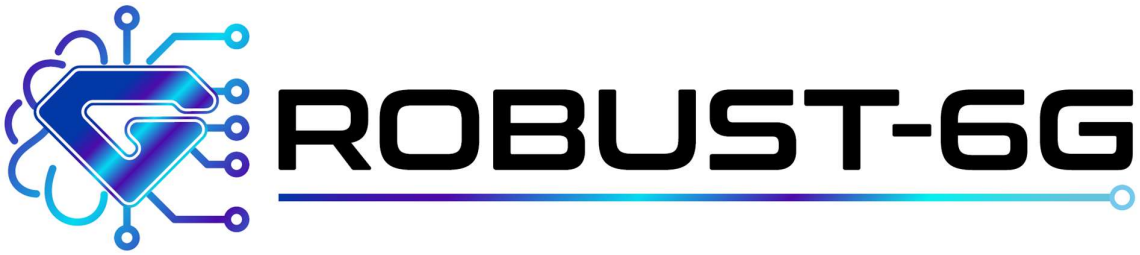


Smart, Automated, and Reliable Security Service Platform for 6G

Deliverable D4.4
ROBUST-6G AI/ML Driven Zero-Touch Security Management Platform Final Prototype

Co-funded by the European Union

6GSNS

ROBUST-6G project has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101139068.

Date of delivery:	31/03/2026	Version:	1.0
Project reference:	101139068	Call:	HORIZON-JU-SNS-2023
Start date of project:	01/01/2024	Duration:	30 months

6GSNS

Document properties:

Document Number:	D4.4
Document Title:	ROBUST-6G AI/ML Driven Zero-Touch Security Management Platform - Final Prototype
Editor(s):	Pietro G. Giardina, Marco Ruta (NXW)
Authors:	Contributors and their organisations are listed below
Contractual Date of Delivery:	31/03/2026
Dissemination level:	PU[1]
Status:	Final
Version:	1.0
File Name:	ROBUST-6G_D4.4_v1.0

Revision History

Revision	Date	Issued by	Description
0.1	20-01-2026	ROBUST-6G WP4	Initial ToC
0.2	16-03-2026	ROBUST-6G WP4	First complete draft
0.3	26-03-2026	ROBUST-6G WP4	Internal and external review
0.4	30-03-2026	ROBUST-6G WP4	Final complete draft after review
1.0	30-03-2026	ROBUST-6G WP4	Final version

Abstract

This document presents the final prototype of the ROBUST-6G Zero-Touch Security Platform (ZTSP). The platform automates security in complex 6G environments through four pillars: service orchestration, pervasive monitoring, AI-driven threat detection/prediction, and closed-loop automation. The document reports the last advancements in terms of design, implementation and integration of the different platform's components. Validation workflows demonstrate the platform's capability to manage security service lifecycles and execute reactive or predictive closed loops.

Keywords

Zero-touch security automation, Security orchestration, Service resource orchestration, AI/ML for security, AI/ML-based incident prediction, Threat detection, Programmable monitoring, Security closed-loops

[1] SEN = Sensitive, only members of the consortium (including the Commission Services). Limited under the conditions of the Grant Agreement

PU = Public

List of Contributors

Participant	Short Name	Contributors
Nextworks	NXW	Pietro G. Giardina, Marco Ruta, Giada Landi
THALES SIX GTS FRANCE SAS	THALES	Louis Cailliot, Dhouha Ayed
Universidad de Murcia	UMU	Alberto García Pérez, José María Jorquera Valero, Manuel Gil Pérez
AXON LOGIC IDIOTIKI KEFALAIOUXIKI ETAIREIA	AXON	Chih-Yang Pee
Chalmers University of Technology	CHA	Masoom Rabbani, Azadeh Tabeshnezhad, Tommy Svensson
EURECOM	EUR	Marios Kountouris, Ioannis Pitsiorlas
Linkopings Universitet	LIU	Nikolaos Pappas, Eunjeong Jeong

List of Reviewers

Participant	Short Name	Contributors
AXON LOGIC IDIOTIKI KEFALAIOUXIKI ETAIREIA	AXON	Wei Chuen YAU
University of Padua	UNIPD	Michele Rossi

Executive Summary

Deliverable D4.4 documents the final refinements in terms of design, implementation, and integration of the ROBUST-6G Zero-Touch Security Platform (ZTSP) that represent the culmination of a process started at M03 and terminated at M27 of the project. The primary objective of the work package is to introduce automated security mechanisms into complex, multi-domain 6G environments, enabling the system to enforce and maintain security levels without (or with limited) human intervention (zero-touch automation). This is achieved through four foundational pillars: i) security service and resource orchestration, ii) programmable pervasive monitoring, iii) AI-driven threat detection and prediction, and iv) closed-loop security automation.

The core of the architecture is the Security Management and Orchestration (M&O) stack, which includes the Security Orchestrator (SSO), the GenAI4SOAR, the Resource Orchestrator (SRO), and the Security Closed-Loop Manager. This deliverable describes updates across all these components, with major advancements in the SSO and GenAI4SOAR modules.

The SSO is the entry point of the security orchestration, managing the lifecycle of end-to-end security services by translating high-level Security Service Level Agreements (SSLAs) into a set of elements building the security service e.g., security applications, configurations, target environments, etc. One of the key advancements of this final prototype is the refinement of the platform's semantic engine. The security ontology has been extended by introducing a dedicated Security Actuator class so that the system now explicitly distinguishes between passive monitoring tools and active enforcement mechanisms, allowing for more precise and automated remediation.

The GenAI4SOAR module evolved from a monolithic Large Language Model (LLM) approach towards a collaborative "crew" of three specialised AI agents: a Cyber Threat Intelligence Analyst, a Cyber Security Software Architect, and an OpenC2/CACAO Expert. This multi-agent design allows each agent to focus on its specific domain of expertise, such as threat analysis or the generation of syntactically correct security playbooks based on OASIS standards. This reduces errors and improves accuracy in the generation of security remediation actions (security playbooks) e.g. by reducing AI “hallucinations".

In addition to the orchestration stack, the Programmable Monitoring Platform (PMP), has been upgraded to supports network traffic analysis in order to determine behavioural relations between different traffic flows monitored. Furthermore, the data exposure has been enhanced by minimising the access latency through the use of Redis as a buffer cache.

The monitoring data can be used to feed advanced AI modules trained on the new CICToN-IoT dataset, which integrates diverse IoT and crypto mining traffic, relevant for the different UC2 scenarios. These models have demonstrated high performance, achieving a 98.15% accuracy rate in network attack detection.

The ZTSP's operational validity has been demonstrated through the definition and validation of different operation workflows covering ZTSP pre-configuration operations, security service provisioning and decommissioning along with three distinct types of security closed loops: i) rule-based reactive loops using traditional IDS alerts, ii) AI-driven reactive loops for advanced anomaly detection, and iii) AI-driven predictive loops capable of forecasting potential attacks within a five-minute window based on preceding traffic patterns.

With the finalisation of the design, implementation, of the different modules and their integration into a ZTSP prototype, WP4 has fulfilled its primary objectives. From M28 to the end of the project (M30) the work will continue in the context of WP6 where the platform’s end-to-end capabilities will be demonstrated within specific project Use Cases.

List of Tables

List of Figures

Acronyms and abbreviations

Term	Description
ADMM	Alternating Direction Method of Multipliers
AoMA	Age of Missed Alarm
AoFA	Age of False Alarm
AI	Artificial Intelligence
APIs	Application Programming Interfaces
CM	Catalogue Manager
CPT	Cumulative Prospect Theory
CTI	Cyber Threat Intelligence
CVE	Common Vulnerability and Exposure
ECS	Elastic Common Schema
GAN	Generative Adversarial Network
IDS	Intrusion Detection System
IoT/IIoT	Internet of Thing/Industrial IoT
K8s	Kubernetes
KPI	Key Performance Indicator
LCM	Life Cycle Management
LLM	Large Language Model
LR	Lagrangian Relaxation
MID	Machine ID
ML	Machine Learning
NBI	Northbound Interface
OM	Ontology Manager
PMP	Programmable Monitoring Platform
PSM	Projected Subgradient Method
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RTDB	Real-Time Database
SBI	Southbound Interface
SCA	Successive Convex Approximation
SCM	Security Context Manager
S-CL Mgmt	Security Closed Loops Management
SLO	Security Level Objective
SOAR	Security Orchestration, Automation and Response
S-RO	Security Resource Orchestration
SSLA	Security Service Level Agreement
VSB	Vertical Service Blueprint
ZTSP	Zero Touch Security Platform
ZTSO	Zero Touch Security Orchestrator

1 Introduction

The main objective of WP4 is to bring security in complex multi-domain and multi-stakeholder 6G environments, by implementing specialized mechanisms capable of enforcing, monitoring, and maintaining a given level of security in target critical environments belonging to the 6G systems. This would include the different 5G/6G network (RAN, Transport, Core) and cloud segments (Extreme/Far Edge, Edge, Cloud). In this regard, WP4, at its beginning, identified four main functionalities to be implemented and integrated into a dedicated security platform, the Zero-Touch Security Platform (ZTSP): i) security service and resource orchestration, ii) programmable pervasive monitoring, iii) threat/anomaly detection and prediction, and iv) closed-loop-based security automation. During its timeframe, the work package worked to design and implement the ZTSP and its main functionalities, reporting the progresses in a set of deliverables of which D4.4 is the last representative. In particular, this deliverable reports the final phase of the work carried out in the period M22-M27, representing the conclusion of a process that began at M03 and culminated with the realisation of a ZTSP prototype.

