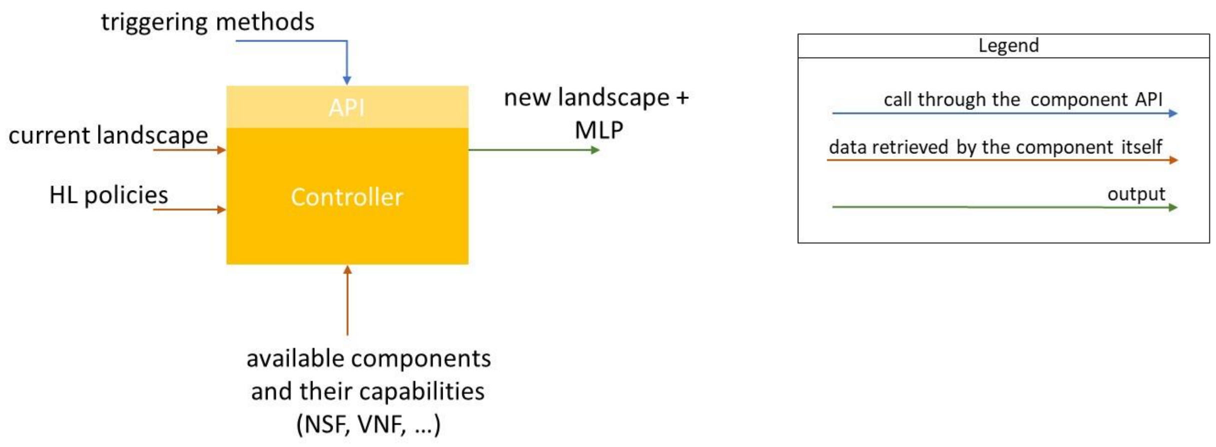

triggering methods
API
Controller
current landscape
HL policies
new landscape + MLP
available components and their capabilities (NSF, VNF, …)
Legend
call through the component API
data retrieved by the component itself
output

Figure 6. The Black-Box view of the Controller

Figure 6 shows the black-box model of the Controller. The Controller needs to be configured to know:

- where to access KB to retrieve the HLP (e.g., the URL);
- where to access KB to retrieve the landscape;
- where to access the Planner to retrieve information about NSF capabilities.

After the initial analysis of the component features, we determined that the API will include the following methods:

Method	Description
refine(<HLP_set_ref>)	Refines the set of HLPs found in the repository at *HLP_set_ref* on the current landscape (e.g., obtained from the repository with the getCurrentLandscape() call)
refine(<HLP_set_ref>, <landscape_ref>)	Refines the set of HLPs found in the repository at *HLP_set_ref* on the landscape found in the repository at *landscape_ref*.

Optionally, depending on the research results we can achieve in the project lifetime, the controller may also provide the following API methods:

Method	Description

refine((<HLP_set_ref >, <delta_ref>)
refine(<HLP_set_ref>,<delta_ref>, <landscape_ref>)
	Refines the *delta_ref* HLP, a data structure that provides information about the changes to the baseline HLP policy set (e.g., the HLP that are no longer valid, new HLP to consider, changes in the landscape)

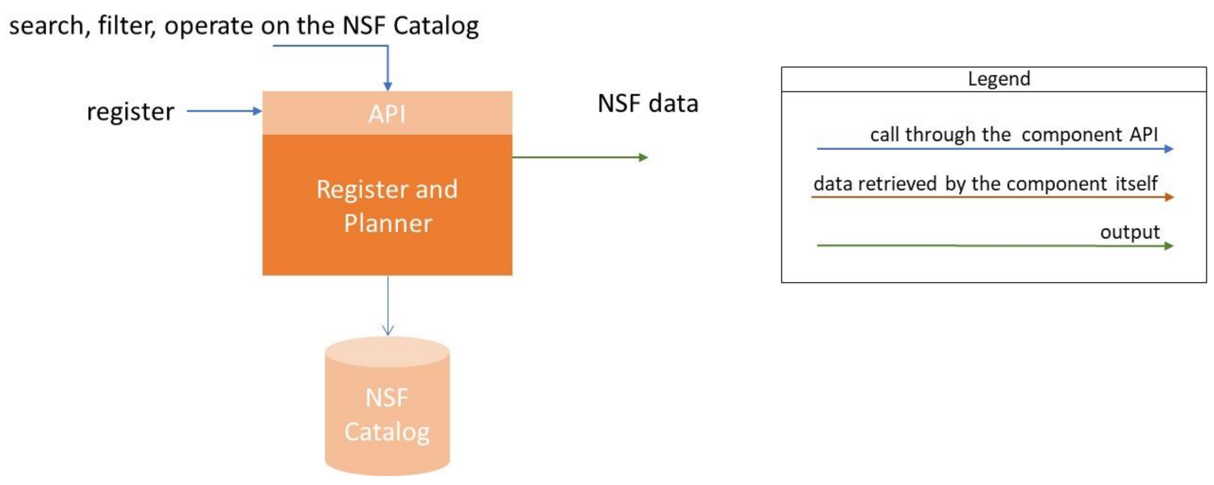
3.5 EDC Component: Register and Planner

3.5.1 Architecture, Functionality, interactions

Regarding the research challenges it has to address, the Register and Planner is much simpler than the other EDC components. Nonetheless, it provides a vital service for the FISHY architecture, including the following features

