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

D5.2 IT-2 FISHY release integrated

Document Identification			
Status	Final	**Due Date**	30/04/2023
Version	1.0	**Submission Date**	28/04/2023

Related WP	WP5	**Document Reference**	D5.2
Related Deliverable(s)	D5.1	**Dissemination Level (*)**	PU
Lead Participant	TID	**Lead Author**	Jose Manuel Manjón
Contributors	TID, TUBS, UPC, UC3M, ATOS, UMinho, STS, SYN, XLAB	**Reviewers**	OPT/Entersoft (Antonis Gonos)
			ATOS (Antonio Álvarez)

Keywords:

Integration, Framework, Tool

Document Information

List of Contributors	
Name	Partner
Jose Manuel Manjón Cáliz	TID
Dulce Artalejo Sacristán	UC3M
Raúl Martín Celaya	UC3M
Borja Nogales Dorado	UC3M
Francisco Valera Pintor	UC3M
Iván Vidal Fernández	UC3M
Eva Marín Tordera	UPC
Jan Antić	XLAB
Hrvoje Ratkajec	XLAB
Mounir Bensalem	TUBS
André Oliveira	UMinho
Henrique Santos	UMinho
Pedro Magalhães	UMinho
Grigoris Kalogiannis	STS
Jorge Martínez Olmo	ATOS
Guillermo Yuste	ATOS
Alexandra Lakka	SYN

Document History			
Version	Date	Change editors	Changes
0.1	15/02/2023	TID	ToC
0.2	15/03/2023	TID	ToC update
	28/03/2023	TID	Contribution to sections 1 and 5
	29/03/2023	UC3M	Contribution to sections 4 and 5
	30/03/2023	UPC	Contribution to sections 3 and 5
	31/03/2023	XLAB	Contribution to section 5
	03/04/2023	TID	Contribution to section 4
0.3	04/04/2023	TUBS	Contribution to sections 2 and 5
	04/04/2023	UMinho	Contribution to section 5

	10/04/2023	STS	Contribution to section 5
0.4	13/04/2023	ATOS	Contribution to section 5
0.5	14/04/2023	TID	Small modifications on the whole document
		UC3M	
		UPC	
0.6	21/04/2023	TID	Final version to be reviewed
0.7	25/04/2023	ATOS	Reviewed version
0.8	25/04/2023	OPT	Reviewed version
0.9	28/04/2023	TID	Final version for Quality Check
0.9a	28/04/2023	ATOS	Quality Assessment
1.0	28/04/2023	ATOS	Final version submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Jose Manuel Manjón (TID)	28/04/2023
Quality manager	Juan Andrés Alonso (ATOS)	28/04/2023
Project Coordinator	Antonio Alvarez (ATOS)	28/04/2023

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AI	Artificial Intelligence
API	Application Programming Interface
CDN	Content Delivery Network
CEF	Common Event Format
CRUD	Create, Read, Update and Delete
CSS	Cascading Style Sheets
CCIPS	Centrally Controlled IPSec
D5.2	Deliverable number 2 belonging to WP5
E2E	End-to-end
EDC	Enforcement and Dynamic Configuration
F2F	Farm-to-Fork
FCS	FISHY Control Services
FRF	FISHY Reference Framework
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HPL	High-level Policy Language
IDCO	Inter-Domain Connectivity Orchestrator
IKE	Internet Key Exchange
IPSec	IP Security
IRO	Intent-based Resilience Orchestrator
JSON	JavaScript Object Notation
K8s	Kubernetes
L2S-M	Link-Layer Secure connectivity for Microservice platforms
MANO	Management and Orchestration
MON	Monitoring
NBI	NorthBound Interface
NED	Network Edge Device
NFV	Network Function Virtualization

Abbreviation / acronym	Description
NFVI	Network Functions Virtualization Infrastructure
NLP	Natural Language Processing
NS	Network Service
NSF	Network Security Functions
OF	Orchestration Function
ONOS	Open Network Operating System
OSM	Open-Source MANO
OvS	Open Virtual Switch
PMEM	Predictive MaintenancE Monitoring
PoC	Proof of Concept
PromQL	Prometheus Query Language
RAE	Risk Assessment Engine
ReM	Remediation Module
REST	REpresentational State Transfer
SACM	Security Assurance & Certification Management
SADE	Securing Autonomous Driving Function at the Edge
SBI	SouthBound Interface
SeCM	Security Capability Model
SIA	Secure Infrastructure Abstraction
SPI	Security & Privacy Data Space Infrastructure
SPI-DM	Security & Privacy Data Space Infrastructure – Data Management
SPI-IDM	Security & Privacy Data Space Infrastructure – Identity Management
SSL	Secure Sockets Layer
SSO	Single Sign-On
TCP	Transmission Control Protocol
TIM	Trust & Incident Manager
Tx.y	Task number y belonging to WP x
VAT	Vulnerability Assessment Tool
VM	Virtual Machine
VNF	Virtual Network Functions
VPN	Virtual Private Network
VxLAN	Virtual Extensible LAN

Abbreviation / acronym	Description
WBPTV	Wood-based Panels Trusted Value-Chain
WP	Work Package
XL-SIEM	Cross-Layer Security Information and Event Management
XML	eXtensible Markup Language
YANG	Yet Another Next Generation

1 Introduction

1.1 Purpose of the document

This deliverable reports the status of the FISHY Reference Framework (FRF) and the tools deployed on it, from the point of view of the implementation. Also, describes the modules that are part of this Work Package 5 itself, the Intent-based Resilient Orchestrator (IRO), the Dashboard and the Secure Infrastructure Abstraction (SIA).

All the software developments can be found in a GitHub repository created specifically for this project: https://github.com/H2020-FISHY.

1.2 Relation to another project work

This document collects all the developments of the Work Packages of the project, specifically WP3 and WP4. All the tools developed previously are implemented on the FRF as final stage of the project and will be integrated to work among them. Also, in contradistinction to D5.1, this deliverable shows a mature and evolved framework, with an intense work on developments from tools providers.

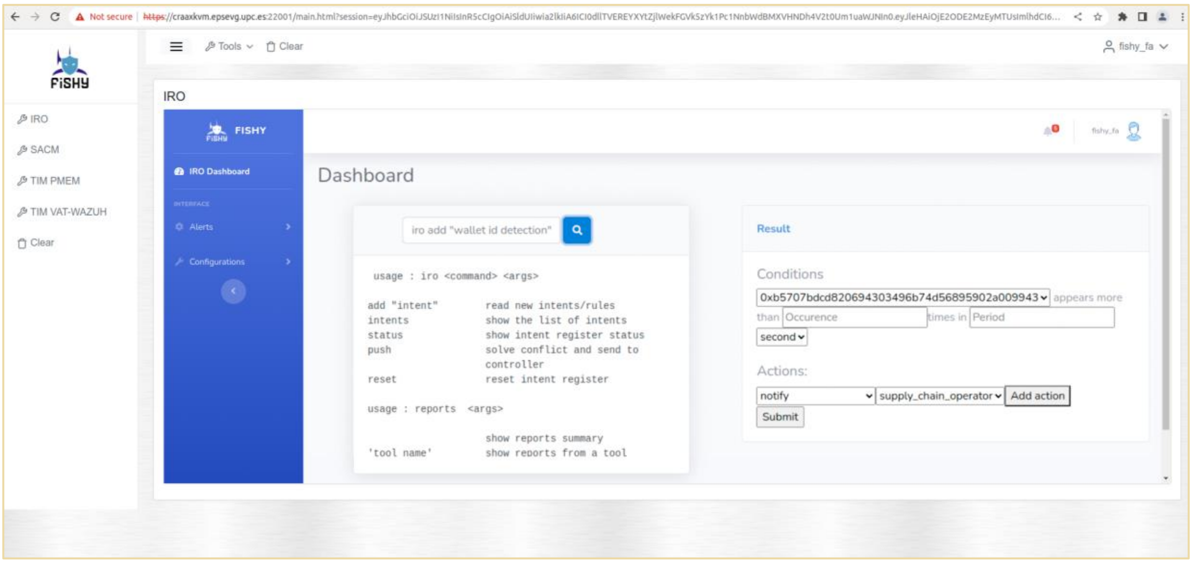
1.3 Structure of the document

This deliverable is divided into 4 main sections:

- **Section 2** resumes the work done on the Intent-based Resilient Orchestrator (IRO), referred to task 5.1.
- **Section 3** reports the status of the Dashboard, related to task 5.2.
- **Section 4** updates the work done on the Secure Infrastructure Abstraction (SIA), also referred to task 5.2.
- **Section 5** presents the status of the FISHY Reference Framework and the integration procedure of the different tools that will comprise the FRF.

2 Intent-based Resilience Orchestrator

The Intent-based Resilience Orchestrator (IRO) aims at automating the interactions between the user defining high level intents and the system applying high level policies, using natural language processing (NLP) and AI techniques. Furthermore, the IRO is also designed to assure the notification of the users with the events and reports received from different monitoring tools of TIM component, as well as Smart Contracts verifications. The front end of the Dashboard for the FISHY user is shown in Figure 1.



iro add "wallet id detection"

```
 usage : iro <command> <args>

add "intent"        read new intents/rules
intents             show the list of intents
status              show intent register status
push                solve conflict and send to
                    controller
reset               reset intent register

usage : reports  <args>

                    show reports summary
'tool name'         show reports from a tool
```

Conditions

0xb5707bdcd820694303496b74d56895902a009943 appears more than Occurence times in Period second

Actions:

notify supply_chain_operator Add action

Submit

Figure 1. IRO Dashboard

2.1 IRO architecture and workflow

IRO is composed by several blocks, which are the IRO-Dashboard, the intent manager, the policy configurator, the Learning & Reasoning component and the knowledge base. IRO-Dashboard is a python-based web application that enables several features for the user such as visualizing reports and security event presentation from different monitoring tools in a single location/screen, allowing the system administrator to define actions in the form input intents that can be translated into network configurations with the help of lower-level FISHY components such as the EDC for enforcement and the Central Repository for data sharing. The intent manager is the IRO module that translates the input text in the intent into a structured format containing several fields such as the intent type and parameters. In the current version, the intent translation is using predefined queries for Elasticsearch engine in order to match in the input text to a predefined ontology. Several queries have been defined and can be enriched in the future to allow the translation of complex input text. An example of the queries used is a search of a word or a part of word, and a search of an attribute related to word. Those queries can be used to identify and match the input text to a set of predefined intent structures, which then can be used to recommend a form of a policy to be filled by the user through the Dashboard. The intent manager is also responsible for providing instructions to the user, for listing the intent templates, and it can validate policy generation after receiving confirmation from the user.