1.1 Scope and Objective

The main goal of this document is to report the final activities of WP4, focusing on the finalisation of the ROBUST-6G Zero-Touch Security Platform (ZTSP) prototype. These activities encompass the software design refinement of each component and the related implementation of new functionalities (or enhancements of existing functionalities) already reported in the previous deliverables. WP4 aims at delivering an integrated ZTSP prototype, not the single components. For this reason, the final part of the work packages has been dedicated to the integration of the different components constituting the ZTSP, in order to deliver a unified prototype capable of managing security services and security automation via dedicate closed-loops (CLs.)

1.2 Document Outline

The document is structured in three main sections that report different types of advancements. The final software design and implementation of the Zero-Touch Security Platform and its modules is reported in Section 2. To avoid any repetition, the section reports only the delta with respect to what already reported in the previous deliverable D4.3 [R6G25-D43], which remain the reference for those software architecture and functionalities that have not undergone any change. In this context, each module is accompanied by a table summarising its main functionalities, where only new or updated functionalities are described in detail.

The integration among the different ZTSP modules is reported in Section 3. In particular, the section defines the two main operation workflows for the security service orchestration, namely security service provisioning and decommissioning. For each step shows screenshots for the real system to demonstrate the integration among the modules involved. A similar work has been done with the three examples of closed loop reported in the section: rule-based, AI-based (reactive), and AI-based (predictive).

Finally, Section 4 discusses the objectives and KPIs achieved through WP4 activities and results.

2 ROBUST-6G Zero-Touch Security Platform

The Zero-Touch Security Platform is the main outcome of WP4; it was designed, developed, and refined over the work package timeframe. The below Figure 2-1 shows the high-level SW architecture, an evolution of the functional architecture reported in D4.3, which highlights the main components of each part of the platform. With reference to the figure, it can be noted that the ZTSP exposes interfaces to interact with external components, i.e., Exposure Framework, Trustworthy and Sustainable AI Service Layer, and 6G Managed System. These interactions, briefly described later in this section, are part of the ROBUST-6G integrations carried out in WP6. For that reason, although the architecture can be considered stable, the feedback collected from the integration activities may lead to further refinements beyond the end of WP4.

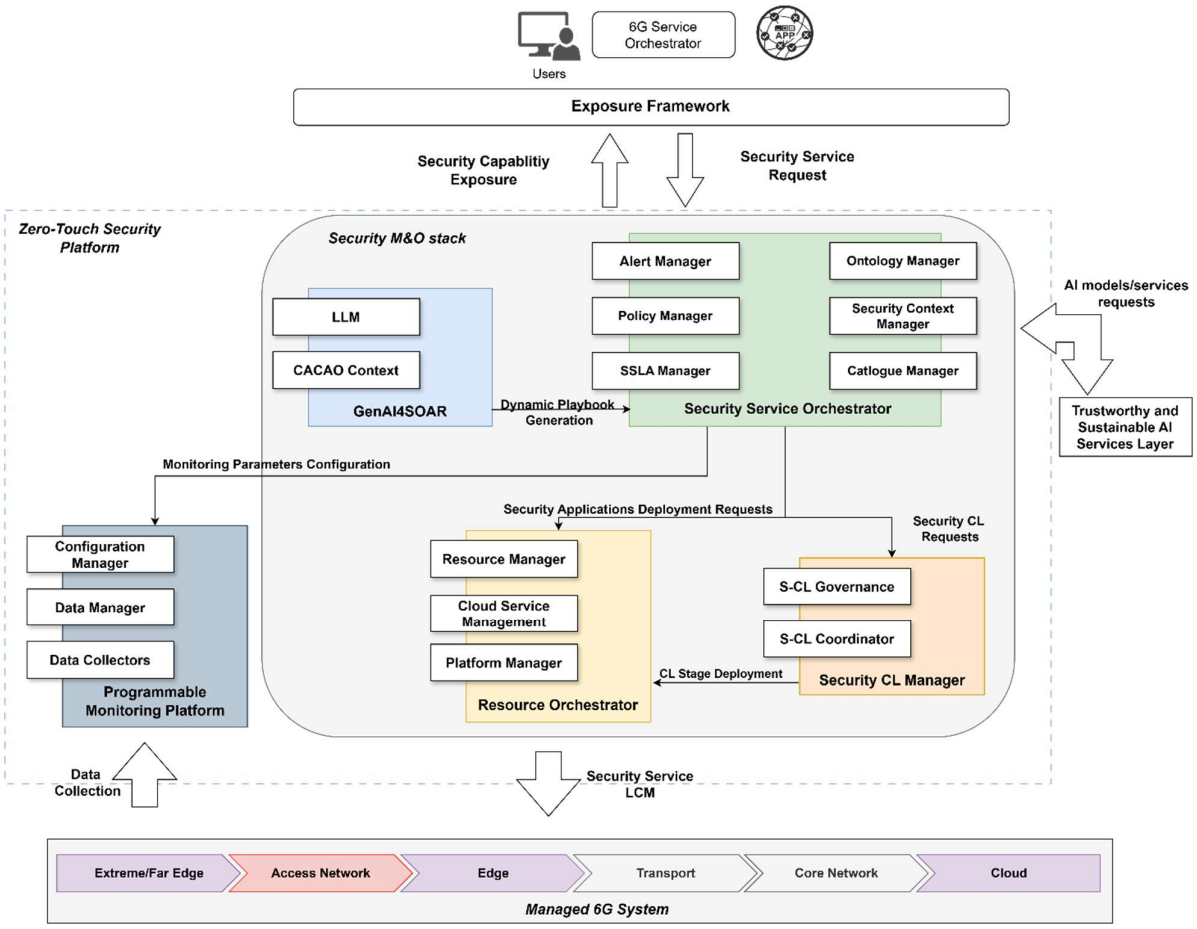


Figure 2-1 Zero-Touch Security Platform's high-level software architecture

The ZTSP architecture comprises five major entities (or blocks) that provide all the functionalities required for security orchestration, automation, and anomaly detection. GenAI4SOAR, Security Service Orchestrator, Resource Orchestrator, and Security Closed-Loop (CL) Manager can be further logically grouped into the Security Management and Orchestration (M&O) stack, which encompasses all the orchestration functionalities. A brief description of each block, along with its main software components, is provided below.

- **Security Service Orchestrator:** Is the main entity of the platform, in charge of managing the lifecycle of end-to-end (E2E) security service, as well as maintaining security policies, Service Level Agreements (SLAs), and semantic entities (security ontology and knowledge graph). Its main internal modules are:
    - **Alert Manager:** Manages alerts generated by the SIEM. If an alert is correlated to an existing security playbook, the latter is executed as a remediation action. If not, a new playbook is requested from GenAI4SOAR. A security playbook is a structured entity e.g. a file, encompassing a structured set of procedure to react/mitigate cybersecurity incidents.