- allow NSF to registers themselves in an NSF catalogue and propose additional management operation (update description);
- uses, understands, and organizes data according to the security capability model;
- exposes interfaces to query NSFs
    - by capability type (e.g., return the list of NSFs that can enforce packet encryption actions)
    - by individual capabilities (e.g., return the list of NSFs that can drop packets based on IP source address);
- (optional) offer additional services such as comparing the capabilities of two NSFs and returning the differences.

3.5.2 Black-box modelling and API



search, filter, operate on the NSF Catalog
register
API
Register and Planner
NSF data
NSF Catalog
Legend
call through the component API
data retrieved by the component itself
output

Figure 7. The Black-Box view of the Register and Planner.

Figure 7 shows the black-box model of the Register and Planner. This component does not need to access additional information from KB. It will be built to properly manage (register, search, compare) the information represented in the security capability model.

The current development of the components that will use the Planner is still in the early stages. Therefore, there are no precise indications of all the API methods that will be needed. As the first set of high-level methods to be refined in the next months, we list:

- *search(search_string)*
- *compare(nsf1, nsf2)*

3.6 EDC Component: Enforcer

3.6.1 Architecture, Functionality, interactions

The Enforcer implements the following functions:

- translates MLP into configurations by using the modelling features available in the security capability model (see Section 0);
- interacts with SIA to manipulate the landscape;
- interfaces with SIA to deploy the configurations and start/stop/restart the services to force the use of the new configurations.

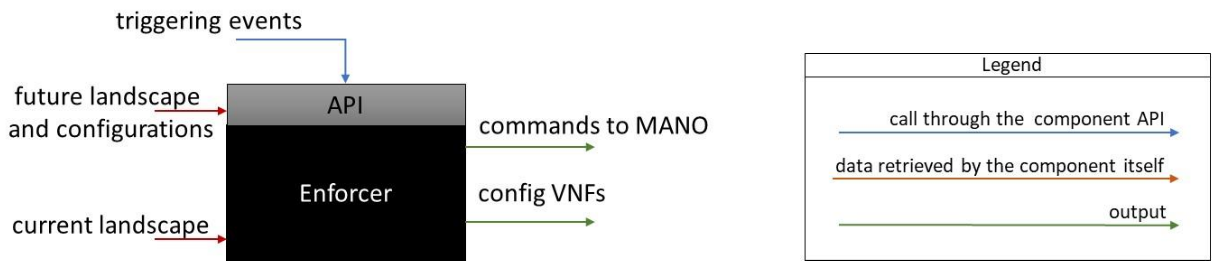

triggering events
future landscape and configurations
current landscape
API
Enforcer
commands to MANO
config VNFs
Legend
call through the component API
data retrieved by the component itself
output

Figure 8. The Black-Box view of the Enforcer.

3.6.2 Black-box modelling and API

Figure 8 shows the black-box model of the Enforcer. The Controller needs to be configured to know:

- where to access KB to retrieve the MLP and save the configurations (e.g., the URL);
- where to access the KB to retrieve the landscape;
- where to access the Planner to retrieve information about NSF capabilities;
- where to access the SIA;

After the initial analysis of the component features, we determined that the API will include the following methods:

Method	Description
translate(<MLP_set_ref >)	Translates the set of MLPs found in the repository at *MLP_set_ref* on the current landscape (e.g., obtained from KB with the *getCurrentLandscape()* call)
configure (<configurations_set_ref >)	Deploys the configurations found in KB in the *configurations_set_ref* into the target landscape
configure_and_restart (<configurations_set_ref >)	
configure (<configurations_set_ref>, <landscape_ref>)	Deploys the configurations found in KB in the *configurations_set_ref* after having initiated the target landscape reported in *landscape_ref*
configure_and_restart (<configurations_set_ref>, <landscape_ref>)	
configure (<configurations_set_ref>, <delta_landscape_ref>)	Deploys the configurations found in KB in the *configurations_set_ref* and changes the landscape according to *delta_landscape_ref.*
configure_and_restart (<configurations_set_ref>, <delta_landscape_ref>)	

It is worth noting that the *configurations_set_ref* may not include the configuration of all the NSFs in the landscape. Only the relevant NSFs may be included (e.g., the NSFs for which the configuration has not been changed or the ones that do not need a new deployment).