Another important module in IRO is the policy configurator, which receives a formatted intent translated by the intent manager, and (with the help of the knowledge base) is used by the policy configurator to generate high-level security policies (XML schema for EDC) and attacks information (a JSON file used by EDC to enforce policies). The knowledge base is a module that uses Elasticsearch to store the intent definition and policy templates and manages the CRUD operations. The Learning and Reasoning module in the current version are running two RabbitMQ consumers: one for Central Repository events, and another for the Smart Contracts verification. The received events are cashed in the system to be used for user notification and alerts. This module can extract information from events and filter them based on the source, type of event, and the defined intents.

In this diagram (Figure 2), we illustrate the component level diagram of the IRO, considering the inter-component communication and the communication between IRO and other FISHY components.

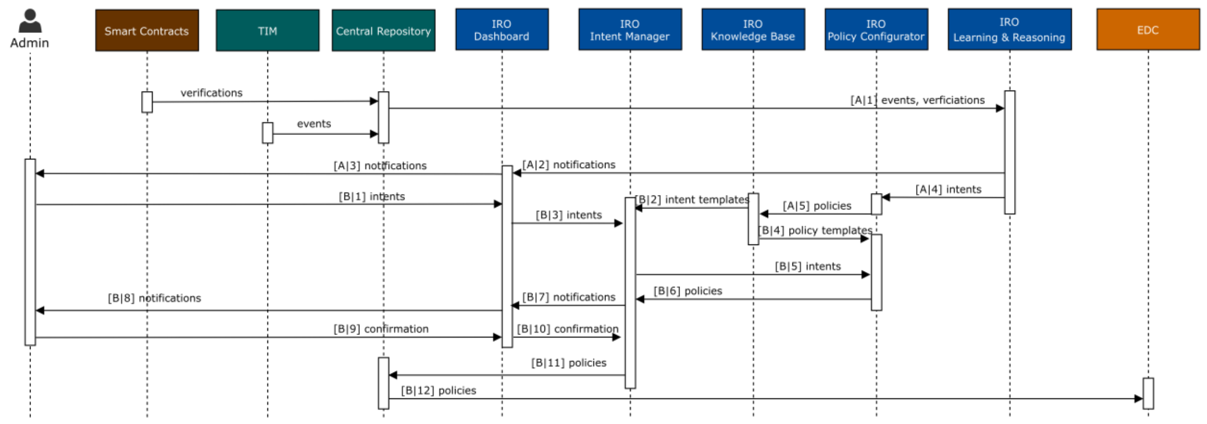


Figure 2. IRO workflow

The workflow can be described as follows:

- [A|1] IRO receives the events stored in the Central Repository by different TIM tools, and verification events posted by Smart Contracts component.
- [A|2] The received events are processed by the Learning & Reasoning components and a notification is created and presented in the IRO Dashboard.
- [A|3] The administrator or the FISHY user can access the interface and receive notifications through an alert center.
- [A|4] The events received from the central repository, which need to trigger a decision-making process, trigger the IRO Policy Configurator using a predefined intent associated to events from the Learning & Reasoning component.
- [A|5] The configured policies are stored in the Knowledge Base temporally to be checked for validity and confirmation.
- [B|1] The admin sends intents through the Dashboard.
- [B|2] The Intent Manager reads intent templates, and policy templates from the Knowledge Base.
- [B|3] The Intent Manager translates the received intents using existing intent templates.
- [B|4] The Intent Configurator reads policy templates from the Knowledge Base.
- [B|5] The Policy Configurator receives formatted intents from the Intent Manager.
- [B|6] The Intent Manager receives configured policies from the Configurator, solves conflicts with other policies and checks if a user intervention is needed.
- [B|7], [B|8] After receiving policies, the Intent Manager notifies the admin through the IRO Dashboard using an alert center developed in the frontend interface.
- [B|9] The admin, who has an overview of the current reports received from different tools, confirms actions through the IRO Dashboard, with the help of a predefined list of actions.

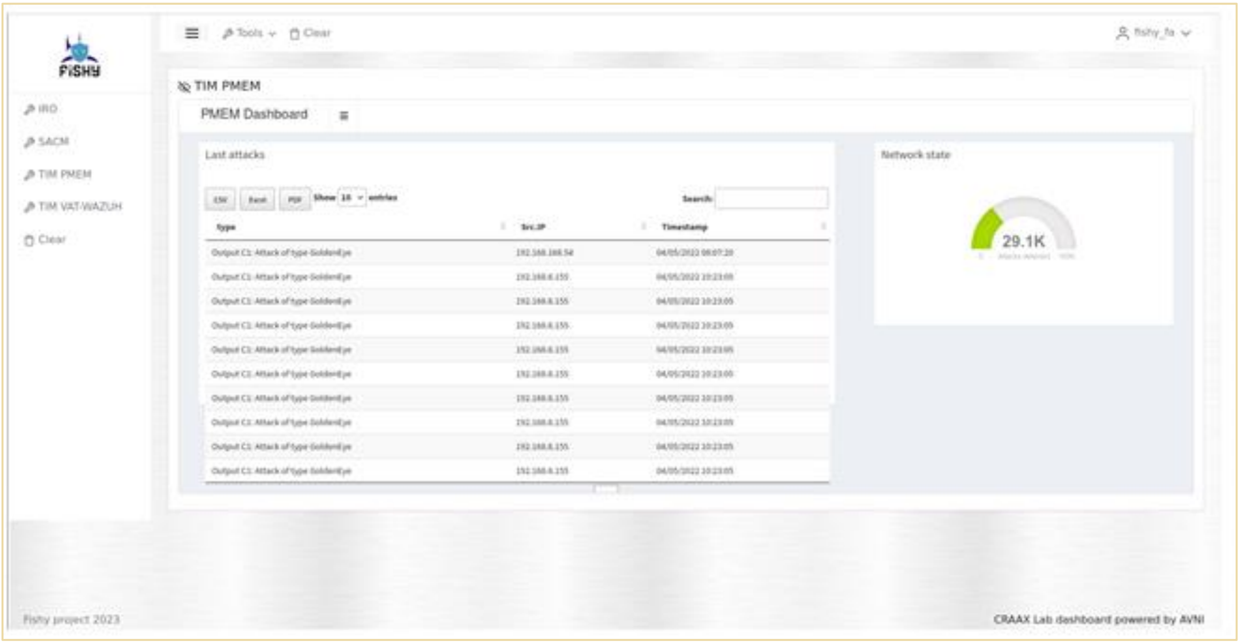
- [B|10], [B|11] The Intent Manager posts the confirmed policies in the Central Repository under a policy endpoint.
- [B|12] Central Repository sends events about policies to the EDC through the RabbitMQ mechanism.

Table 1. Communication interfaces of IRO

Component	Origin	Data	Type of communication	Status
Central Repository	TIM tools SACM	Reports in CEF format	REST HTTP	Implemented
Central Repository	TIM tools SACM	Reports in CEF format	RabbitMQ	Implemented
	Smart Contracts	Smart Contracts validation in CEF format	RabbitMQ	Implemented

3 Dashboard

The main objective of the FISHY dashboard is to provide a unified, harmonised and consistent web application where users of the FISHY framework are able to perform the different set of actions for which they are responsible for. The dashboard is basically a Node.js based web application that provides a graphical user interface (GUI) for visualizing and managing data from multiple tools in a centralized space. It provides a way for the FISHY users to interact with all the tools and provides a unified overview to manage the front-end of the different tools. The FISHY dashboard can be deployed either in cloud resources made available by the FISHY platform provider or at the client's premises (at client's owned cloud resources). This allows FISHY to provide flexibility of the FISHY as the dashboard can be deployed in a centralized manner or on client's premises depending on the end user requests. During this last year of the project, there are two versions of the dashboard working: one instance is responsible to manage the two FISHY use cases, F2F and WBP TRUST, while another instance is working separately managing one FISHY use case, SADE. The front-end of the dashboard for the FISHY user is shown in Figure 3.



Type	Src.IP	Timestamp
Output C1: Attack of type GoldenEye	192.168.188.54	04/05/2022 08:07:20
Output C1: Attack of type GoldenEye	192.168.8.155	04/05/2022 10:23:05
Output C1: Attack of type GoldenEye	192.168.8.155	04/05/2022 10:23:05
Output C1: Attack of type GoldenEye	192.168.8.155	04/05/2022 10:23:05
Output C1: Attack of type GoldenEye	192.168.8.155	04/05/2022 10:23:05
Output C1: Attack of type GoldenEye	192.168.8.155	04/05/2022 10:23:05
Output C1: Attack of type GoldenEye	192.168.8.155	04/05/2022 10:23:05
Output C1: Attack of type GoldenEye	192.168.8.155	04/05/2022 10:23:05
Output C1: Attack of type GoldenEye	192.168.8.155	04/05/2022 10:23:05
Output C1: Attack of type GoldenEye	192.168.8.155	04/05/2022 10:23:05

Figure 3. FISHY Dashboard

3.1 Dashboard Functionalities

Different functionalities provided by the FISHY dashboard are discussed in this section.

Dashboard access

The gate to the FISHY dashboard is via a web browser accessible through any platform. Accessing the dashboard requires the user to log-in using the provided credentials. Once the user is logged, the user is redirected to the homepage of the application, and certainly the system does not close the connection until either a logout or session timeout occurs.

Single Sign-On

One of the key features provided by the FISHY dashboard is the support of a single sign-on (SSO) mechanism for different users. SSO allows users to log in once and access multiple applications without having to enter their credentials again. The credentials entered by the user on FISHY dashboard are verified using the centralized Keycloak server provided by an SPI module (SPI identity management). After the verification of the credentials, an access token is generated which is used throughout the session to authenticate the users within the different tools.

Multitenancy:

The multitenancy in the FISHY dashboard is achieved with the help of Keycloak and user scope. The scope and access of the users to different tools is managed internally with the help of the Keycloak. Multitenancy in a dashboard is the ability to serve multiple tenants, which are typically different organizations or users. This means that a dashboard can be used by multiple tenants, each with their own data, user accounts, and settings, while still maintaining a shared infrastructure and code base. The dashboard achieved multitenancy with the help of the Keycloak server. Users' authentication and authorization is done using the Keycloak server while customization of the access to different tools is managed internally by the dashboard.

Style harmonization and Usability improvement:

The whole design is unified and improved to be more usable and harmonized, hence transmitting to the user the feeling that is a single application providing with different tools, instead rather than of a group of diverse tools accessible from a website.

Dashboard interfaces

All the tools with a GUI are integrated in the dashboard by means of an iframe. On the other hand, FISHY dashboard communicates with SPI identity management for requesting token and verifying user credentials, using a communication HTTPS.

Dashboard Sections

The FISHY dashboard is designed to show different views for the different users. It provides a way to connect to different FISHY tool GUIs by relying on iframes. All the tool GUIs are integrated in the dashboard using the iframe. The view for the user administrator for each one of the different use cases is shown in the Table 2, showing different FISHY tools depending on the use case for IT-2.