  - **Policy Manager:** Manages the Security Policies (ingestion, parsing, validation) from where security requirements are extracted by querying the SSLA Manager in the form
  - SLOs (Security Level Objective). SLOs are used to generate a security orchestration request. The Policy Manager is also responsible for contextualizing policies and alerts with infrastructure and environment information the ZTSO orchestrates, to generate dynamically and automatically remediation playbooks to maintain those policies or to mitigate threats.
  - **SSLA Manager:** Management, parsing, and validation of Security Service Level Agreements formatted according to the SPECS Standard [specs15].
  - **Ontology Manager:** Represents the ZTSP semantic engine and is responsible for the platform's semantic layer. It manages the security ontology, the knowledge graph, and provides the reasoning that is crucial for the composition of security services.
  - **Security Context Manager:** Responsible for security service lifecycle management, including provisioning, decommissioning, service status, etc.
  - **Catalogue Manager:** Maintains catalogues related to available Security Functions, Target Environments, and Infrastructure.
- **GenAI4SOAR:** Responsible for the dynamic generation of security playbooks according to the CACAO v2.0 and OpenC2 specifications from the OASIS [CACAO23] [OpenC222]. The playbooks are generated by exploiting AI agents and Large Language Models (LLMs) specifically coordinated in a crew, dedicated for the design of mitigations workflow in respect for the former standards, allowing their dynamic and automatic customization based on security service characteristics. As the generated playbooks may contain errors, they can be accessed and updated by a human operator if necessary.
- **Resource Orchestrator:** Manages the computational resources and the lifecycle of the security application in the different cloud segments i.e., Far Edge, Edge, Cloud. It encompasses three main modules:
  - **Platform Manager:** Maintains the coordinates e.g., IP address port, etc. of the different cloud platforms (e.g., Kubernetes).
  - **Resource Manager:** Collects and maintains resource information from the integrated cloud platforms.
  - **Cloud Service Management:** This management is realized through two different components: i) Cloud Service Lifecycle Manager, for cloud-based security applications and ii) Cloud Service Repository, which maintains the different security application descriptors.
- **Security CL Manager:** Manages two aspects of the Security CLs. On the one hand, it selects the proper stages for building the loop and manages its provisioning and decommissioning through the CL Governance. On the other hand, through the CL coordination it prevents conflicts between loops insisting on the same set of resources at the same time.

Aside the orchestration stack, the **Programmable Monitoring Platform** (**PMP**), provides the implementation of the Programmable Pervasive Monitoring, which is crucial for building an awareness of the system status and enabling security automation. It encompasses three main modules.

- **Configuration Manager:** Enable runtime configuration of monitoring and detection jobs using specific collection plugins (e.g., Telegraf) and intrusion detection systems (e.g., Snort).
- **Data Collectors:** A set of collection plugins and IDS applications configurable at runtime.
- **Data Management:** Although represented as a single module for the sake of simplicity, it is actually the logical representation of several modules responsible for i) data manipulation, ii) data storage, and iii) data exposure to consumers.

The security functionalities provided by the ZTSP are accessible via the Exposure Framework that provides both an abstraction of the orchestration APIs and an overview of the available security capabilities. It is important to highlight that the interaction depicted in the figure is simplified to ease the architecture readability. As reported in D2.3 [R6G26-D23], the Exposure framework is also directly connected with the Data Management Layer, which collects the security capabilities from the security orchestration. Furthermore, system administrators always have the possibility of directly accessing the ZTSP interfaces bypassing the Exposure Framework, although not shown in the figure.

The Security Service, once requested, can be provisioned in the underlying 6G Managed System, i.e., the set of network and cloud segments building a 6G network, by following the procedure defined and demonstrated

in Section 3.1.4. Currently, the orchestration works mainly with cloud segments (Far Edge, Edge, and Cloud), while the network segments have only been partially targeted. In particular, ZTSP will integrate PHY layer security in the Access Network, as part of an inter-layer integration planned in WP6. Although Core and Transport network have not been targeted by WP4, the technology used to define and execute security playbooks (CACAO, OpenC2), can be exploited to also enforce security actions in those segments as well.

The integration of AI in ZTSP is two-fold. With GenAI4SOAR, the platform exploits agents and LLMs to generate playbooks at orchestration time, so that the orchestration of the security service can be considered AI-driven at least in part. The second usage is in the security CLs where Analysis and Decision stages can exploit AI/ML algorithms to detect/predict anomalies and threats, and to decide the remediation. This enables AI-driven reactive/predictive orchestration, where security actions are automatically triggered upon the detection or prediction of anomalies.

2.1 Security Service and Resource Orchestration

This section reports design and implementation updates with respect to D4.3 [R6G25-D43], related to the Security Service and Resource Orchestration layer of the ROBUST-6G Zero-Touch Security Platform. It describes how the Zero-Touch Security Orchestrator (ZTSO), Security Closed-Loop Management (S-CL Mgmt) and Security Resource Orchestrator (S-RO) work together to translate Security policies, into Security Service Level Agreements (SSLA) format, at the business level into actionable security functions establishing and maintaining an optimal security posture in a target infrastructure. As detailed in Section 2.1.1, the ZTSO operates at both the business and service levels: at the business level, it accepts security policies from verticals via the North-Bound Interface and interprets them by translating SSLAs into a set of Security Level Objectives (SLOs) metrics; at the service level, it enables the composition of infrastructure-specific security services, composed of Security Functions (SFs) and Security Closed Loops (S-CLs), that are deployed in the target infrastructure to enforce these policies. To implement these policies within the target infrastructure, the ZTSO coordinates two deployment/orchestration components: the S-RO and the S-CL Mgmt. As described in D4.3 and further detailed in Section 2.1.3, the S-CL Mgmt is responsible for instantiating, orchestrating and coordinating S-CLs. These are automation constructs that establish and maintain the security posture derived from a policy. As described in D4.3 and further detailed in Section 2.1.4, the S-RO acts as the actuator across the cloud–edge continuum, deploying and operating the security applications and S-CL functions required by ZTSO's plans and by the S-CL management to establish and enforce security automations. The following sections provide a more detailed overview of the main functionalities, design updates and implementation details of these components.

2.1.1 Zero-Touch Security Orchestrator (ZTSO)

The Zero-Touch Security Orchestrator, whose updated software architecture is reported in Figure 2-2. With respect to the architecture defined in D4.3 [R6G25-D43], a single major change can be noted. In the Security Context Manager (SCM), the previous Service Context Manager and Orchestrator Clients submodules are now integrated and implemented as a single comprehensive component, the Security Service Lifecycle Manager. The SCM is now composed of only two modules: the Infrastructure Context Manager, unchanged from D4.3, and the Security Service Lifecycle Manager. This architectural consolidation allows for a cleaner separation of functional concerns with respect to two main aspects related to Security Services: the composition of a Security Service (SSe), starting from an SSLA or an Alert and resulting in the semantic-driven choice of SFs and S-CLs configurations and the instantiation and management of the SSe lifecycle. While the other components were considered finalized in terms of design in D4.3, the SCM is the only component that was declared in an early design stage. In the following sections, details about the design and implementation updates of all the ZTSO components are provided.

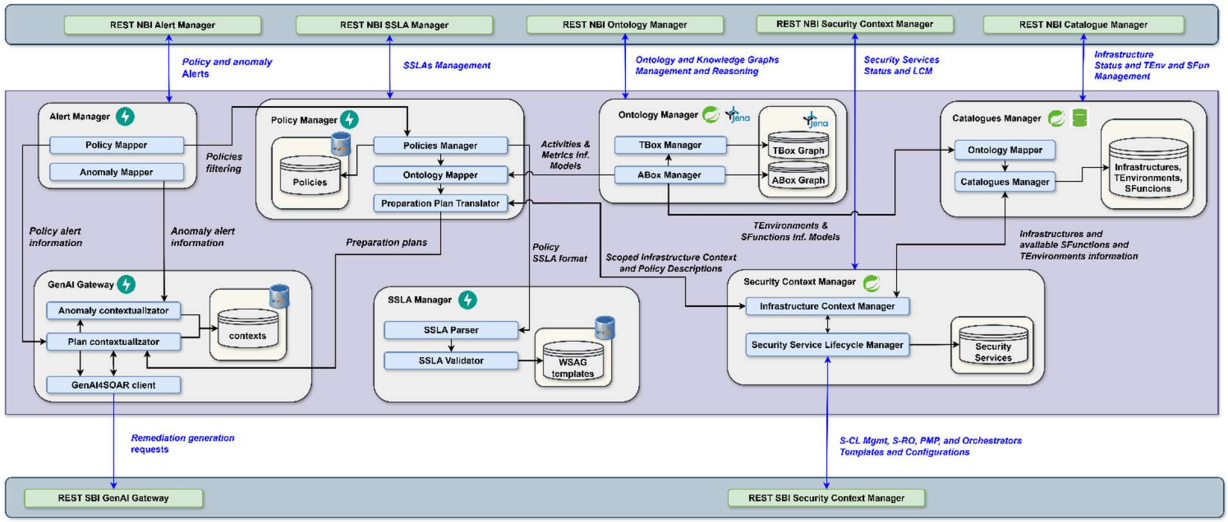


Figure 2-2 ZTSO Updated software components architecture

2.1.1.1 Main functionalities

The main functionalities of the Zero-Touch Security Orchestrator, first defined in D4.3 [R6G25-D43], are summarized in

Table 2-1. This table provides a high-level description of each functionality together with its update status (YES / NO / NEW). After the table, detailed descriptions are provided for those entries that have changed. Overall, the updates are driven by three main goals: (i) a more expressive and operationally complete semantic model of Security Functions, distinguishing passive applications from active enforcement actuators; (ii) an enriched catalogue that stores machine-readable interaction manuals for every registered security component; and (iii) the complete design and implementation of the Security Context Manager (SCM) that closes the loop between AI-generated remediation plans and the deployment of complete security services comprehensive of SFs and S-CLs.