4 The Security Assurance and Certification Management

4.1 Architecture and components

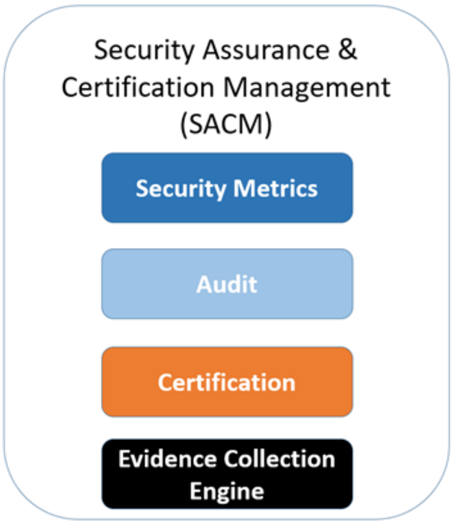
The Security Assurance & Certification Management (SACM) component will be responsible for monitoring, testing and assessing complex ICT systems under the scope of the FISHY project. Through an Evidence Collection Engine developed for the purpose, this component will **audit** critical components and processes of the ICT infrastructure while leveraging monitoring mechanisms developed in the context of the project. Based on that input, the component will provide an evidence-based, certifiable view of the security posture of the ICT system. It will also provide accountability provisions for changes that occur in said posture and the analysis of their cascading effects, supporting the runtime checking based on sets of associated claims and metrics.

The mechanisms developed for this component will also enable and provide the design of audit procedures in ICT systems by considering all ICT components within the supply chain. Finally, the methodology and procedures for the automation of security **certification** are also part of this component, providing different certification models tailored to, e.g., specific security standards, service level agreements or legal and regulatory obligations (e.g., GDPR).

The real-time, continuous assessment of the security posture of the complex ICT systems will be enabled by a purpose-built **Evidence Collection Engine** using Elasticsearch[3] (ELK stack). This component will be responsible for aggregating the required evidence from multiple sources related to the operation of individual components and the overarching processes where these components are involved. This functional group of modules will also include **Audit** and **Certification** functions, leveraging the evidence-based approach of the Assurance solution integrated into the platform. Several built-in **Security Metrics** addressing the Confidentiality – Integrity – Availability (CIA) principles among custom metrics tailored to the use cases of the FISHY pilot's needs will be evaluated by the auditing component.

The SACM is composed of four independent modules (See **Figure 9**).

[3] https://www.elastic.co/



Security Assurance & Certification Management (SACM)
Security Metrics
Audit
Certification
Evidence Collection Engine

Figure 9. SACM: high-level view of the components.

4.2 Workflow

The workflow performed by the SACM is described in Figure 10

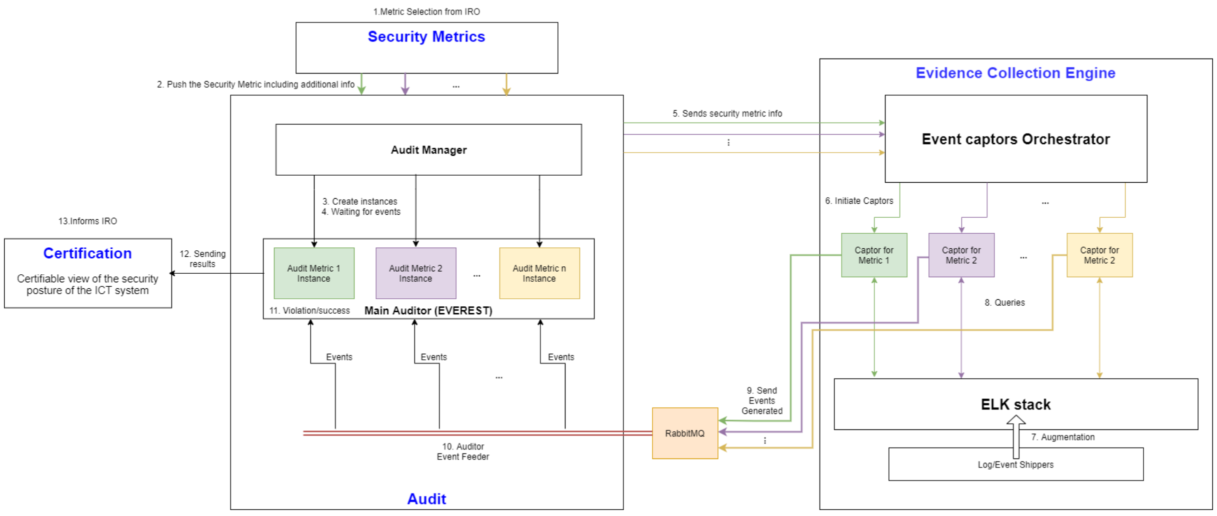


Figure 10. The SACM workflow.