Table 2. Tools in each use case for IT-2

Tool name	F2F	WPB	SADE
IRO	YES	YES	YES
PMEM	YES	NO	NO
XL-SIEM	NO	YES	YES
RAE	NO	YES	YES
VAT	YES	NO	NO
WAZUH	YES	NO	NO
Trust Monitor	NO	NO	YES
SACM	YES	YES	YES
EDC	YES	YES	YES

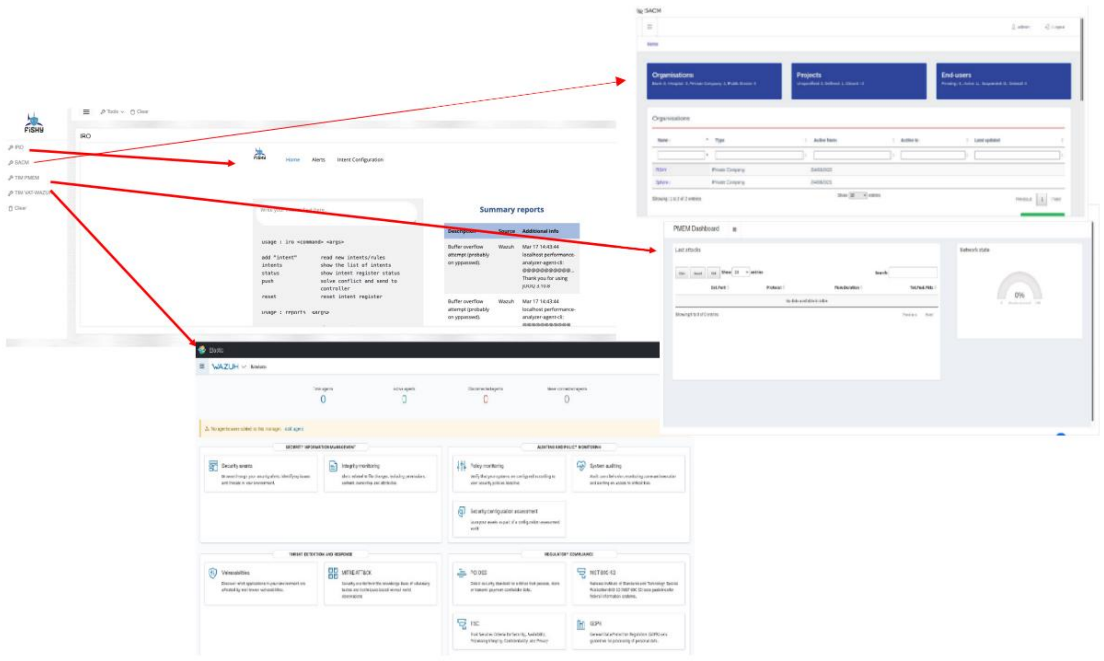


Figure 4. FISHY Dashboard view of F2F use case

Some of the GUI’s tools integrated in the Farm to Fork (F2F) use case are shown in Figure 4. In the same way the different use cases are able to see different views in the FISHY dashboard as defined by their scope.

3.2 Dashboard’s Internal Architecture

In this subsection, we detail the internal architecture of the FISHY dashboard. In a high-level view, the dashboard frontend is developed using HTML, CSS and JavaScript. The backend is built with Node.js and Express[1]. To start the dashboard, you have to run the main index.js file using the node. The dashboard is also available in the dockerized form and you can also run it with the help of Docker. The final version of the FISHY dashboard is installed in the FISHY reference framework (FRF) and the YAML configuration to deploy it in the FRF are also developed.

The frontend manages the login strategy with the help of the Keycloak server. The centralized Keycloak server is used to login users with the username and password, thus avoiding the need for the dashboard to store users and/or passwords. In this way, the dashboard doesn't have to deal with login forms, authenticating users, and storing users.

The Express web server is used to serve the frontend on top of the Node server. This configuration of servers’ stack has the advantage of providing a high level of scalability when having different simultaneous connections.

In the login phase, the user provides its username and password to get access to the frontend as well as to the different tools and services. The front-end, then, forwards this information to the Keycloak server responsible for its verification. If the information about username and password is valid the server returns a token to the user which will be used by the rest of the tools for the verification of the user.

[1] https://expressjs.com/

4 Secure Infrastructure Abstraction

This section describes the implementation details corresponding to the **Secure infrastructure Abstraction** (SIA) module. The initial design of the SIA has been presented in deliverable D2.2 [1]. In the following, we review the architectural design of this module, describing the refinements that have been specified as a result of the research and implementation work conducted during the iteration 2 (IT-2) of the project. Figure 5 outlines the SIA architectural design.

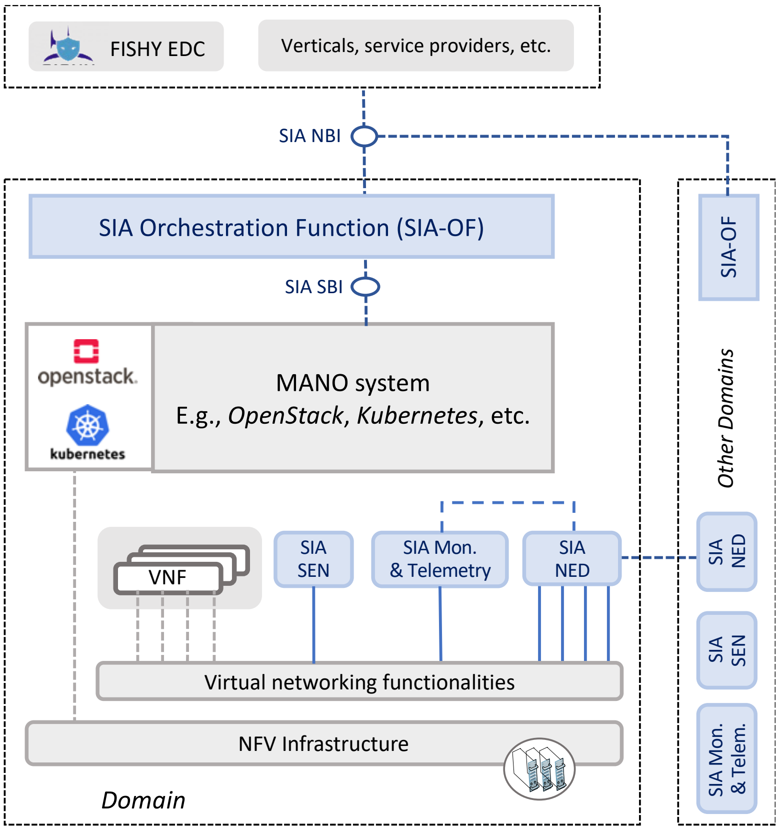


Figure 5. Overview of the SIA architecture

The Secure Infrastructure Abstraction (SIA) is responsible for the provisioning of a data-plane interface to support external and inter-domain communications within the FISHY platform (e.g., between an IoT/edge infrastructure and a cloud infrastructure, or between multiple cloud infrastructures). In addition, it controls the network access to the FISHY domains, protecting data traffic entering and leaving the domains. This functionality is mainly provided by the SIA Network Edge Device (**NED**) component of the SIA which is further described in section 4.3. The SIA also includes a specific component for monitoring and telemetry information collection (SIA Monitor, **MON**) associated with the NED operations. The MON component has been developed within the framework of WP3 and is here described on section 4.4.

According to the FISHY approach, organizations are structured into different *realms***,** based on the cybersecurity constraints, policies or rules, and realms are divided into *domains*, where a domain is defined as a *group of assets with certain relationships (same network, infrastructure, location, etc.)* [1]. The SIA operates at a domain level providing the proper means to interact with the NFV infrastructure resources that are available at every domain, regardless of the particular technologies that are used (OpenStack [2], Kubernetes [3], etc.). This functionality is provided by the SIA

Northbound interface (**NBI**) and an Orchestration Function (**OF**). The OF is deployed at every domain, whereas the SIA NBI is a centralized component that can be used by other modules of the FISHY platform, like the Enforcement and Dynamic Configuration (EDC). Both components are described in section 4.1. To support a proper interaction with any specific management and orchestration software stacks that exist in a domain, the SIA includes an adaptable southbound interface (**SBI**), which is covered in section 4.2.

Another tool that is part of the SIA is the Centrally Controlled IPSec (CCIPS). The CCIPS goes beyond the classical point-to-point IPsec setup and provides a centralized architectural solution to control multiple IPsec endpoints or gateways. This solution is composed of a centralized E2E manager (controller) and two or more agents, based on IPsec engine in IKE-less mode (no IKE protocol is needed).

4.1 SIA Northbound interface (NBI) and Orchestration Function (OF)

The SIA NBI provides the point-of-access to interact with the NFVI resources that are available at every domain. This point-of-access is offered to other FISHY blocks and components, such as the EDC. To support this functionality, the SIA NBI interfaces with the Orchestration Function (OF) available at every domain.

Conceptually, the SIA NBI can support different functionalities across domains, which involves: (1) the management of NFV descriptors (*e.g.,* upload/delete/update the data that describes the network services and VNFs that are to be deployed); (2) the lifecycle management of network services and VNFs, including Network Security Functions (NSFs) [4] as a particular case; (3) the collection of performance information related to the execution of network service; (4) issuing notifications under fault conditions; and (5) to provide information on the capacity of NFV infrastructure resources.

As for the design criteria agreed upon in WP5, the SIA NBI is aligned with the Application Programming Interface (API) specification defined by ETSI for their NFV orchestrator, which is included in ETSI NFV-SOL 005 [5]. This enables the SIA NBI to be consistent with standard specifications. In this regard, the NFV descriptors are based on the YANG models specified in ETSI NFV-SOL 006 [6] to ensure the interoperability and compatibility with other NFV solutions. Table 3 summarizes the formal definition of the SIA interface offered through the SIA NBI.

Focusing on the implementation aspects, and following the design criteria outlined above, the SIA NBI is based on HAProxy [7]. HAProxy is an open-source load-balancing software commonly used in web application architectures and content delivery networks (CDNs). It operates as a reverse proxy, receiving requests and distributing them to different backend servers according to established load-balancing rules. In addition to its main function as a load balancer, HAProxy also offers other useful features, such as the ability to protect against Distributed Denial-of-Service (DDoS) attacks and compatibility with different network protocols such as HTTP, TCP, and SSL.

On the other hand, the OF component is based on Open Source MANO, or OSM [8]. OSM is an ETSI-hosted project that provides a Management and Orchestration (MANO) software stack aligned with the ETSI NFV specifications. A noteworthy aspect is that OSM exposes an API based on ETSI NFV-SOL 005. Under the context of the project, this allows the SIA NBI to properly distribute the requests to the correspondent API of each OF available at every domain, and being compliant with the standard specification.