Table 2-1 List of ZTSO Functionalities

Name	Description	Updated (YES/NO/NEW)
Ontology Management	Enables the management of the ontology schema (TBox), allowing uploading, retrieving, and clearing the ontology definition.	YES
Knowledge Graph Management	Enables the management of instance-level knowledge (ABox) within the Security Orchestrator knowledge graph.	YES
Security Functions Management	Enables the management of Security Functions, allowing the onboarding, querying, and deletion of Security Functions (SFs).	YES
Data Models Management	Provides access to semantically enriched data models, exposing structured representations of security functions, capabilities, metrics, activities, and target environments.	NO
Target Environments Management	Provides functionalities to onboard, query, and manage Target Environments (e.g., Kubernetes clusters, IoT environments, etc.).	NO
Infrastructure Context Retrieval	Provides a scoped infrastructure context (available and deployable Security Functions, and available Target Environments) to the Policy Manager.	YES
Security Service Assembly	Enables the assembly of security service blueprints by linking Security Functions with Security Closed Loops configurations and remediation playbooks.	NEW

Service Lifecycle Management	Enables the inspection, query, and deletion of security service instances.	NEW
Policy and Compliance Automation	validates security policies (e.g., SSLA standards) to extract Security SLOs (Service Level Objectives) and adapts the orchestrated security functions to fulfil the policies at deployment time. It also automatically generates remediation playbooks to maintain the security policies in the environments and infrastructures contexts.	YES
AI-Driven Security Remediations	Generates CACAO/OpenC2-compliant playbooks using GenAI4SOAR (AI agents + LLMs) for customized threat mitigation. The ZTSO also leverages AI/ML models in Closed Loops for anomaly detection and prediction and autonomous remediation decisions.	YES

The updates to the functionalities described above were introduced to enhance the adaptability of the Security Service Orchestrator when dealing with complex, multi-domain scenarios. Specifically, these changes provide a much better and more granular modelling of the *Security Functions* and *Target Environments* from a semantic point of view. By refining how components are classified and managed, the orchestrator can more accurately align security policies with the actual capabilities and execution mechanisms present in the underlying infrastructure. Moreover, two new functionalities were introduced for the assembly and lifecycle management of SSe. These two new functionalities, already envisioned in D4.3 [R6G25-D43], are now provided by the SCM, in particular by the Security Service Lifecycle Manager, as described in the following sub-sections.

2.1.1.2 Design updates

- **Semantic modelling updates**: In the previous design, the ontology provided a generalized view of *SecurityFunctions*. However, to effectively support automated remediation, we needed to explicitly differentiate between passive monitoring/analysis tools and active enforcement mechanisms. To address this, the semantic model (TBox) was updated by splitting the security function hierarchy to introduce a dedicated *SecurityActuator* class. While security applications provide functional capabilities (like threat detection), actuators act as the operational hands of the system. *SecurityActuators* are defined as entities capable of actuating on both *SecurityApplications* (e.g. to enforce a particular configuration and/or start an activities) and *TargetEnvironments* (e.g. to apply a configuration to a Target Environment). Consequently, the ABox was updated to reflect the updates of the TBox.

- **Catalogue Management updates**: To support the new semantic paradigm described above, the catalogue management functionality was updated to shift from storing only high-level metadata of security functions and target environments to acting as a complete repository for technical descriptors. Because the orchestrator now deals with SecurityApplications exposing APIs and SecurityActuators exposing APIs to interact with the SecurityApplications and TargetEnvironments, the catalog was re-designed to store and provide the actual manuals for interactions with SecurityApplications and SecurityActuators.

- **Infrastructure Context Retrieval update:** The Security Context Manager (SCM) continues to expose a scoped infrastructure context to the Policy Manager as its primary northbound interface. In the updated design, this context is enriched by the richer catalogue: it now includes not only the list of available SecurityApplications and TargetEnvironments for a given infrastructure, but also the registered SecurityActuators and the can_actuate_on / can_actuate_on_environment relationships. This richer context enables the Policy Manager, acting in conjunction with the GenAI Gateway and GenAI4SOAR, to generate remediation playbooks that are both semantically grounded and directly actionable, referencing real actuator identifiers and their interaction semantics.

- **Security Service Assembly and Lifecycle Management**: In D4.3 [R6G25-D43], the Security Service Assembly functionality was described as a design goal, with only the Infrastructure Context Retrieval functionality implemented as an initial prototype. In the finalized design, the SCM implements the complete two-step interaction with the Policy Manager. In the first step, unchanged from D4.3

[R6G25-D43], the SCM provides the scoped infrastructure context (the set of available SecurityApplications, SecurityActuators, and TargetEnvironments semantically relevant to the requested security scenario) retrieved by validating the orchestration plan against the ontology and querying the Catalogue Manager. In the second step, the SCM acts as the assembly engine that validates the submitted Security Function identifiers against the stored catalogue result and advances the plan towards deployment. Plan finalization, comprising the Security Function selection from the ones available in the scoped context, and the associated CACAO playbook, can follow two paths: it can be submitted automatically by the Policy Manager with support of the GenAI4SOAR as the concluding step of the AI-driven orchestration workflow, or it can be performed manually by an authorized administrator who selects the Security Functions and provides the CACAO playbook, which may still be generated with the support of GenAI4SOAR. The resulting plan descriptor fully specifies the security service to be deployed and managed: it carries (i) the list of Security Functions (SecurityApplications and SecurityActuators) selected and validated against the available and deployable functions retrieved from the catalogue; (ii) the CACAO playbook encoding the remediation workflow to be enforced; and (iii) the Vertical Service Blueprint (VSB), authored by an authorized administrator, that binds the selected Security Functions to their deployable component templates registered in the S-RO and to any S-CL templates registered in the S-CL Mgmt, together with the full parameterisation required at instantiation time. This separation between plan finalization and VSB authoring preserves flexibility in how the security service is composed: the upper layer of the ZTSO remains focused on SSLA interpretation, security semantics, policy generation, and AI-assisted remediation generation, while the VSB and the Security Service Lifecycle Manager handle the operational details of deploying and managing service instances across the cloud-edge continuum.

2.1.1.3 Prototype Implementation

This section reports the updates regarding the prototype implementations of all the components of the ZTSO.

- **Ontology Manager:** With respect to what was already stated in D4.3 [R6G25-D43], the Ontology Manager interfaces and APIs are the same. The Ontology Manager is implemented as a Spring Boot microservice exposing a REST API, deployed as a containerised component within the ZTSO Helm chart. It acts as the semantic engine of the ZTSO, managing two logically separate named graphs stored in an Apache Jena Fuseki triplestore: a TBox graph (http://security-orchestrator.nextworks.it/graph/tbox/) holding the ontology schema, and an ABox graph (http://security-orchestrator.nextworks.it/graph/abox/) holding the instance-level knowledge graph. The only modification with respect to the TBox defined in D4.3 concerns the ROBUST-6G TBox. The ABox is inherently dynamic, based on the information model defined in the TBox, and will be updated and populated accordingly with the appropriate instances to demonstrate the ROBUST-6G security orchestration use cases.

    - *TBox updates*: With respect to D4.3, the ontology schema (the .ttl file loaded into the TBox graph) was extended in two ways. First, the *SecurityActuator* class was added as a sub-class of *SecurityFunction* (alongside the pre-existing *SecurityApplication* and *SecurityConfiguration*), providing a dedicated type for active enforcement components. Second, two new object properties were introduced: *can_actuate_on* (domain: *SecurityActuator*, range: *SecurityApplication*) and *can_actuate_on_environment* (domain: *SecurityActuator*, range: *TargetEnvironment*). Additionally, datatype properties for descriptive attributes of security components (e.g., actuation_profile, protocol) were added to support the enriched catalogue descriptors.