1. Administrator through the IRO dashboard selects to initiate one or more security metrics for auditing. The latter security metrics in this step are described on a high level.
2. Security Metric component pushes the additional information regarding the selected security metrics with a low-level description to the Audit Component. This low-level description may include the type/period (continuous or within time margins) of the evaluation and the type of asset the security metric is referred to.
3. The Audit Manager creates the respective audit instances for each selected Security Metric.
4. Each audit instance starts listening for incoming events from the Evidence Collection Engine.
5. The Audit Manager sends each selected security metric information to the Evidence Collection Engine to initiate the respective event captors.
6. Event Captors Orchestrator receives the incoming request from the audit component and automated generates the respectively Event Captor.
7. ELK stack is continuously augmented by the log shippers fetching events and data from the ICT infrastructure.

8. Event Captor queries the ELK stack with a query explicitly generated for the audited security metric.
9. When a change to the security metric is detected, the Event captor's log shippers push the information to the RabbitMQ[4] Message Broker that allows communication between the audit and event collection engine component.
10. Message Broker pushes the event captor findings to the audit component.
11. The Audit Component decides whether the security metric is violated or fulfilled.
12. In both latter cases, the Audit Component sends the auditing results to the certification component, which generates the respective report (certifiable view of the metric)
13. The certification component informs the IRO regarding the results of the certifiable view of the metric.

4.3 Data models

4.3.1 Evidence Collection Engine data model

As stated before, the Evidence Collection Engine for real-time, continuous assessment of the security posture of the complex ICT systems will be enabled using Elasticsearch. In this context, the Elastic Common Schema (ECS)[5] will be used as an open-source specification for defining a common set of fields among the data model used when storing event data in Elasticsearch as logs. Furthermore, ECS will be used to specify the field names and data types to be stored while supporting uniform data modeling, enabling data analysis from diverse sources.

The main categories of ECS that are used are the core and extended categories. The former is referring to the most common across all use cases data types. Any field that is not defined as a core field belongs to the second category of extended fields.

At a high level, ECS provides fields to automated classify events and data in two different ways: “Where the data is coming from” and “What data it is”. ECS defines four categorization fields for this purpose, each of which falls under the event.* fieldset which is the following:

- **event.kind.** This is the highest level in the ECS category hierarchy. It provides high-level information about what type of information the event contains without being specific to the contents of the event
- **event.category** is the second level in the ECS category hierarchy. It represents the "big buckets" of ECS categories.
- **event.type** is the third level in the ECS category hierarchy, and it represents a categorization "sub-bucket" that, when used along with the event.category field values enable filtering events down to a level appropriate for a single visualization.
- **event.outcome** is the fourth and last level in the ECS category hierarchy that simply denotes whether the event represents a success or a failure from the perspective of the entity that produced the event.

4.3.2 Audit data model

The rules and metrics that need to be audited during the FISHY project by the audit component will be specified within security and dependability (S&D) Patterns [2] using an XML based language called

[4] https://www.rabbitmq.com/
[5] https://www.elastic.co/guide/en/ecs/current/index.html

EC-Assertion. Based on event calculus [3], EC Assertion is a first-order temporal logic language primarily created to point and represent actions and their respectively results over time. The two basic components of Event Calculus are events and fluents. An event in EC is specified as something that occurs at a specific time and its duration last promptly. This event can cause changes in the status of other states or variables. The fluents represent this state. The EC assertion may express rules that will be evaluated by the audit component in terms of violation and compliance.

To represent the occurrence of an event, EC uses the predicate Happens(*event*, *time*, $\mathfrak{R}$(t1,t2)), which represents the incident of an *event* that happens at some *time* point within the time frame *(t1, t2).* For example let us imagine an admin user that logins to a web service and thus this can be represented as *Happens(admin_login_to_web_service, time,* $\mathfrak{R}$*(t1,t2))* where *t1 ≤ time ≤ t2.* EC may use the *Initiates(event, fluent, time)* predication in order to signify that a *fluent* starts to hold after the *event* occurs at *time*. Furthermore, the EC may uses the *Terminates(event, fluent, time)* predication in order to signify that a *fluent* stops to *hold* after the *event* occurs at *time*. Furthermore, EC formula has the Initially(*fluent*) and HoldsAt(*fluent*, *time*) predications. Initially(*fluent*) predication signals that *fluent* holds at the start of an operation while HoldsAt(*fluent*, *time*) signals that *fluent* holds at time t.