Table 3. Formal definition of the SIA interface (inbound)

Component	Origin	Data	Type of communication	Status
SIA	Other FISHY modules (EDC); NFV stakeholder, such as 5G/6G vertical and service providers.	Standard data models defined by ETSI [5][6]	RESTful protocols specification for the ETSI MANO Os-Ma-nfvo Reference Point [5]	Implemented through HAProxy and OSM

4.2 SIA Southbound interface (SBI)

As previously commented, the SIA must provide its functionalities regardless of the technologies that are used at every domain (e.g., OpenStack or Kubernetes). To this purpose, the module includes an adaptable southbound interface (SBI), supporting the interaction with different NFV management and orchestration (MANO) technologies.

In this regard, the software that provides the basis for the SIA OF, ETSI OSM, already provides support for OpenStack infrastructures, since it is an already well-established solution for NFV technologies. In consequence, no modifications are required in those domains that implement that virtual infrastructure management solution. However, new platforms based on container technologies like Kubernetes (K8s) are starting to become an enticing prospect for the deployment of VNFs in cloud and edge environments, since containers provide a light-weight solution for the development and deployment of applications and Network Services (NS). K8s popularity can be seen across both industry and academia due to its high adoption rate for the deployment of applications and services.

However, OSM has limited support for the deployment of network services in K8s clusters since it only allows the deployment of VNFs as regular K8s pods. Due to the nature of the FISHY modules and functionalities that need to be deployed in the project, it is necessary to have a solution inside the cluster that enables the creation and management of virtual links and networks that can securely interconnect the VNFs of a network service, and isolate data traffic transmitted on these virtual links and networks.

In this regard, the L2S-M K8s [9] operator provides the necessary tools to create isolated link-layer virtual networks inside K8s clusters. Therefore, the SIA SBI in K8s clusters is implemented using a combination of OSM and L2S-M to fully enable its proper functionality, providing the flexibility to interact with any management and orchestration tools in a particular domain. L2S-M has been released as an open-source project. The available documentation can be found in the project website [10].

Currently, the deployment of NSs in K8s cluster can be performed with the combination of both L2S-M and OSM by using Helm charts [11] that can be specified as part of NFV descriptors, which can be deployed later in a K8s cluster with L2S-M installed (since it provides the appropriate tools to create the necessary virtual networks to connect different VNFs). This behavior has been tested in a Proof of Concept (PoC) [12] to leverage the potential of OSM for virtual networking in K8s environments.

With the knowledge that the PoC provided for the deployment of NSs in K8s clusters using OSM, and thanks to the high interest from the community, a new feature has been approved to enable the introduction of virtual networking capabilities in the OSM code [13] (i.e., provide this behavior natively instead of manually configuring the virtual network charts beforehand). This feature is currently under the design phase.

4.3 SIA Network Edge Device (NED) overlay

The SIA architecture includes the ability to create and delete virtual link-layer networks that connect VNFs running in different domains, independently of the specific management and orchestration software stacks employed in those domains. This way, it supports link-layer inter-domain communications among remote VNFs. This inter-domain connectivity system is enabled by SDN technologies and comprises two main elements: the Network Edge Device (NED) overlay network and the Inter-Domain Connectivity Orchestrator (IDCO).

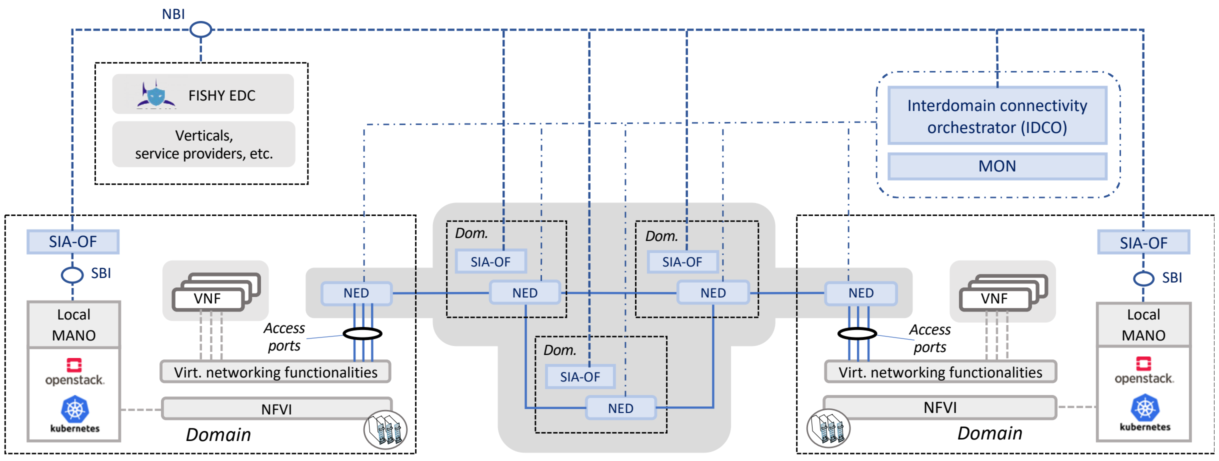

NBI
FISHY EDC
Verticals, service providers, etc.
Interdomain connectivity orchestrator (IDCO)
MON
SIA-OF
SBI
Local MANO
openstack
kubernetes
VNF
Access ports
Virt. networking functionalities
NFVI
Domain
NED
Dom.
SIA-OF
NED

Figure 6. Overview of the NED overlay

NEDs are programmable switching functions, implemented using Open Virtual Switches (OvS) [14]. They forward traffic between domains. Each domain containing VNFs that require connectivity with other VNFs in different domains must have at least one NED. The NEDs are connected between them through point-to-point protected IP tunnels (e.g., IPSec VXLAN tunnels), thereby creating a NED overlay network that interconnects all the FISHY domains. The topology of the NED overlay is established manually in our implementation.

The IDCO functions as an SDN controller, implemented as an internal application that runs within an instance of the Open Network Operating System (ONOS) [15]. All the NEDs connect to the IDCO via the OpenFlow 1.3 protocol [16]. The IDCO is accessed through the SIA NBI using a custom HTTP REST API inspired in the ETSI GS NFV-IFA 032 ([17]) MSCS Management Interface that allows creating and deleting inter-domain virtual networks.

Figure 6 illustrates the SIA architecture, considering a scenario where multiple domains exist, each containing a SIA OF and a NED component. In such scenario, when two or more VNFs located in different domains require inter-domain connectivity, each of them is connected to an access port of its domain NED through an intra-domain NED. Simultaneously, an inter-domain virtual network connecting those NED ports is created by communicating it to the IDCO through the SIA NBI. Subsequently, the IDCO establishes tunnel-id-based tunnels point-to-point or multi-point virtual circuits on top of the NED overlay, through which the inter-domain communications for those VNFs are delivered. This is achieved using the Virtual Network Identifier (VNI) in the case of Virtual eXtensible Local Area Network (VXLAN). In our implementation, the virtual circuits created in this manner follow the least-cost path between domains considering the hop count as the cost metric.

4.4 SIA Monitor (MON)

The MON is a SIA component that is able to monitor different network parameters of multi-domain NFV environments. Its design and development was originally introduced in the context of WP3, as it focused on monitoring the integration, resource consumption, and communication of tools developed in that work package and it is now integrated in the Fishy Reference Framework (see section 5) within the framework of WP5 to monitor the whole platform.

The implementation of the MON component is based on Prometheus [18], an open-source monitoring system that is primarily used for monitoring and alerting on the performance of various systems and services. It uses a time-series database to store the collected metrics and the Prometheus Query Language (PromQL) to retrieve and process the data. The system is capable of retrieving metrics from a variety of sources, including applications, servers, and other devices, and it can also trigger alerts when certain conditions are met. This makes it a powerful and flexible monitoring system that is widely used in cloud and container environments, making it ideal for FISHY Kubernetes-based integration.

However, in order to display all the desired metrics and dashboards and in order to make it useful to monitor different domains, it has been necessary to include a more powerful visualization tool.

Grafana has been selected [19], as it is a powerful and popular data visualization and monitoring platform that is fully compatible with Prometheus data sources. Grafana allows to create –and if desired, to share– interactive and informative dashboards, alerts and notifications to monitor their systems and services. Besides, it allows the addition of multiple data sources, making it optimum to monitor the different domains that can make up a multi-domain NFV environment each one with their own Prometheus deployment.

4.5 SIA Centrally Controlled IPSec (CCIPS)

As described at the beginning of section 4, the CCIPS is composed by a controller and two or more agents, deployed where the IPsec tunnel is established. In this IKE-less case, the RFC specifies a procedure on the re-keying process that is handled by the controller, when requested by the nodes.

On one side, the CCIPS controller architecture relies on a REST API as the central component to provide the NBI and establish sessions with the agents using the NETCONF protocol. On the other side, the CCIPS agents must provide an endpoint for the NETCONF protocol and manage the YANG models. To do this, the agent uses: a) sysrepo library [20], which is a datastore used to store configurations based on different YANG models and it provides a set of C bindings; b) Netopeer2 library [21], a NETCONF server that exposes the endpoint to manage the sysrepo datastore.

The process of configuring the IPSec tunnel starts with the controller, that gets a request (from the OAM) and generates the necessary configuration materials that will be sent to the agents later, so that they store the information. These requirements that arrives to the controller includes information about IP addresses of the agents (it can distinguish between management and data networks), the type of algorithm to be used and the lifetimes for the re-key process.

5 FISHY Reference Framework

The FISHY Reference Framework (FRF) is a testbed environment able to integrate and support the execution of all the FISHY components and the possible relevant functions that are necessary for the functionality of the FISHY project. In other words, the FRF holds the integrated status of the whole FISHY platform, including all its components and other infrastructures (internal or external). This testbed can be used to showcase the FISHY functionality and to implement the use cases defined for the project.

The FRF, thanks to its flexible design, can incorporate components and infrastructures that are not directly located in the same premises where the FRF is hosted. By using the SIA module, in charge of the communications between FISHY components, other external infrastructures can be added into the FRF, if there exist IP level communications with one, or several, NEDs inside the FRF.

The main architecture of the FRF can be seen in Figure 7, which depicts the main components of the FRF and their connectivity. As it can be seen in the figure, the undelaying FRF infrastructure hosts all the FRF components. These components are composed of Virtual Machines (VMs) managed inside an OpenStack infrastructure. This infrastructure is in the 5TONIC laboratory premises in Madrid [22].

The FRF, following the architectural design of the FISHY project, is divided into multiple domains, each one presented as a Kubernetes (K8s) [3] cluster (which can be composed of multiple VMs, physical devices, etc…) or an OpenStack [2] environment. Each one of these domains host different FISHY modules and functionalities, deployed as K8s pods (i.e., containers) or Virtual Machines respectively. To enable the secure communications between components within a domain, each one of them uses the SIA's South Bound Interface to create isolated link-layer networks, used by the modules to securely interact with the modules that need a point-to-point (or multipoint) connectivity with other modules/functionalities. This SBI is implemented in a different way depending on the characteristics of each domain: for OpenStack environments, Open Source MANO (OSM) [8] directly supports this creation; for K8s platforms, it is necessary to combine the use of OSM and the L2S-M [9] K8s operator.