- **Catalogues Manager:** With respect to what was already presented in [R6G25-D43], the Catalogues Manager interfaces and APIs are the same. The Catalogues Manager is implemented as a Spring Boot microservice exposing a REST API, co-deployed with the other ZTSO components within the same Helm chart, backed by a PostgreSQL database. The only significant update concerns the data models for SecurityFunctions and TargetEnvironments, which were extended to carry a complete JSON descriptor alongside the semantic metadata already present in D4.3.

- *Security Functions descriptors updates*: The descriptor is differentiated by sub-type. For SecurityApplication instances, the descriptor includes the full OpenAPI specification of the application's REST interface, covering available endpoints, input/output schemas, and authentication requirements, everything needed to programmatically invoke the application from the SCM or from a CACAO playbook executor. For SecurityActuator instances, the descriptor follows a structured JSON format (manualDescriptor) which captures the actuator's interaction profile: the target application or environment it acts upon, the OpenC2 action/target pairs it supports, and any configuration parameters required at actuation time.

- *Target Environments descriptors updates*: the descriptor was extended to include the platform type (e.g., Kubernetes cluster, ThingsBoard instance, Docker-based environment), access endpoint coordinates, credential references, and the set of non-functional capabilities exposed by the environment, providing the SCM with the full picture of where and how a security service component can be deployed or enforced.

- **Security Context Manager (SCM)**: As already stated in D4.3 [R6G25-D43], the SCM is implemented as a Java Spring Boot microservice deployed within the ZTSO Helm chart. Its internal structure reflects the architectural separation described in Section 2.1.1.2: an Infrastructure Context Manager sub-component, unchanged from the D4.3 prototype, and the Security Service Lifecycle Manager sub-component. The Infrastructure Context Manager retains the same behavior described in D4.3: upon receiving an orchestration plan from the Policy Manager, it validates the plan against the ontology, to verify that all declared activities and metrics are known within the TBox, then queries the Catalogue Manager to retrieve the set of SecurityApplications, SecurityActuators, and TargetEnvironments that are semantically relevant to the security scenario at hand. This scoped infrastructure context is returned to the Policy Manager, which forwards it to GenAI4SOAR to support the generation of the remediation plan and the CACAO playbook. Plan finalization, comprising the Security Function selection and the associated CACAO playbook, can then follow two paths: it can be submitted automatically by the Policy Manager as the concluding step of the AI-driven orchestration workflow, or it can be performed manually by an authorized administrator who selects the Security Functions and provides the CACAO playbook, which may still be generated with the support of GenAI4SOAR. Once the finalized plan is submitted back to the SCM, the Security Service Lifecycle Manager takes over, driving the plan through VSB association and Service Instance instantiation via the S-RO Client and S-CL Management Client southbound plugins until the service reaches a fully operational state. Each orchestration plan is governed by a dedicated per-plan state machine whose transitions are summarized in Figure 2-3.

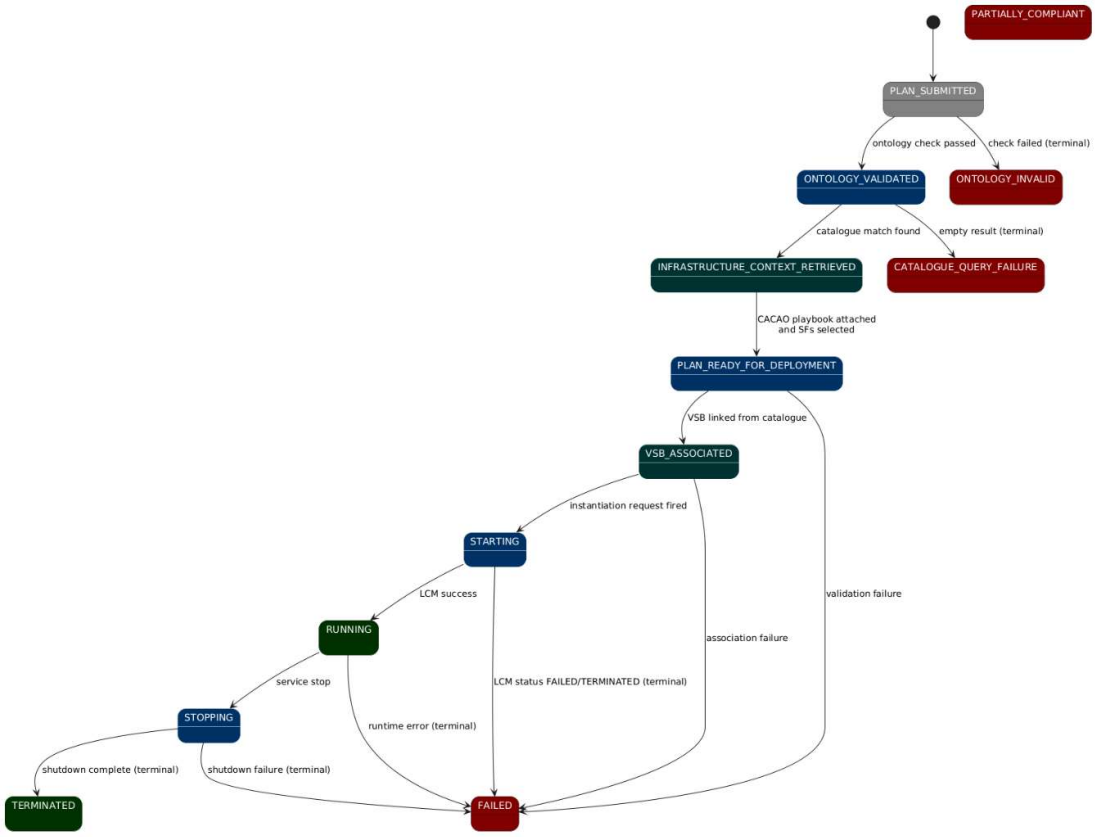


Figure 2-3 SCM State Machine

Upon plan submission the SCM machine enters PLAN_SUBMITTED and immediately performs ontology validation: success advances the plan to ONTOLOGY_VALIDATED; failure terminates it in ONTOLOGY_INVALID. From ONTOLOGY_VALIDATED the Catalogue Manager is queried: a full match advances the plan to INFRASTRUCTURE_CONTEXT_RETRIEVED while an empty or partial result terminates the plan in CATALOGUE_QUERY_FAILURE. Once the Policy Manager submits the finalized plan, the Security Function (SF) identifiers are validated against the stored catalogue results, advancing the plan to PLAN_READY_FOR_DEPLOYMENT. An authorized administrator then associates a Virtual Service Blueprint (VSB) from the SOCCER VSB Catalogue, moving the plan to VSB_ASSOCIATED. The deployment step subsequently creates a Service Instance and issues an instantiation request to the SOCCER Life Cycle Management (LCM), transitioning the plan to STARTING; a background scheduler polls the LCM at a configurable interval (set to 10 seconds by default to balance system overhead with responsiveness) and transitions the plan to the terminal state RUNNING on success, or to FAILED on a reported FAILED or TERMINATED LCM status. Failure transitions to FAILED are additionally defined from PLAN_READY_FOR_DEPLOYMENT and VSB_ASSOCIATED to handle errors arising during those intermediate steps. The management interfaces exposed by the SCM are summarized in Table 2-2 while a complete list of the APIs exposed by the SCM components in provided in [Nex25f].

Table 2-2 Security Context Manager Interfaces

Interface	Consumers in the ZTSO	Description
Orchestration Plan Submission	Policy Manager	Accepts a proactive security orchestration plan from the Policy Manager. The plan is validated against the ontology to verify that all declared activities and metrics are known, then the Catalogue Manager is queried to retrieve the matching infrastructure context (available SecurityApplications, SecurityActuators, and

		TargetEnvironments). The resulting context is returned to the Policy Manager to support security plan finalization.
Plan Finalization	Policy Manager, ZTSO Admin	Receives the finalized security plan from the Policy Manager or the ZTSO Admin, comprising the confirmed selection of Security Functions and the associated CACAO remediation playbook. All submitted Security Function identifiers are validated against the catalogue result stored for the plan. On success, the plan is marked as ready for VSB association.
VSB Association	ZTSO Admin	Allows an authorized administrator to associate a Vertical Service Blueprint (VSB) from the Service Lifecycle Manager VSB Catalogue with a finalized plan. The VSB is fetched and validated before being linked to the plan, enabling the Security Service Lifecycle Manager to proceed with Service Instance creation and instantiation.
Service Deployment	ZTSO Admin	Triggers the creation and instantiation of a Service Instance for a plan that has a VSB associated. The deployment is initiated immediately and the plan transitions to a starting state; background polling then monitors the Security Service Lifecycle Manager until the service instance reaches a fully operational status.
Plan Status Query	ZTSO Admin, Policy Manager	Enables querying of the current execution state and associated metadata of a security orchestration plan, including the current state machine state, the infrastructure context retrieved, and the linked VSB and Service Instance identifiers where applicable.
Plan Termination	ZTSO Admin, Policy Manager	Allows an authorized administrator to stop the execution of a running security orchestration plan, triggering the termination of the associated Service Instance through the SOCCER LCM.