Events in EC-Assertion that represents calls of systems operations and their responses or message exchange between different system components are also supported in EC-Assertion. In order to do that the latter adopts a specific structure for the events that is represented by the event term *event(_id, _sender, _receiver, _status, _sig, _source)*

In this event term:

- _*id* is a unique identifier that represents the event.
- _*sender* is the identifier of the component that sends the message.
- _*receiver* is the identifier of the component that receives the message.
- _*status* is the processing status of an event. It may be a request (REQ) or a response (RES).
- _*sig* is the signature of the dispatched message.
- _*source* is the identifier of the component where the event happened.

4.4 SACM Component: Security Metrics

4.4.1 Architecture, Functionality, interactions

The Security Metrics component is a submodule that communicates with a respective database that contains the description of several prebuilt metrics in XML format. Triggered by the administrator from the IRO, the component pushes the selected latter metrics towards the audit-monitor component. These metrics are currently focusing on the CIA principles; however, several other metrics will be tailored to be adequate to the scenarios approached by the project. The XML files of the security metrics contain information regarding the type of the metric, the period of the evaluation, the type of the asset that the security metric is referred to among other internal information necessary that the audit component needs to operate properly.

4.4.2 Black-box modelling and API

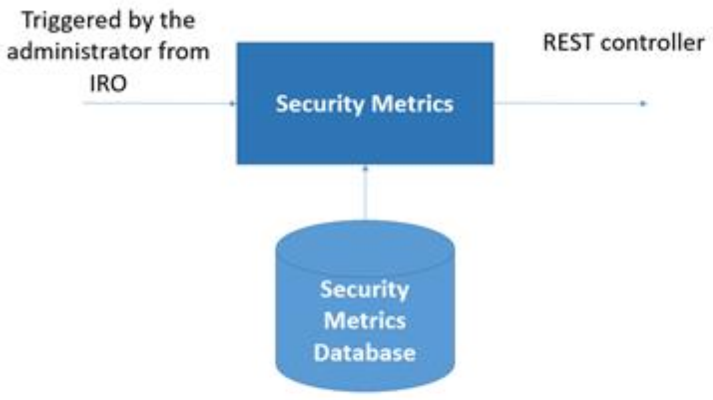

Triggered by the administrator from IRO
Security Metrics
REST controller
Security Metrics Database

Figure 11. The Black Box view of the Security Metrics component.

¡Error! No se encuentra el origen de la referencia. shows the black-box model of the Security M etrics component. Currently, it is triggered by the administrator through the IRO; therefore, no API is considered. The selected Security Metrics are forwarded to the Audit component through an appropriate REST controller which wraps and exposes the fuctionalitties of the latter component throught the respectively API.

4.5 SACM Component: Audit

4.5.1 Architecture, Functionality, interactions

The **Audit** component is a monitor module that will be responsible for initiating, coordinating, and reporting the monitoring process results. Audit is a runtime monitoring engine built in Java that offers an API for establishing monitoring rules to be checked. The module is made of two basic submodules: the main auditor and an audit manager.

The main auditor is responsible for initiating, coordinating, and reporting the monitoring process results, and it is based on a monitoring framework called EVEREST (EVEnt REaSoning Toolkit). When an S&D pattern is activated, it undertakes responsibility for checking conditions regarding the runtime operation of the components that implement the pattern. These conditions are specified within S&D Patterns by monitoring rules expressed in Event Calculus Assertion. EVEREST can detect violations of monitoring rules against streams of runtime events, which are sent to it by different and distributed event sources, through the Event Evidence Engine. It also has the capability to:

(i) deduce information about the state of the system being monitored by using assumptions about the behaviour of a system and how runtime events may affect its state. This assumptions are actually EC formulas that determines how events affect the status of system that is monitored.

(ii) detect potential violations of monitoring rules by estimating belief measures in the potential of occurrence of such violations, and

(iii) perform diagnostic analysis to identify whether the events causing a violation are genuine or the result of a system fault or an attack.

The audit manager submodule interacts with the REST controller component that initiates and provides monitoring assessments. It is implemented in the Docker[6] environment, which offers flexibility, portability and parallel execution capabilities.

Furthermore, the Audit component includes an Audit database in its internal architecture (currently, MongoDB seems the best candidate), which holds all the important attributes to conclude the auditing assessment capability. Additionally, Audit includes a message broker (RabbitHQ is the candidate tool), which interacts with the Evidence Collection Engine through an Event Chanel and handles the events received from the Audit Manager component. Finally, the Audit component may communicate with an external database that holds the security assurance model and its components.