Since some of the modules deployed in the FRF might be located in different domains, a NED component is deployed per domain to oversee the secure inter-domain communication between each component. In this regard, different NEDs are connected using IP tunneling mechanisms to build an overlay, which follows a set topology according to the necessities of the platform/project (using as example the overlay seen in the figure). Since each of the functions are deployed as pods and/or VMs, each platform can enable external communications using their own networking mechanisms (for example, services in K8s clusters). These mechanisms are reserved for specific purposes for each module (e.g., GUI display), but data exchange between domains will always be performed using the NED overlay.

The initial design of the FRF includes the definition of three domains hosted in the 5TONIC premises (all deployed as K8s clusters):

- FISHY Control Services (FCS): This domain is responsible for hosting all the functionalities related with the control and management of all the FISHY modules spread in the rest of the FRF.
- FISHY Domain 1 & FISHY Domain 2: Provide a K8s abstraction for the deployment of FISHY modules.

The FRF design enables the dynamic incorporation of external domains and infrastructures, thanks to the flexibility that the overlay of NEDs provides for the communication of modules and components in the FRF. However, since the FRF is located inside the 5TONIC premises, access from the outside is limited (since it is hosted inside its private network) to preserve secure communications within the FRF. To enable external components access, 5TONIC uses a VPN server that enables any VPN client (with the appropriate credentials) to connect with the infrastructure. Any external domain will need to install a VPN client with the credentials to enable this communication. Once this step is performed, any external domain can deploy a NED in its premises and attach it to the overlay (using an IP tunnel), effectively enabling the inter-domain communications between components. This VPN can also be used to access the exposed services/modules from the outside (for example a GUI).

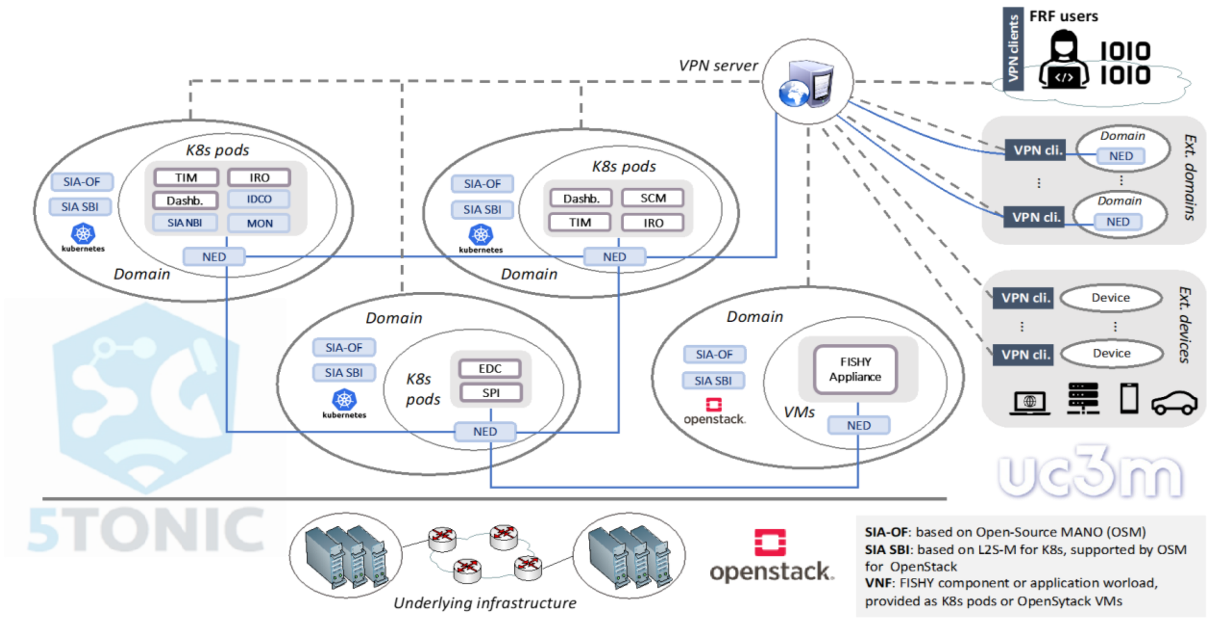

VPN server
VPN clients
FRF users
1010
1010
K8s pods
SIA-OF
SIA SBI
TIM
IRO
Dashb.
IDCO
SIA NBI
MON
kubernetes
NED
Domain
SCM
VPN cli.
Ext. domains
Device
Ext. devices
EDC
SPI
FISHY Appliance
openstack
VMs
5TONIC
uc3m
Underlying infrastructure
SIA-OF: based on Open-Source MANO (OSM)
SIA SBI: based on L2S-M for K8s, supported by OSM for OpenStack
VNF: FISHY component or application worload, provided as K8s pods or OpenSytack VMs

Figure 7. FISHY Reference Framework diagram

To build and update the FRF during the lifetime of the project, each partner can interact with the corresponding domain to deploy their corresponding modules and attach them into their respective virtual networks to enable the connectivity with the remaining FRF elements. To keep track of the integration process of each component, the project uses **Error! Reference source not found.**. This table details the main characteristics of the modules that are integrated inside the FRF, including their location (i.e., the domain where they are being deployed), hardware requirements, components that each module interacts with and if external connectivity is required. This information must be filled in before the integration starts, since it is used for accommodating the resources of a domain (increase its resources if necessary) and to establish the virtual networks that are going to be used by each module.

Currently, there are four virtual networks defined in the FRF:

- Network net-dashboard: This network is used for communicating all the components that need to send data to the dashboard (deployed in the FCS).
- Network net-spi: If a component needs to send/retrieve data to/from the SPI module, it will be attached to this network.
- Network net-central-repo: Modules that write or read from the central repository module located in the FCS domain) will use this network for their communication.
- Network ned-rabbitmq: If a module needs to pick up events from the RabbitMQ module located in the FCS, it will be retrieved using this network.

The list of virtual networks created in the FRF along with their current IP addressing for each module can be seen in Table 5.

Table 4. FRF Integration status table

Component	Deployment Status	Description	Module	Organization	Location	vCPUs	RAM (GB)	Storage Disk (GB)	Correspondent components	External Communications	Additional Requirements
Name of the component	*Status*	*Brief description of component deployment status (to be updated periodically)*	*Name of the module to which the component corresponds*	*Organization responsible for component deployment*	*Specify where to deploy each component: is it a centralized component (FCS), or is it a component that operates at a domain level?*	*Required resources in terms of computation, memory, and storage (take into account that, given that 5TONIC is a shared experimentation environment, components with high resource demands may be subject to a reduction on the resource allocation according to their actual resource utilization, and may eventually and transitorily be stopped to conduct other experiments on 5TONIC facilities)*			*Other components with which the component needs to directly communicate. Are these components in the same domain or in a different one (i.e., are they intra or inter-domain communications?)*	*Specify whether or not the component requires external communications, that is, communications with entities outside the domains, e.g., a GUI component (provide more details in the next column, if needed)*	*Include any additional requirements needed in the deployment of the component, or any additional remarks related with the request*
SIA-NED	Deployed	A functional NED is deployed at every domain	SIA	UC3M	FCS	N/A	N/A	N/A	N/A	no	
SIA-NED	Deployed	A functional NED is deployed at every domain	SIA	UC3M	domain#1	N/A	N/A	N/A	N/A	no	
SIA-NED	Deployed	A functional NED is deployed at every domain	SIA	UC3M	domain#2	N/A	N/A	N/A	N/A	no	
L2S-M	Deployed	L2S-M provides the SIA SBI for K8s. It is installed at every domain	SIA	UC3M	FCS	N/A	N/A	N/A	N/A	no	
L2S-M	Deployed	L2S-M provides the SIA SBI for K8s. It is installed at every domain	SIA	UC3M	domain#1	N/A	N/A	N/A	N/A	no	
L2S-M	Deployed	L2S-M provides the SIA SBI for K8s. It is installed at every domain	SIA	UC3M	domain#2	N/A	N/A	N/A	N/A	no	
SIA-OF	Deployed	Based in OSM.	SIA	UC3M	FCS	N/A	N/A	N/A	N/A	no	
SIA-NBI	Deployed	SIA NBI deployed in the FCS domain. Based on NFV SOL-005.	SIA	UC3M	FCS	N/A	N/A	N/A	N/A	yes	External port: 30050
SIA-IDCO	Deployed	IDCO has been integrated in the cluster, with its central controller located in the FCS.	SIA	UC3M	FCS	N/A	N/A	N/A	N/A	no	
SIA-MON	Deployed	Deployed. Currently managing all the resources of the FISHY Domains.	SIA	UC3M	FCS	1	2	25	N/A	no	External Port: 30001, 30090 (GUI), 30002 (NodePort)
XL-SIEM	InProcess	VPN Credentials and K8s credentials sent by UC3M. Waiting for deployment of the component in the FRF.	TIM	ATOS	FCS	8	8	30	SPI,RAE,FISHY Dasboard (intra-domain),	yes	External ports: 8080 (GUI), 41000 (Events)
RAE	Deployed	VPN Credentials and K8s credentials sent by UC3M. Waiting for deployment of the component in the FRF.	TIM	ATOS	FCS	2	8	60	SPI,XL-SIEM,FISHY Dasboard (intra-domain),	yes	External ports: 9000 (GUI)
SPI-IDM	Deployed	Deployed. Can be configured inside console GUI (Available @ https://10.4.34.249:30443)	SPI	UMinho	domain#1	2	4	8	SPI DM, Fishy Aplliance agents, Data collectors (intra-domain)	yes	Node Port: 30080, 30443 (Console GUI) Private port: 8080, 8443
SPI-DM	Deployed	Missing the Central repository output endpoint. All input endpoints are integrated	SPI	UMinho	domain#1	2	2	4	SPI IDM, Fishy Aplliance agents (intra-domain), Central Respository(inter-domain)	no	API Port: 5000
IRO	Deployed	Component installed in the FRF.	IRO	TUBS	FCS	2	2	2	Central repository, Dashboard	no	NodePort: 30500 (5000)
Central repository	InProcess	VPN Credentials and K8s credentials sent by UC3M. Waiting for the deployment of the component in the FRF. YAML for k8s deployment already made for sandbox, will be modified for FRF.	TIM	XLAB	FCS	1	1	10	IRO, TIM tools	no	HW requirements are minimal, API port: 10010, also requires RabbitMQ
PMEM	InProcess	Integration proccess started	TIM	UPC	FCS	1	1	20	SPI Data Management(Inter Domain), SPI Identity Manager, Central Repository and Fishy Dashboard(intra domain)	no	
Fishy Dashboard	Deployed	Component deployed in the FRF. Available https://10.4.34.136:30756	Fishy Dashboard	UPC	FCS	2	4	16	SPI Identity Manager, TIM tools GUI, SACM GUI, IRO GUI	yes	NodePort:22001 (Made Public to be accessd by users)
SPI-IDM	Deployed	Deployed. Available @ https://10.4.34.136:30443	SPI	UMinho	FCS	2	4	8	FISHY Dasboard, Central Respository, SPI DM, TIM Tools (intra domain)	yes	Node Port: 30080, 30443 (Console GUI) Private port: 8080, 8443
SPI-DM	Deployed	Missing the Central repository output endpoint. All input endpoints are integrated	SPI	UMinho	FCS	2	2	4	Central Respository, SPI IDM, TIM Tools (intra-domain)	no	API Port: 5000
SIA-CCIPS	InProcess	New version ready. Docker controller image sent.	SIA	TID	FCS	2	2	10	N/A	no	
SIA-CCIPS	InProcess	New version ready. Docker agent image sent.	SIA	TID	domain#1	2	2	10	N/A	no	
SIA-CCIPS	InProcess	New version ready. Docker agent image sent.	SIA	TID	domain#2	2	2	10	N/A	no	
SACM	InProcess	VPN Credentials and K8s credentials sent by UC3M. Waiting for the deployment of the component in the FRF.	TIM	Sphynx	FCS	8	8	30	SPI DM, Fishy Aplliance agents, Data collectors (intra-domain),FISHY Dasboard, Central Respository	yes	
Smart Contracts	InProcess	VPN Credentials and K8s credentials sent by UC3M. Waiting for the deployment of the component in the FRF.	TIM	Synelixis	FCS	4	8	60	The components can interface with the component through the RabbitMQ of the Central Repository	no	