- **Policy Manager:** As stated in [R6G25-D43], the Policy Manager creates the preparation plans that outlines security functions, activities, applicable security policy, and information about the infrastructure and the execution environment, from a user's policy submission. Due to the new design that integrates a GenAI Gateway to connect the ZTSO to generative AI services (as described below), new functionalities for the Policy Manager are implemented. Thanks to the already existing connections of the Policy Manager with the Security Context Manager, the Policy Manager is now responsible for requesting infrastructure and environment context to forward them to the GenAI Gateway. This implementation involves:
    - A new backend process for the Security Context Manager REST client. This process handles GET requests to receive infrastructure and environment information regarding a policy by providing an ID.
    - A new REST API endpoint, that listens to REST GET requests for infrastructure and environment information by providing a policy ID. It invokes the new process described above.

  To handle the created CACAO playbook from the GenAI Gateway component, the Policy Manager also implements:
    - A new backend process for the Security Context Manager REST client. This process handles POST requests to send CACAO playbooks that must be enforced on targeted infrastructure and environments.
    - A new REST API endpoint, that listens to REST POST requests for CACAO playbooks. It invokes the new process described above.
- **GenAI Gateway:** To fulfill the role of this component in creating CACAO playbooks, as well as enforcing them, the Policy Manager REST client is implemented for security policy maintenance plans or remediation plans for security alerts. It handles REST POST requests towards the Policy Manager that contains created CACAO playbooks. The GenAI Gateway can afterward send the generated

playbooks to the Security Context Manager, which will enforce them in Security Orchestration Automation and Response (SOAR) frameworks.

2.1.2 GenAI4SOAR

As stated in [R6G25-D43], the main objective of using generative AI to produce mitigation playbooks is to reduce the mean time to respond to security alerts for organization's security teams. Since the security team's mitigation processes are manually written by security experts, the usage of generative AI in ROBUST-6G reduce the time to produce mitigation workflows. By enforcing the produced workflows in Security Orchestration Automation and Response (SOAR) frameworks, security experts can also review them to validate the process regarding their organization's policy.

The initial design of the GenAI4SOAR component worked as a single large language model (LLM) responsible for all stages of playbook generation. It took high-level security policies or alerts, contextualized with infrastructures and environments details, and then generated CACAOv2-compliant playbooks.

This approach demonstrated several limitations:

- A single model had to handle all the context in a single task: threat analysis, tool selection, technical command generation and CACAO compliance, leading to lower accuracy and hallucinations in the results.
- No validation of CACAO playbooks against the standard language specification, beyond what the LLM could infer from prompts, leading to incorrect playbooks impossible to translate or enforce in SOAR frameworks.
- Limited modularity, making it harder to update or refine individual components without affecting the entire generation.

The current implementation replaces this monolithic structure with a collaborative crew of three specialized agents, each powered by two LLMs respectively optimized for security alerts contextualization and code writing for CACAO format. The new design includes the usage of the security catalogue from the ZTSO to produce workflows compliant with organization security assets. In this case, AI models developed in WP3 can be included in this catalogue to produce dynamically workflows containing steps powered by AI to mitigate threats. Finally, the new design also introduces OpenC2 commands inside the workflow steps of the remediation playbook, to reach the desired level of interoperability with the orchestrated domains.

The new crew introduces the following agents, whose details are reported in Table 2-3:

- **Cyber Threat Intelligence (CTI) Analyst** powered by a textual LLM like Mistral Large:
  - Analyses raw threat reports or alerts to explain the nature of the threat, including how an attacker might exploit it.
  - Generates a detailed mitigation plan with step-by-step remediation actions, ensuring that security experts can review and refine the logic before technical implementation.
- **Cyber Security Software Architect,** also powered by a textual LLM:
  - Takes the mitigation plan and cross-references it with an organization's security catalogue to select the most appropriate tools for each remediation step.
  - Produces a technically actionable playbook draft (not CACAO yet), including execution instructions, success criteria, and justifications for each tool's use.
- **OpenC2 and CACAO Expert**, powered by a code-focused LLM like Mistral Devstral:
  - Converts the actionable playbook into structured OpenC2 commands and assembles them into a CACAO 2.0-compliant playbook.
  - Validates all outputs against the official OpenC2 and CACAO JSON schemas, ensuring strict compliance before deployment.

Table 2-3 Crew Agents description and role

Agent	LLM	Role	Outputs

Cyber Threat Intelligence (CTI) Analyst	Textual/Contextual LLM (e.g., Mistral Large)	Analyses threats, explains exploitability, and designs high-level mitigation plans.	Threat explanation, mitigation plan
Cyber Security Software Architect	Textual/Contextual LLM (e.g., Mistral Large)	Maps mitigation steps to real security tools, ensuring technical feasibility.	Tool-mapped actionable playbook draft
OpenC2 & CACAO Expert	Code-Generation LLM (e.g., Mistral Devstral)	Converts playbook drafts into **OpenC2 commands** and **CACAO 2.0 playbooks**, validates compliance.	Validated CACAO playbook with OpenC2 commands

This division of tasks, using multiple agents and different LLMs for each purpose ensures that each agent operates within its domain of expertise, reducing errors and improving the precision of the final playbook. For example, a textual LLM excels at understanding threat reports and crafting mitigation strategies, while a code-specialized LLM is better suited for generating syntactically correct CACAO playbooks and OpenC2 commands.

As an experiment, we took the CVE-2023-49147 text content as the input for the two approaches. This content is available in [CVE202349147].

The first approach resulted in a very simple yet incorrect JSON CACAO playbook, depicted in Figure 2-4 in which we had to fix syntax, formatting and compliance mistakes manually. Instead, by using the proposed revision, the playbook resulted in a correct but very minimalist approach for mitigation, ensuring the verification of the corrupted executable, its update towards a non-vulnerable version of the same software and finally the verification of the new installed version:

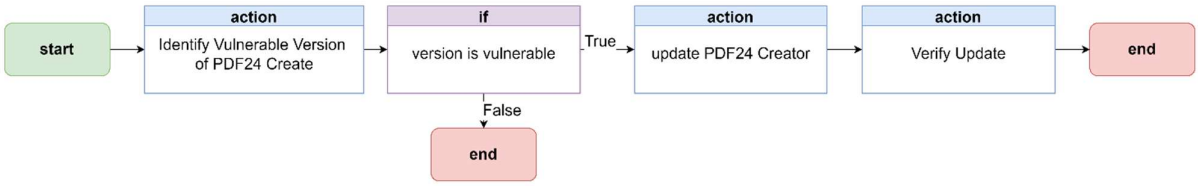


Figure 2-4 CACAO Playbook generated v1

The solution generated by the first approach's design includes many gaps to ensure a satisfying security coverage against the CVE: no process termination against the vulnerable software, no isolation action is proposed against potential infected hosts, no inspection of attack traces, no action against attack persistence methods and finally no reporting action is proposed for further organizations processes.

Finally, the playbook was not including:

- security tools selected from an organization's catalogue to enforce the actions;
- OpenC2 commands for the interoperability of remediation actions.

The new implemented approach has been tested using the same CVE as an input to the crew. The resulting playbook is better in every aspect, as depicted in Figure 2-5.

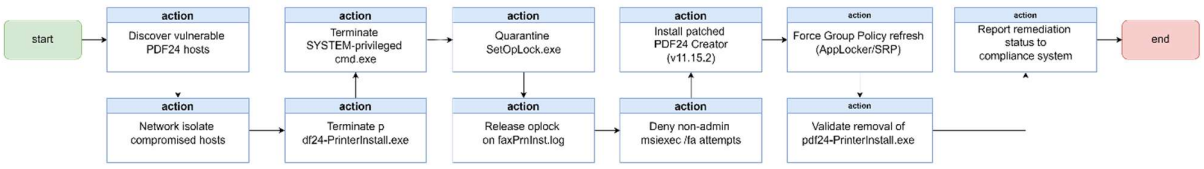


Figure 2-5 CACAO Playbook generated v2

In this new playbook, the security remediation and coverage were significantly more complete, including isolation actions, termination processes, update and validation of the vulnerable software, actions against attack persistence, reporting and even a quarantine action for further vulnerable executable inspections. Finally, the

playbook was immediately compliant with the CACAO language specification, and all actions were formatted using the OpenC2 language.