4.5.2 Black-box modelling and API

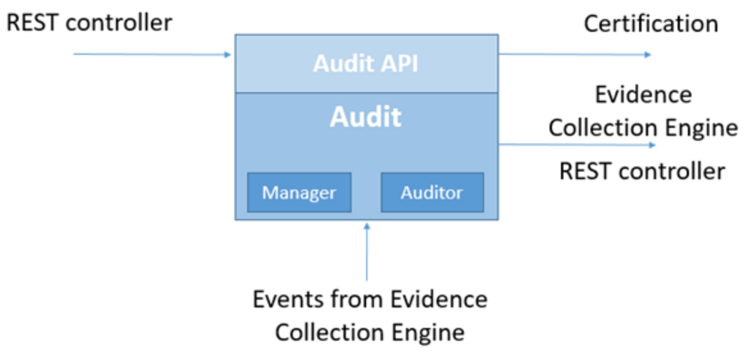


Figure 12. The Black-Box view of the Audit component.

Figure 12 shows the black-box model of the Audit component. After the initial analysis of the component features, we determined that the API will include the following methods:

Method	Description
void sendEvents(String eventsFile,Long groupID)	Declaring the channel and the queue that the events will be accessible to the Event Captors. Parameters: • eventsPath: The description of where the events are. • groupID: The unique id of the assessment. Returns: Void
void initiateEvents(String eventsPath,Long groupID)	pushes the events to RabbitMQ from the Audit Manager. Parameters: • eventsPath: The description of where the events are. • groupID: The unique id of the assessment.

[6] https://www.docker.com/

	Returns: Void
void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body)	Handles the Results that come to the RabbitMQ component from the Event Captors and lets a consume port open for receiving findings and format them to the desiring form to be consumed gracefully. Parameters: • consumerTag: Consumer that is permitted to consume from that chanel. • envelope: A way to compress the information to proceed to the sending of results. • Properties: Protocol that will be used. • Body: Results message body that can be consumed Returns: Void

4.6 SACM Component: Certification

4.6.1 Architecture, Functionality, interactions

The purpose of the **Certification** component, is to provide an evidence-based security reporting and certification to the needs of different stakeholders ranging from senior management to external auditors and regulators, incorporating different access level to the respectively users. The latter component supports the creation of specialized reports based on the findings of the Audit component while it may inform the IRO component for the former results.

Certification process checks for compliance of technical and organisational requirements of the auditee (organisation trying to achieve compliance to some standard). Since FISHY does not focus on a specific standard but rather tries to build a continuous process of compliance checks of technical requirements (and even these specific ones) of a generic security standard similar to EUCS[7]. The output from this process can be taken into account within (internal, external) audit processes of the organisation. The results can also help with the decisions w.r.t. additional controls being applied to the target infrastructure (e.g. in the IRO of FISHY). In this process each Audit Metric Instance (see **¡Error! No se encuentra el origen de la referencia.**) is equipped with an additional information (attributes and values) whether the measurement passes or fails expected value of the metric. These values are provided by the configuration process (through set of rules) of the certification component (its API). Evaluation of certification compliance means avaluation of the rules (organised as a tree) of values of the audited metrics. At the point of writing methodology of the evaluation is still not well defined, but can have different strategies of evaluation: asset-based evaluation or metric-based evaluation. In the former measured metrics would be aggregated and evaluated per asset. In the latter strategy the evaluation would first be done by “fixing” a metric and evaluate values per assets. Distinction is given in **¡Error! No se encuentra el origen de la referencia.**.

[7] https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme

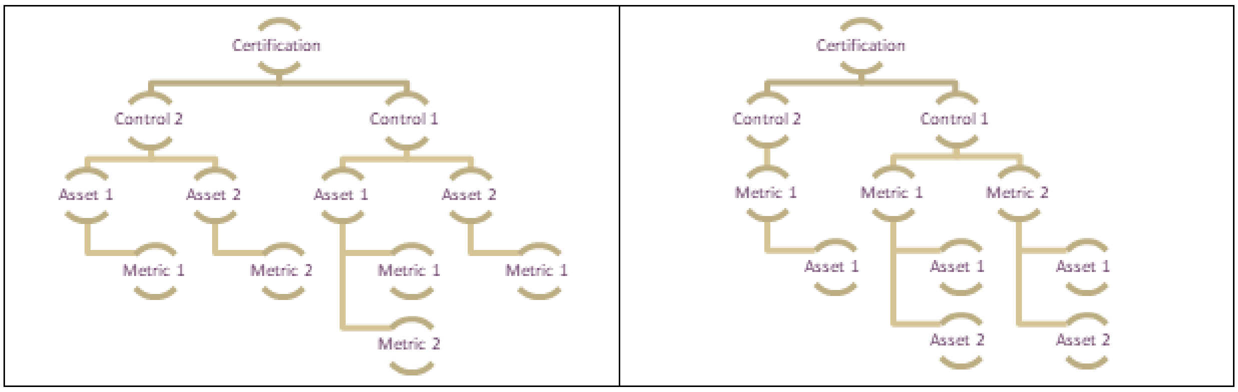


Figure 13: Asset-based and metric-based evaluation of the compliance.