Table 5. Virtual networks table in FRF

TO BE FILLED BY UC3M					
Component	**Network net-dashboard (192.168.0.0/24)**	**Network net-spi (192.168.1.0/24)**	**Network net-central-repo (192.168.2.0/24)**	**Network net-rabbitmq (192.168.3.0/24)**	**…**
Name of the component	*Network created to enabling communications with the Dashboard*	*Network created to enabling communications with the SPI module*	*Network created to enabling communications with the Central Repository module*	*Network created to retrieving events from the RabbitMQ server*	*Information about the IP addresses to be configured on each component according to the network where it is connected (to be completed by uc3m)*
SIA-NED	192.168.0.1	192.168.1.1	192.168.2.1	192.168.3.1	
SIA-NED					
SIA-NED					
L2S-M					
L2S-M					
L2S-M					
SIA-OF					
SIA-NBI					
SIA-IDCO					
SIA-MON					
XL-SIEM	192.168.0.2	192.168.1.2			
RAE	192.168.0.3	192.168.1.3			
SPI-IDM		192.168.1.4			
SPI-DM		192.168.1.5	192.168.2.5		
IRO	192.168.0.6		192.168.2.6	192.168.3.6	
Central Repository			192.168.2.7	192.168.3.7	
PMEM					
Fishy Dashboard	192.168.0.9	192.168.1.9			
SPI-IDM	192.168.0.10	192.168.1.10	192.168.2.10		
SPI-DM		192.168.1.11	192.168.2.11		
	192.168.0.12	192.168.1.12	192.1268.2.12		

5.1 SIA integration

In the following, we detail a list of the SIA components. For each of these components, we describe the decisions taken regarding their integration into the FISHY FRF, along with their current integration status (the resources allocated to each of the SIA components, in terms of virtual CPUs, memory, and storage, which are collected in Table 4):

- **SIA NED**: The SIA NED component has been installed in each of the three domains of the FRF (it will also be installed in any additional domains that might be needed, including any external sites requiring connectivity to the FRF). Based on Open V-Switch (OVS) [14] technology, this component has been deployed as a K8s pod in those domains that have been instantiated as K8s clusters. Similarly, it has been deployed as a virtual machine in OpenStack based domains. Even though only one NED is necessary on each domain to enable inter-domain connectivity, more than one NED can be made available in a domain for the sake of redundant and performant operation.
- **SIA NBI**: The deployment of the SIA NBI component has been carried out on the Kubernetes cluster available in the FCS domain, and it is based on a Helm chart [11]. In particular, in the open source HAProxy chart available in the Bitnami repository [23]. This chart is then configured at deployment time with the appropriate values to serve as an access point both to the different FISHY components, and to every function available at each domain, enabling the management of each NFV infrastructure resources. In terms of computational resources, since we have opted for a lightweight NBI SIA implementation, no specific values were set to avoid allocating a greater number of resources than needed. Nevertheless, the monitoring service offered by the MON component indicates that SIA NBI utilizes less than 1 vCPU on average, and an instant memory does not exceed 25 MB.
- **SIA OF**: With respect to the SIA OF component, its deployment has been conducted at every domain available within the FRF. As previously mentioned in section 4.1, its implementation is based on the OSM software stack provided by ETSI. Particularly, the OF utilizes the OSM Release THIRTEEN, and has been installed on each domain of the FRF as a virtual machine with Linux Ubuntu 20.04.5 LTS, with 2 vCPUs of processing, 6 GB RAM, and 60 GB of storage.
- **SIA SBI**: As commented, the SIA SBI is based on ETSI OSM, which already provides a comprehensive southbound interface for OpenStack based domains. To support K8s-based domains, the SIA SBI used L2S-M. Hence, the latter has been installed as a K8s operator in every K8s-based FISHY domain. This enables the management of virtual networks (creation/deletion) and the attachment of FISHY components to those networks, so that they can communicate on isolated network segments.
- **SIA IDCO**: The SIA IDCO component has been deployed in an independent virtual machine that belongs to the FCS domain. It runs as a Kubernetes pod in a Kubeadm [24] 1.26 cluster in the virtual machine. The latter has a Linux Ubuntu 20.04.3 LTS operating system, with 4 GB of RAM, 2 virtual CPUs and 20 GB of storage. All the existing SIA NED instances communicate with the IDCO instance.
- **SIA CCIPS** implementation is being done by providing to the different domains the role of CCIPS agents. These agents will receive the information from the CCIPS controller, which manages the secure connection by specifying some parameters, such as the type of algorithm and the lifetimes. The CCIPS controller will be deployed on the FISHY Control Services as a Kubernetes pod and the CCIPS agents, also in Kubernetes pod, will be on FISHY Control Services and FISHY domains 1 and 2.

5.1.1 Interfaces

Although the formal definition of the SIA interface is presented in section 4.1, details on this interface are summarized here for completeness and coherence of the document.

Table 6. SIA inbound interfaces.

Origin	Data	Type of communication	Status
EDC	NFV data following standard data models defined by ETSI [5][6]	RESTful protocols specification for the ETSI MANO Os-Ma-nfvo Reference Point [5]	Implementation completed using HAProxy and OSM
NFV stakeholder (e.g., 5G/6G vertical or service provider)	NFV data following standard data models defined by ETSI [5][6]	RESTful protocols specification for the ETSI MANO Os-Ma-nfvo Reference Point [5]	Implementation completed using HAProxy and OSM

Table 7. SIA outbound interfaces.

Destination	Data	Type of communication	Status
N/A	N/A	N/A	N/A

5.2 XL-SIEM integration

The XL-SIEM consists of two main components, the agent, and the server. The agent is installed in each use case infrastructure to collect information.

The server is installed on the FRF and is composed of 3 modules: topology, database, and dashboard. The topology receives information from the agent and analyzes it to create alarms. The alarms are then stored in the database, sent to the SPI (SPI translates the alarms to CEF format and sends them to the Central Repository) and shown in the dashboard.

In the following we describe the actions carried out to integrate the XL-SIEM into the FRF. *All the resources needed, and integration status are shown in Table 4.*

- **Multitenancy and SSO**

To achieve multitenancy, users have been created for each use case. We have worked to incorporate an authentication system based on Keycloak.

- **Upload images and add to the repository.**

As a first step, a docker image of every XL-SIEM component was built and uploaded to the machine in the FRF. Once the docker images are in the FRF, the related information has been added to the container registry.

- **Deployment of the XL-SIEM in the Kubernetes Cluster.**

The final deployment of the XL-SIEM components was completed on a Kubernetes cluster available in the FRF domain. For that purpose, we have defined the Kubernetes deployment file for the asset. We created a YAML configuration file that will be used to deploy it in the FRF as a Kubernetes pod.

In this file, the container images that we have previously uploaded are referenced, as well as the number of replicas and environment variables and configuration options.

- **Provide connectivity to the components.**

Finally, we have added the needed networks to the XL-SIEM. The information about such networks must be included in the YAML file.

5.2.1 Interfaces

Table 8. XL-SIEM inbound interfaces.

Origin	Data	Type of communication	Status
Data collectors	Data collected at the data collectors	No specific one. Interfaces, use SIA connectivity	Ongoing

Table 9. XL-SIEM outbound interfaces.

Destination	Data	Type of communication	Status
SPI data management	Alarms generated in native format	AMQP	Ongoing

5.3 RAE integration

Risk Assessment Engine (RAE) is a Python and R-based tool that is composed of three components:

- the Dashboard, that has the user interface communication modules and a database;
- the Engine, that has the Dexi and R model, communication modules and a database;
- the Deployment module, with the Docker configurations, that were tailored for FISHY.

Actions regarding integration into the FRF, all the resources needed, and integration status are shown in Table 4:

- Uploading images and adding them to the repository: a docker image of the Dashboard, Engine and Deployment modules were uploaded into the FRF domain and added to the container registry.
- Deployment in the Kubernetes Cluster: YAML configuration file used to deploy the images in the FRF as a Kubernetes pod.
- Providing connectivity to the components, adding networks into the YAML configuration file.
- Sending risk reports to the SPI, that translates them into the CEF format and sends them to the Central Repository.
- Integration with Keycloak: integration with FISHY Keycloak server is in progress to allow SSO and multitenancy for the three use cases.

5.3.1 Interfaces

Table 10. RAE inbound interfaces.

Origin	Data	Type of communication	Status
XL-SIEM	Alarms in native format	AMQP	Ongoing

Origin	Data	Type of communication	Status
VAT	Reports of detected vulnerabilities	AMQP	Not started, this integration was decided at a later stage, has no dependent activities and can be done quickly.

Table 11. RAE outbound interfaces.

Destination	Data	Type of communication	Status
SPI	Risk report in native format	API endpoint	Ongoing

5.4 SPI integration

The Security and Privacy Infrastructure (SPI) module establishes seamless communication with all components of the FISHY Project, thereby serving as a crucial interface between low-level constituents, higher-level modules, and users. Notably, the Identity Manager (IDM) and Data Manager (DM) modules of the SPI have been successfully deployed in the FRF using two pods.

The IDM pod, comprising of a Keycloak instance and a PostgresSQL database, is responsible for access control and authentication within FISHY. Any pod within one of the networks where the SPI-IDM pod is deployed can directly send HTTP requests to Keycloak, either to request or validate an access token.

On the other hand, the SPI-DM pod, composed of a single container featuring an HTTP Python Server, accepts requests from tools requiring data transformation into the CEF format. Subsequently, the Data Manager transforms the data and stores it in the Central Repository.