2.1.2.1 Agents and task explanation

As depicted in Figure 2-6, the GenAI4SOAR system is composed of agents communicating sequentially together, meaning that each result from an agent's task is forwarded to the next one, to finally generate CACAO playbooks, as:

- **Task 1** consists of explaining the threat and is executed by the agent **Cyber Threat Intelligence (CTI) Analyst**. It receives as input a security alert or a security policy and Contextual data (e.g., infrastructure details, existing security policies, past incidents). The CTI Analyst parses its content and generates a structured threat explanation, including Nature of the threat, potential attack vectors and impact assessment.

- **Task 2** consists of crafting a step-by-step mitigation strategy in human-readable format and is executed by the agent **Cyber Threat Intelligence (CTI) Analyst**, ensuring clarity for security teams. Each step includes an objective, a rationale and high-level actions.

- **Task 3** consists of selecting appropriate security tools from a catalogue and is executed by the **Cyber Security Software Architect Agent.** The task cross-references each mitigation step from the mitigation plan in input with the tools within the catalogue to identify the best-fit tools.

- **Task 4** consists of designing OpenC2 commands and is executed by the **OpenC2 CACAO specialist agent.** It takes the selection of tools from task 3 in inputs and converts each tool-specific action into structured OpenC2 commands in JSON format.

- **Task 5** validates and fix the OpenC2 commands, generated in the task 4, against the OpenC2 language specification using JSON schemas, and is executed by the **OpenC2 CACAO expert agent.** Executing this validation in a dedicated task, separated from the generation one, limits the context for validation only, which increases the efficiency of the execution and the accuracy of the results.

- **Task 6** consists of designing a CACAO playbook and is executed by the **OpenC2 CACAO expert agent.** This task embeds the validated OpenC2 commands from the previous task and analyses the mitigation plan generated in task 2 to create a CACAO mitigation playbook, embedding the OpenC2 commands into the workflow steps.

- **Task 7** validates and fixes the CACAO playbook, generated in the previous task, against the CACAO v0.2 language specification using JSON schemas, and is executed by the **OpenC2 CACAO expert agent.** Executing this validation in a dedicated task, separated from the generation one, limits the context for validation only, which increases the efficiency of the execution and the accuracy of the results.

A final algorithm step is processed over the CACAO playbook to encode the embedded OpenC2 commands in *base64* to follow the recommendation of the CACAO standard's extension for OpenC2 commands. All the tasks and agent's configuration are provided in [ZenGAI426].

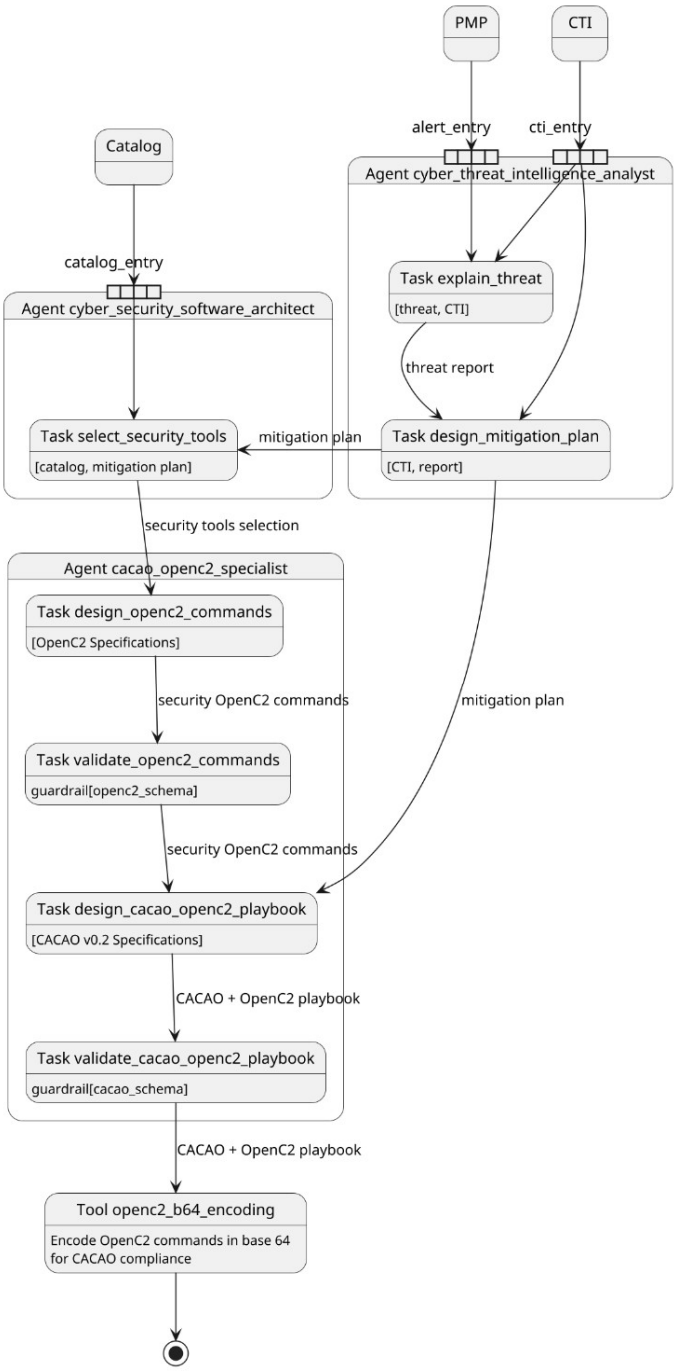


Figure 2-6 GenAI4SOAR Agents Architecture

2.1.3 Security Closed Loop Management (S-CL Mgmt)

The main functionalities of the Security Closed Loop Management (S-CL Mgmt), already defined and described in detail in D4.3 [R6G25-D43], are reported in Table Table 2-4 for reference. No update to the S-CL Mgmt functionalities has been introduced in D4.4: the component was already at a mature design and implementation stage as reported in D4.3, and D4.3 remains the reference document for its functional specification.

Table 2-4 List of S-CL Mgmt Functionalities

Name	Description	Updated (YES/NO/NEW)
Closed-Loop Lifecycle Management (LCM)	Enables the management of the ontology schema (TBox), allowing uploading, retrieving, and clearing the ontology definition.	NO
Closed-Loop Instance Management	Enables the management of instance-level knowledge (ABox) within the Security Orchestrator knowledge graph.	NO
Closed-Loop Function Management	Enables the management of Security Functions, allowing the onboarding, querying, and deletion of Security Functions (SFs).	NO
Error & Status Reporting	Provides access to semantically enriched data models, exposing structured representations of security functions, capabilities, metrics, activities, and target environments.	NO

2.1.3.1 Design updates and Prototype Implementation

The S-CL Mgmt component was not subject to any design or implementation update in D4.4. Its architecture and prototype implementation follow the Closed Loop LCM component developed in the HEXA-X-II project and described in [HXII-D63], as reported in D4.3. The complete APIs are available at [Nex25c] and [Nex25d] and remains unchanged. D4.3 [R6G25-D43] is the reference for the full description of this component.

2.1.4 Security Resource Orchestrator (S-RO)

The main functionalities of the Security Resource Orchestrator (S-RO), already defined and described in detail in D4.3 [R6G25-D43], are reported in Table 2-5 for reference. No update to the S-RO functionalities has been introduced in D4.4: the component was already at a mature design and implementation stage as reported in D4.3, and D4.3 remains the reference document for its functional specification.

Table 2-5 List of S-RO functionalities

Name	Description	Updated (YES/NO/NEW)
Platform Management	Manages platform information and controls Security Applications. Includes listing, adding, and removing platform entries; retrieving platform details; and deploying, updating, or deleting applications.	NO
Service Lifecycle Management	Provides lifecycle operations for Security Services, including deployment, update, and deletion.	NO
Service Template Catalogue	Manages Service Templates, including listing, adding, removing, fetching, and downloading template artefacts.	NO
Resource Management	Maintains a registry of infrastructure resources, supporting inventory management across multi-cluster cloud-edge environments.	NO

2.1.4.1 Design updates and Prototype Implementation

The S-RO component was not subject to any design or implementation update in D4.4. Its architecture and prototype implementation follow the multi-cluster extreme-edge resource orchestration component developed

in the HEXA-X-II project and described in [HXII-D63], as reported in D4.3. The complete API surface is available at [Nex25e] and remains unchanged. D4.3 [R6G25-D43] is the reference for the full description of this component.