Based on the chosen certification methodology evaluation of the controls of the certification migh vary and might impact the final evaluated level of the certification compliance. This process of equipting alerts/metric evaluation with additional attributes would be similar to a process done by Wazuh[8] tool, also being considered in FISHY as a tool for gathering security metric in the context of Trust and Incident Management component (TIM). In this process "Compliance" attributes are added to the report from a tool (in JSON format, relevant sections are marked with grey color):

```
{  "_index": "wazuh-alerts-4.x-2021.05.20",
  "_type": "_doc",
...
    "input": {
      "type": "log"
    },
    "agent": {
      "ip": "10.0.2.15",
      "name": "agent2",
      "id": "001"
    },
...
 "data": {
     "sca": {
       "scan_id": "577929282",
       "check": {
         "result": "failed",
         "remediation": "Use your package manager to update all packages on the system according to site policy. The following command will
install all available packages # yum update  ",
         "previous_result": "Not applicable",
         "compliance": {
           "pci_dss": "5.2",
           "hipaa": "164.312.b",
           "tsc": "A1.2",
           "cis_csc": "3.4,3.5",
           "gdpr_IV": "35.7.d",
           "cis": "1.9",
           "nist_800_53": "AU.6,SI.4",
```

[8] https://wazuh.com

```
          "gpg_13": "4.2"
        },
        "description": "Periodically patches are released for included software either due to security flaws or to include additional functionality.",
        "id": "6049",
        "title": "Ensure updates, patches, and additional security software are installed",
        "rationale": "Newer patches may contain security enhancements that would not be available through the latest full update. As a result, it is recommended that the latest software patches be used to take advantage of the latest functionality. As with any
software installation, organizations need to determine if a given update meets their requirements and verify the compatibility and supportability of any additional software against the update revision that is selected.",
        "command": [
          "yum check-update"
        ]
      },
      "type": "check",
      "policy": "CIS Benchmark for CentOS 7"
    }
  },
  "rule": {
    "mail": false,
    "level": 9,
    "pci_dss": [
      "2.2",
      "5.2"
    ],
    "tsc": [
      "CC7.1",
      "CC7.2",
      "A1.2"
    ],
    "hipaa": [
      "164.312.b"
    ],
    "gdpr_IV": [
      "35.7.d"
    ],
    "description": "CIS Benchmark for CentOS 7: Ensure updates, patches, and additional security software are installed: Status changed from 'not applicable' to failed",
    "groups": [
      "sca"
    ],
    "cis": [
      "1.9"
    ],
    "nist_800_53": [
      "CM.1",
      "AU.6",
      "SI.4"
    ],
    "gdpr": [
      "IV_35.7.d"
    ],
    "firedtimes": 2,
    "cis_csc": [
      "3.4",
      "3.5"
    ],
    "id": "19014",
    "gpg_13": [
      "4.2"
    ]
  },
...
 ]
}
```

4.6.2 Black box modelling and API

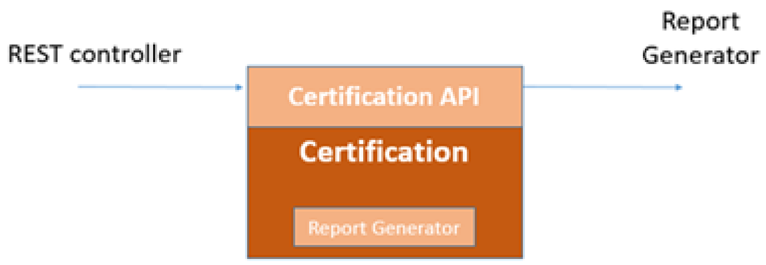


Figure 14. The Black-Box view of the Certification.

Figure 14 shows the black-box model of the Certification component. It is triggered by the results and finding of the audit component through the appropriate REST controller, and therefore, the respective API is considered. Currently, the component includes creating reports that include the evaluation of the selected security metrics; however, no additional API is being considered for this purpose. The report will consist of the evaluation of the complete compliance rule-set based on the provided evaluation of metrics' values.

4.7 SACM Component: Evidence Collection Engine

4.7.1 Architecture, Functionality, interactions

The Evidence Collection Engine component will be used for real-time, continuous assessment of the security posture of the complex ICT systems as it will aggregate in real-time, cross-layer evidence pertinent to the security posture of the ICT infrastructure. This module will use incoming data from Event Captors, a software module that, based on collected data and triggering events, formulates a rule or a set of rules and pushes the latter towards the audit component for evaluation.

The Evidence Collection Engine is developed around Elastisearch[9]. In the current implementation, data and events are collected through several lightweight shippers named Beats[10] (e.g., Filebeat, MetricBeat, PacketBeat) that centralize log data and forward them to Elastisearch. Event Captors are activated when the appropriate event happens. They query Elasticsearch, evaluate the results and push back the relevant information to the Audit component for further evaluation.