5.4.1 Interfaces

Table 12. SPI inbound interfaces.

Origin	Data	Type of communication	Status
FISHY appliance	Raw data from data collectors and processed data from Zeek tool	HTTPS (REST API) and/or Pub-sub, if required	FISHY appliance under deployment
FISHY Dashboard and tool's GUI	Client credentials/Access token	HTTPS (API)	SPI-IDM has been deployed to the FRF. The FISHY Dashboard and tool GUIs are currently using Keycloak outside the FRF, the process of integration with the version inside the FRF is ongoing.
Zeek	Alarms generated in native format	HTTPS (API)	The Zeek input endpoint has been integrated. Currently

Origin	Data	Type of communication	Status
			Zeek writes directly in the Central Repository (the process of sending it through SPI data management is ongoing)
XL-SIEM	Alarms generated in native format	HTTPS (API)	The XL-SIEM input endpoint has been integrated. Currently XL-SIEM writes directly in the Central Repository (the process of sending it through SPI data management is ongoing)
RAE	Risk report in native format	HTTPS (API)	The RAE input endpoint has been integrated. Currently RAE writes directly in the Central Repository (the process of sending it through SPI data management is ongoing)
TIM VAT (Including WAZUH)	Alarms generated in native format	HTTPS (API)	VAT and WAZUH do not require a CEF transformation endpoint. Currently VAT/WAZUH writes directly in the Central Repository (the process of sending it through SPI data management is ongoing)
PMEM	Alarms generated in native format	HTTPS (API)	The PMEM input endpoint has been integrated. Currently PMEM writes directly in the Central Repository (the process of sending it through SPI data management is ongoing)
Trust Monitor	Attestation reports on enterprise infrastructure	HTTPS (API)	The Trust Monitor input endpoint has been integrated. Currently Trust Monitor writes directly in the Central Repository (the process of sending it through SPI data management is ongoing)

Table 13. SPI outbound interfaces.

Destination	Data	Type of communication	Status
Central Repository	Processed data in tools own format in CEF format Different tool's security events in CEF Format	HTTPS (API)	Central repository hasn't been deployed to the FRF yet. However, the DM has been deployed with the correct pre-configured output

5.5 IRO integration

The integration of the IRO on the FRF as a Kubernetes pod is already achieved, where IRO software contains two containers: IRO source code and Elasticsearch. They can leverage the NED to communicate with the container hosted on the other VM of the Sandbox. The communication between IRO and other components such as TIM tools, Smart contracts and EDC can now be achieved only through the Central Repository, which has been introduced to facilitate the communication between different FISHY components. The communication with Central Repository can be achieved either using REST HTTP APIs, for basic CRUD (Create, Read, Update, Delete) operations, or using a publish/subscribe RabbitMQ message bus, to receive real-time updates from TIM tools and Smart Contract. In order to manage user information and sessions, IRO can also achieve communication with Keycloak server instance deployed inside the FRF as a pod, using REST HTTP.

5.5.1 Interfaces

Table 14. IRO inbound interfaces.

Origin	Data	Type of communication	Status
Central Repository	Reports	AMQP	Completed. IRO consumes reports from different tools.
Central Repository	Reports	REST API	Completed. IRO reads all events stored in the Central Repository.
Smart Contracts	IRO receives a verification about reports from Smart Contracts	AMPQ	Completed. IRO consumes events from Smart Contracts.

Table 15. IRO outbound interfaces.

Destination	Data	Type of communication	Status
Central Repository	IRO generates policies to be enforced by EDC and saves them in the Central Repository.	REST API	Completed. The generated policies are automatically sent to the Central Repository

5.6 Central Repository integration

In IT-1 the Central Repository, then known as the Threat/Attack Repository, was strictly a WP3 component and in IT-2 it was expanded to serve as a central repository and facilitate the communication between the components (modules) of all technical WPs (WP3 – WP5). In this sense, it was renamed to Central Repository.

The communication between modules, both internal to TIM and other platform modules, is facilitated by an HTTP(S) REST API, that allows all CRUD (Create, Read, Update, Delete) operations over the data entities handled by TIM; and a publish/subscribe RabbitMQ message bus, that allows any involved components to receive real-time updates on any relevant new or updated data.

In IT-2, its data model definitions have been expanded from only dealing with events and alerts to the ability to store various policies (high level and medium level), configuration data, and certification data and facilitate immediate responses from the platform thanks to the pub/sub system providing instant notifications. The Central Repository has also been modified to enable the storage of the TIM tool reports in the CEF format.

For the purposes of integration with the FISHY Reference Framework (FRF), the networking configuration has been expanded to use two networks, one for HTTP(S) access to Central Repository REST API and the second one for communication with the RabbitMQ. Services that only read notifications from the Central Repository need to connect to the RabbitMQ network only.

5.6.1 Interfaces

Table 16. Central Repository inbound interfaces.

Origin	Data	Type of communication	Status
SPI data management	Outcome of the TIM and SACM tools in CEF format and RAW data	REST API	Currently TIM tools and SACM write directly in the Central Repository (ongoing the process of sending it through SPI data management)
LOMOS	Processed data with anomaly score. LOMOS analyses the log data and assigns it	REST API	Integration in progress

Origin	Data	Type of communication	Status
	an anomaly score. Entries that surpass an anomaly threshold are stored in Central Repository.		
IRO	IRO generates policies to be enforced by EDC and saves them in the Central Repository.	REST API	Done. The generated policies are automatically sent to the Central Repository
SACM	Reasoning results	REST API	Implementation in progress, SACM will write its reasoning results to the Central Repository
EDC	Policies (at all the abstraction levels). Controller reads HLPs and stores MLPs. Enforcer reads MLPs and stores LLCs. ReM stores intents and HLPs.	REST API	Only Controller reads HLPs. The rest of the work is ongoing, data models on Central Repository are completed
PMEM	Detected attack: Type and time stamp	Pub-Sub mechanism (REST API)	Done with with Central repository andcurrently working on data formats
Trust Monitor	Attestation reports on enterprise infrastructure	Rabbit MQ/API	Not started

Table 17. Central Repository outbound interfaces.

Destination	Data	Type of communication	Status
Smart Contracts	Receive events/policies/alerts from FISHY components	AMQP	Smart Contracts receives all relevant data for storage on blockchain.
IRO	Reports and events from Smart Contracts	AMQP	Done. IRO consumes reports from different tools and events from Smart Contracts

Destination	Data	Type of communication	Status
SPI data management	Raw data in CEF, with pseudoanonimization, when required, and processed data from TIM tools and SACM in CEF	REST API	Process of adding the CEF data model to the central repository is ongoing. Once it's added, SPI-DM will post the normalized data
EDC	Policies (at all the abstraction levels). Controller reads HLPs and stores MLPs. Enforcer reads MLPs and stores LLCs. ReM stores intents and HLPs.	REST API	Only Controller reads HLPs, data models in Central Repository are completed.

5.7 PMEM integration

The PMEM is an R[2] based application which is used to identify the networks anomalies and attacks. The integration of this tool in the FRF is being done by first dockerizing the R application which contains both server and front end. The YAML configuration file is used to deploy it in the FISHY Control Services as a Kubernetes pod. The pod is able to interact with the networks created inside the FRF: net-dashboard, net-spi. Through these networks PMEM is able to communicate with both, the dashboard and the Keycloak server instance deployed on the FRF as a pod.

5.7.1 Interfaces

Table 18. PMEM inbound interfaces.

Origin	Data	Type of communication	Status
FISHY Appliance	Raw data	Ansible deployment	Ongoing

Table 19. PMEM outbound interfaces.

Destination	Data	Type of communication	Status
SPI data management	Detected attack: Type and time stamp	Pub-Sub mechanism (REST API)	Done with Central repository and ongoing the work of sending data to SPI data management

[2] https://www.r-project.org/

5.8 Dashboard integration

The Dashboard is a Node.js based web application which provides the SSO and multi-tenancy mechanism for users to interact with the different FISHY tools working in the different use cases. The users just have to login in the dashboard, and they are able to interact with the different tools based on their scope.

The implementation of this tool has been done by first dockerizing the Node.js application. The YAML configuration file is used to deploy it in the FRF as a Kubernetes pod. The pod is able to interact with two networks: net-dashboard, net-spi to do the communication between different tool GUIs and Keycloak server instance deployed inside the FRF as a pod.

5.8.1 Interfaces

Table 20. Dashboard inbound interfaces.

Origin	Data	Type of communication	Status
SPI identity management/access control	Access tokens & Verification	HTTPS (API)	SPI-IDM has been deployed to the FRF. The tools have integrated their user login mechanisms with Keycloak

Table 21. Dashboard outbound interfaces.

Destination	Data	Type of communication	Status
SPI identity management/access control	Verified user credentials	HTTPS (API)	SPI-IDM has been deployed to the FRF. The tools have integrated their user login mechanisms with Keycloak

5.9 SACM Integration

The SACM platform consists of a series of individual dockerized components along with a collection of tools in a modular manner offering cybersecurity related services such as security and privacy assurance of complex ICT systems. The platform supports the continuous assessment of the security and privacy posture of organizations and enterprises. To do so, the platform provides the following key capabilities among others:

- modelling and (automated) discovery of organization/enterprise IT and business assets through its asset loader component;
- continuous multi-layer runtime monitoring of threats, vulnerabilities, incidents, and security controls effectiveness based on Event Calculus and the EVEREST engine through its auditing and event collection components;

- a front-end novel GUI that allows FISHY users to create/update assets and deploy/start its monitoring capabilities.

All individual components of SACM have been deployed in the IRO component providing its own GUI access to the user platform. These components are moved to the FRF as individual Kubernetes pods while the networking configuration are expanded to facilitate HTTP(S) access to Central Repository due to the fact that the reasoning results of SACM are stored to the Central Repository.

The pod of the SACM GUI is able to interact with the networks created inside the FRF: *net-dashboard*, *net-spi*. Through these networks the pod of the SACM GUI is able to communicate with the dashboard and the *Keycloak* server instance deployed on the FRF.

5.9.1 Interfaces

Table 22. SACM inbound interfaces.

Origin	Data	Type of communication	Status
Smart Contracts	Raw data from Smart Contracts	AMQP	Started. Currently reading data from smart contracts and performing reasoning
FISHY Appliance	Raw data connected for analysis	AMQP/ElasticSearch	Interfacing of data collectors and agents is done in isolation, agent integration into the Appliance is in progress

Table 23. SACM outbound interfaces.

Destination	Data	Type of communication	Status
FISHY Dashboard	GUI of the SACM	SACM GUI will provide a URL to its main dashboard for the FISHY dashboard in terms of interaction.	GUI in the dashboard, has been deployed successfully.
Smart Contracts	Auditing Module of the SACM sends data to Smart Contracts	RabbitMQ	Ongoing
Central Repository	Reasoning results. SACM can store to the central repository its reasoning results	REST API	Ongoing

5.10 Platform Monitoring Tool

Considering all the different elements that have been developed in this project, the need to verify a correct integration takes great precedence. The MON is a particular SIA component that allows the monitoring of the different modules integrated in the FRF, as a multi-domain NFV ecosystem. With this monitoring component, it is possible to verify if the different elements have been deployed according to their specifications, if their interfaces are connected as defined and if the traffic is being sent and received as desired. In addition, it can also be used to monitor the platform operation in the different tests.

As it has been previously described, the MON uses Prometheus. The basis Prometheus has a web-based interface for visualizing the data –accessible at http://<host_IP>:30090 although not used–, and it can be integrated with other systems for more advanced dashboarding and alerting functionality. In our case, Grafana has been deployed, as it enables the integration of various data sources, making it ideal for monitoring the diverse domains comprising the FRF. With this, it becomes possible to merge panels from components located in either FISHY Control Services (FCS) domain, Domain 1 or 2 into a single dashboard, based on preference.

Grafana's GUI is accessible at http://<FCS_IP>:30001 and allows to see many different dashboards and metrics. As far as this integration goes, the focus has been placed on the main metrics to check the functionality of the different components, creating a dedicated dashboard for each FISHY tool deployed in the FRF. In the following figure the metrics displayed for the SPI-IDM component located in the FCS domain can be observed.

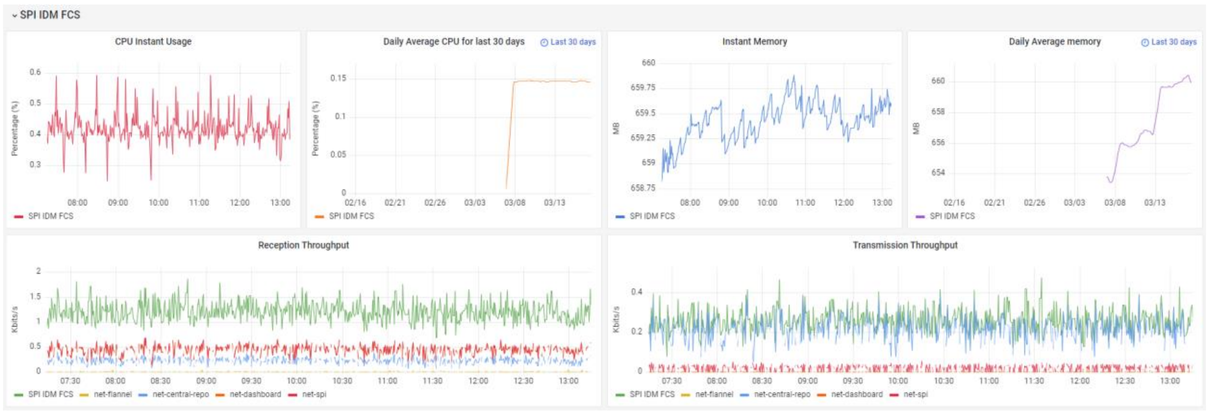


Figure 8. Panel of the SPI-IDM tool

These metrics include the ***CPU Instant Usage***, which shows the percentage of the number of cores that are being used by each container. This allows to monitor the resource utilization of a pod or container and can be used to identify performance bottlenecks, detect potential issues, or track the overall system health. On the right, the ***Daily Average CPU*** for the past 30 days is included, computing the average CPU utilization in the past 24 hours and extending this graph for 30 days, so that any strange behavior can be easily detected. The ***Instant Memory*** metric represents the amount of memory (in Megabytes) that is currently being used by a specific container. The working set is the portion of a process's memory that is held in RAM and actively used. It includes all memory that the kernel determines to be in use, including memory that is used by the kernel, by the container itself, or by any processes running inside the container.

In the same way as before, the daily average is computed and graphed for the past 30 days, to be able to detect anomalies. Finally, the transmission/reception ***Throughput*** or rate is displayed (in Kbits/s). We show this metric for the whole pod and for each network interface allocated to the pod. It is important to note that this metric will be specific to the pod's network namespace, meaning that it will only show the network traffic that is happening inside the pod and not the node of the Kubernetes cluster where the pod runs.

In particular, in the dashboard shown as an example in "Figure 8. Panel of the SPI-IDM tool", the performance of SPI Identity Manager is depicted both in instant values and average measures. A small percentage of CPU usage is appreciated due to the recent inclusion of the component in the domain, which means that it is already integrated but still not executing any demanding task. The same occurs with instant memory which suffers small oscillations but still requires little resources.

However, the average memory can be seen to grow as time passes, although it is still a small value, which is the expected behaviour as the pod begins to perform new tasks. Finally, the throughput is seen to be quite low as the pod is not exchanging much traffic with other pods yet. Once all the components are integrated and interconnected, this would be reflected in the different panels of the MON, that would allow to verify the proper final integration, as well as to monitor the performance metrics of the FISHY components during the functional tests that will validate their proper behaviour.

5.11 Smart Contracts

The Smart Contracts component is a validation mechanism for the different events/policies/reports that the rest of the FISHY components produce. The validation process involves the persistence of, for example, a report's details, that arrives from the Central Repository, in an IPFS node and the information to retrieve it from IPFS in a private Quorum network. The use of these two components ensures that any attempt to tamper with the data cannot go undetected.

The Smart Contract component is a Django application, always monitoring the RabbitMQ of the FISHY Central Repository for any new events/policies/reports that are written. The component, as well as the IPFS and the Quorum network, are available in a dockerized format and is integrated in the FRF as Kubernetes pods. The Smart Contracts component is connected to the RabbitMQ network, to read the Central Repository notifications and send back an answer to confirm the validation. An additional network to host IPFS and Quorum has been created and connected to the component.

5.11.1 Interfaces

Table 24. Smart Contracts inbound interfaces.

Origin	Data	Type of communication	Status
Central Repository	Notification of reports, events, policies of FISHY Components	AMQP	Completed

Table 25. Smart Contracts outnound interfaces.

Destination	Data	Type of communication	Status
IPFS	The details from the notifications of the Central Repository	HTTP	Completed
Quorum Network	The details of the stored notification to retrieve from IPFS	HTTP	Completed
Central Repository	Smart Contract validation answer	AMQP	Completed

6 Conclusions

The work done in WP5 and related WPs is summarized in this deliverable. All the tools that are part of the FISHY Reference Framework have been explained -in terms of deployment-. Also, latest updates on the Dashboard, IRO and SIA have been reported on this document.

This deliverable closes the WP5, where all the tools involved in the project have been integrated in a common framework. The FISHY Reference Framework is the stable platform created to provide the necessary connections for the tools to interact and communicate between them.

References

[1] Admela Jukan et al., FISHY Deliverable D2.2: IT-1 architectural requirements and design, H2020 FISHY project (Grant agreement ID: 952644), August 2021.

[2] OpenStack, Open source software for creating private and public clouds, [Online]. Available: https://www.openstack.org (accessed on March 24, 2023).

[3] Linux Foundation, Kubernetes - A production-grade container orchestration stack, [Online]. Available: https://kubernetes.io (accessed on March 24, 2023).

[4] S. Hares, D. Lopez, M. Zarny, C. Jacquenet, R. Kumar, J. Jeong, "Interface to Network Security Functions (I2NSF): Problem Statement and Use Cases," RFC 8192, Jul. 2017. [Online] Available: https://datatracker.ietf.org/doc/html/rfc8192 (accessed on March 28, 2023)

[5] ETSI GS NFV-SOL 005 V4.3.1, "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point," European Telecommunications Standards Institute, ETSI, 2022.

[6] ETSI GS NFV-SOL 006 V3.7.1, "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; NFV descriptors based on YANG Specification," European Telecommunications Standards Institute, ETSI, 2023.

[7] HAProxy, An open source software implementation of a high availability load balancer and reverse proxy for TCP and HTTP-based applications, [Online]. Available: https://www.haproxy.com (accessed on March 24, 2023).

[8] ETSI, Open Source MANO (OSM) - An open source NFV Management and Orchestration (MANO) software stack aligned with ETSI NFV specificactions, [Online]. Available: https://osm.etsi.org (accessed on March 24, 2023).

[9] L. F. Gonzalez, I. Vidal, F. Valera and D. R. Lopez, "Link Layer Connectivity as a Service for Ad-Hoc Microservice Platforms," in IEEE Network, vol. 36, no. 1, pp. 10-17, January/February 2022, doi: 10.1109/MNET.001.2100363.

[10] Universidad Carlos III de Madrid. L2S-M. Available at: http://l2sm.io (accessed on March 27, 2023).

[11] Linux Foundation, Helm – The package manager for Kubernetes, [Online]. Available: https://helm.sh (accessed on March 24, 2023).

[12] Luis F. Gonzalez, B. Nogales, I. Vidal and F. Valera. OSM PoC 14: Leveraging OSM virtual networking in Kubernetes clusters. Available at: https://osm.etsi.org/wikipub/index.php/OSM_PoC_14_Leveraging_OSM_virtual_networking_in_Kubernetes_clusters (accessed on March 28, 2023).

[13] Luis F. Gonzalez, I. Vidal, F. Valera, B. Nogales and Diego R. Lopez. Connectivity among CNFs using SDN. Available at: https://osm.etsi.org/gitlab/osm/features/-/issues/10921 (accessed on March 28, 2023).

[14] Open vSwitch. (v3.1.0), Linux Foundation. Accessed: March 24, 2023. [Online]. Available: https://www.openvswitch.org/

[15] Open Network Operating System. (2.7.0), Open Networking Foundation. Accessed: March 24, 2023. [Online]. Available: https://opennetworking.org/onos/

[16] Openflow® Switch Specification 1.3.0, ONF TS-006, June. 2012.

[17] Network Functions Virtualization (NFV) Release 3; Management and Orchestration; Interface and Information Model Specification for Multi-Site Connectivity Services, ETSI GS NFV-IFA 032 v3.2.1, ETSI, France, April. 2019

[18] Prometheus Official Documentation, [Online]. Available: https://prometheus.io/docs/ (accessed on March 28, 2023).

[19] Grafana Official Documentation, [Online]. Available: https://grafana.com/docs/grafana/ (accessed on March 28, 2023).

[20]Sysrepo. Storing and managing YANG-based configurations for UNIX/Linux applications. Available: https://www.sysrepo.org/

[21]Netopeer2 – NETCONF Server. Available: https://github.com/CESNET/netopeer2

[22] 5TONIC - An open research and innovation laboratory focusing on 5G technologies, [Online]. Available: https://www.5tonic.org/ (accessed on March 28, 2023).

[23] VMware, Bitnami - A Kubernetes applications catalog, [Online]. Available: https://bitnami.com/stacks/helm (accessed on March 24, 2023).

[24]Kubeadm. Available: https://kubernetes.io/docs/reference/setup-tools/kubeadm/