In the future development of this component, we will add Logstash[11] in the chain between Beats and Elastisearch. Logstash is an open server-side data processing pipeline that ingests and process data from a multitude of sources. It will have the role of an aggregator/augmenter of data before these are sent to ElasticSearch.

Currently, Event Captors are aligned to facilitate the security metrics provided by the respective component of SACM and with respect to the CIA venerable model. However, these captors will be augmented with several others in order to provide a more user case/pilot oriented approach.

Event captor’s tool is initiated through respectively REST calls from the Audit module, while it communicates with the Elasticsearch via the respective API.

4.7.2 Black-box modelling and API

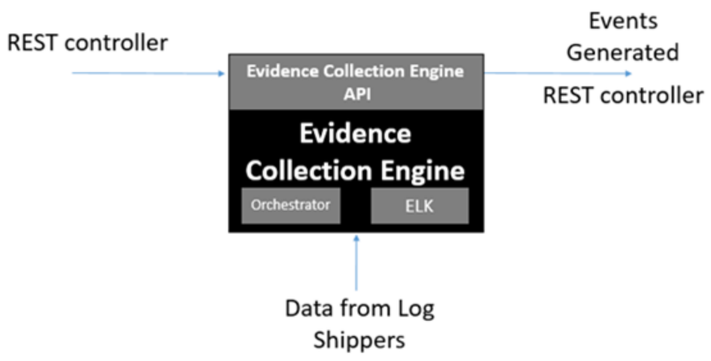


Figure 15. The Black-Box view of the Evidence Collection Engine

Figure 15 shows the black-box model of the Evidence Collection Engine component. After the initial analysis of the component features, we determined that the API will include the following methods:

Method	Description
void getEvents(String eventsFile,Long groupID)	Declaring the channel and the queue that the Evidence Collection Engine is listening from the Audit Component. Parameters: • eventsPath: The description of where the events are. • groupID: The unique id of the assessment. Returns: Void

[9] https://www.elastic.co/
[10] https://www.elastic.co/beats/
[11] https://www.elastic.co/logstash

void generatedEvents(String eventsPath,Long groupID)	Pushes the events to RabbitMQ from the Evidence Collection Engine. Parameters: • eventsPath: The description of where the events are. • groupID: The unique id of the assessment.

5 Conclusions

This deliverable has presented the main WP4 concepts, the initial design of the EDC and SACM, and the data models that are needed to ensure their correct functioning. It also provided hints for further implementation and the integration activities that will start with T4.3. These activities are part of the first iteration (IT-1) of the WP4 component development.

During the next months, EDC components will be implemented, as soon as research results will be available and use case definition will progress. Moreover, the data models will be completed and validated. SACM components, whose first implementation is available, will be integrated to complete IT-1.

During the section iteration (IT-2), the initial design of the EDC and SACM components will be enhanced, and the enhancements will be implemented. The integration will concern both WP4 components and components from other WPs. The definition of the use cases will help concentrate the effort on a more focused subset of concepts that could be adequately addressed during the project lifetime.

The results in this deliverable will be used by:

- T4.1 will use this deliverable as a starting point for the design of the final version of the EDC.
- T4.2 will use this deliverable as a starting point for the design of the final version of the SACM.
- T4.3 requires these initial models to fire the implementation of the components and the finalization of the data models. Moreover, all the interactions highlighted here will be used to identify the integration points and anticipate issues.
- WP5 will jointly work to design the high-level policies and will collaborate on the definition of the security capability model, as the capabilities will also describe, in the next iteration, the features offered by the NSF to deploy configurations and restart.
- WP3 will also use the results here to plan how to interact with the EDC (e.g., tailoring the HLP and asking for ad hoc methods to the Controller).

This deliverable allows to achieve the milestone M4.1(MS16), which has been split in individual points:

- M4.1 FISHY Sec.&Cert. block components ready for integration (IT-1) WP4 M9 D4.1
- M4.1.a.1 FISHY SACM design ready WP4 M9 D4.1
- M4.1.a.2 FISHY SACM block components ready for integration (IT-1) WP4 M9 D4.1

6 References

[1] Gamma, E. Editor (2010), *Design patterns (38th edition)*, Addison-Wesley professional computing series. Boston.

[2] Maña A et al (2006) Security engineering for ambient intelligence: A manifesto. In: Integrating Security and Software Engineering: Advances and Future Vision. Idea Group Publishing, 244–270

[3] Shanahan M.P. (1999) The event calculus explained. In: Artificial Intelligence Today. Volume 1600 of Lecture Notes in Artificial Intelligence. (1999) 409–430