2.1.5 Risk-averse Resource Management Framework

The main functionalities of the Risk-averse Resource Management Framework (RA-RRM), already defined and described in detail in D4.3 [R6G25-D43], are reported in Table 2-6 for reference. No updates to the RA-RRM functionalities were introduced in D4.4. The resource management modelling and the algorithmic components for resource optimization had already reached a mature design stage, as reported in D4.3, and D4.3 remains the reference document for the functional specification. During the period between D4.3 and D4.4, additional simulations and experiments were conducted, including numerical validation, benchmark comparison, convergence and complexity assessment, and mapping to project-specific use cases.

Table 2-6 List of RA-RRM functionalities

Name	Description	Updated (YES/NO/NEW)
Resource Allocation Optimization	Optimises the allocation of resources (e.g., bandwidth, power) across multiple users under uncertainty by solving a risk-aware resource allocation problem.	NO
Heterogeneous User Requirements Support	Supports multiple users with heterogeneous objectives and application-specific requirements, including target service levels, QoS, or QoE constraints.	NO
CPT-based Utility Modelling	Models user preferences through Cumulative Prospect Theory (CPT), capturing risk aversion/risk seeking, reference dependence, loss-gain sensitivity, and subjective probability perception.	NO
User Risk Profile Modelling	Accounts for different user risk attitudes, including risk-averse, risk-neutral, and risk-seeking behaviours, through the parameters and shape of the corresponding utility functions.	NO
Reference-point-aware Decision Modelling	Incorporates a reference point representing a baseline or expected operating level, enabling the framework to evaluate gains and losses relative to application-specific targets.	NO
CPT-based Optimization Engine	Processes the framework inputs and formulates the underlying optimization problem to compute resource allocations that satisfy service objectives while accounting for behavioural patterns.	NO
Scalable Optimization Pipeline	Applies a combined optimization strategy based on Successive Convex Approximation (SCA), Lagrange Relaxation (LR), and Projected Subgradient Method (PSM) to address non-convexity, constraints, and non-smoothness.	NO
Per-user Resource Allocation Output	Produces the recommended resource allocation for each user as output of the optimization process.	NO
S-RO Integration Support	Supports the communication of the recommended resource allocation to the S-RO through an API for potential integration into broader system-level decision-making.	NO

2.1.6 Trustworthy AI for Intent-Driven RAN

We have developed mechanisms for assessing the trustworthiness of the resource orchestrator, in which resource optimization and slice negotiation are performed by LLM-based agents. Because multiple agents are involved in negotiating slice intents and resource allocations, ensuring reliable and aligned agent behaviour is

critical. To address this challenge, we introduce a rule-based trust scoring framework that evaluates agent reliability along two key dimensions: (i) the alignment of an agent's decisions with mediator expectations and underlying network policies, and (ii) the behavioural coherence of the agent's communications during negotiation.

The *trust score* is computed as a weighted sum of two components, satisfaction and coherence, as follows:

$$T = w_S S + w_C C$$

where S denotes the satisfaction score, which measures the alignment between the agents' decisions and those of the central mediator, while C represents the coherence score, which evaluates the quality and consistency of agent's communications. In our implementation, we set $w_s = 0.15$ and $w_c = 0.85$, placing greater emphasis on communication quality as a determinant of trust.

The *satisfaction* component captures how well an agent's decision aligns with the expected negotiation outcome (offer validity, intent alignment, and mediator agreement), and is explicitly modeled as a deviation-based metric:

$$S = 1 - (w_o D_o + w_i D_i + w_m D_m)$$

where w_o, w_i , and w_m are the weights assigned to deviations from valid offers, intent, and the mediator's recommendations and are set for equal importance to each component.

The *coherence* component evaluates how the agent reasons and integrates three core behavioral dimensions: factual accuracy, logical consistency and semantic coherence:

$$C = w_f F + w_l L + w_e E$$

where w_f, w_l and w_e are the weights assigned the the factual accuracy F, the logical consistency L and the semantic coherence E, respectively. Given the importance of factual correctness in negotiation scenarios, we prioritize it by setting $w_f = 0.8$, while allocating smaller weights to the other dimensions (0.1 and 0.1) This weighting scheme emphasizes truthfulness while still accounting for reasoning quality and communicative coherence.

Table 2-7 Trust score comparison of LLM models in multi-agent negotiations

Model	Satisfaction	Coherence	Trust Score (0 – 5)
gpt-4.1	3.88 ± 0.75	5.00 ± 0.00	4.83 ± 0.10
gpt-4.1-mini	3.88 ± 0.75	4.96 ± 0.03	4.81 ± 0.10
Llama-3.1-8B-instruct-FT	4.44 ± 0.75	3.86 ± 0.40	3.94 ± 0.25
Llama-3.1-8B-instruct	5.00 ± 0.00	3.73 ± 1.65	3.91 ± 1.35
Llama-3.2-3B-instruct-FT	3.89 ± 0.75	2.23 ± 1.40	2.48 ± 1.10
Llama-3.2-3B-instruct	4.43 ± 0.78	2.01 ± 1.55	2.36 ± 1.20
Llama-3.2-1B-instruct	3.33 ± 0.00	2.06 ± 1.51	2.25 ± 1.25
Llama-3.2-1B-instruct-FT	5.00 ± 0.00	1.53 ± 0.77	2.05 ± 0.64

The results clearly demonstrate that decision alignment alone is insufficient to guarantee trustworthiness. Several smaller models achieve high satisfaction scores, indicating that they select valid and goal-aligned solutions. However, their low coherence scores significantly reduce their overall Trust Score, highlighting that

these models fail to produce consistent and factually reliable reasoning. In contrast, larger models such as GPT-4.1 achieve high Trust Scores primarily due to near-perfect coherence, confirming that trust in multi-agent systems is dominated by reasoning quality rather than decision correctness alone. This validates the design of the Trust Score framework, where behavioral consistency is prioritized as the key determinant of reliable agent behavior. The framework has been evaluated on an OpenAirInterface and FlexRIC testbed, where it demonstrates significant system level improvements, including +37% aggregate throughput, −73% URLLC latency, and 8% PRB savings compared to a static SLA-based baseline, in which slice resources are pre-allocated and remain fixed across network phases, without adaptive negotiation or dynamic reallocation mechanisms.

2.2 Continuous Monitoring and Threat Detection

This section describes the updates and new features relating to the continuous monitoring provided by the Programmable Monitoring Platform, the anomaly detection module with semantic recognition, and rule-based threat detection. The following subsections cover improvements relating to usability, optimisation, and compatibility with other components.

2.2.1 Programmable Monitoring Platform

2.2.1.1 Main functionalities

The PMP [PMP25a] maintains the same functionalities described in the last deliverable 4.3 [R6G25-D43]. The development is focused on the modularity and the dynamism of the configurations. Some of the modules have been updated as shown in Table 2-8, improving their performance in terms of the rate of results, including more tools, or simply enhancing the communication and reaction time. In addition, a new module has been developed to analyse network traffic resulting in behavioural flows.

Table 2-8 List of PMP Functionalities

Name	Description	Updated (YES/NO/NEW)
Data Aggregation and Normalisation Module	Aggregate the numerical data and normalise the rest of the logs to aggregate them using “Machine ID” as the unified key.	YES
Databases	Compendium of databases to store long-term data, medium-term data, raw streaming data and configurations.	YES
Configuration Manager API	The API REST to configure, update or delete tool configurations. It also manages endpoints for interacting with ThingsBoard platform for information gathering.	YES
Flow Module	Analyse the network information to determine the relation between the traffic flows and share the information to the Communication Bus.	NEW

The update of the Data Aggregation and Normalisation Module relates to the use of the data collected, due to the data sources provide numerical information such as health metrics (CPU, RAM, disk, etc.) and logs statistics, and, on the other hand raw logs with different non-standardised format. For this reason, data processing had to be different in each case, but it requires a common parameter as the Machine ID (MID) [Mid25] to associate the numerical information with the processed logs. The MID is cited in deliverable 4.3 as a hash identifier associated with a device that uses the most static hardware and software data.

Databases are not a module as such, but they have been modified to include different types, not just the long-term data storage or the configuration data storage. Two new types of data storage have been introduced. The first one is linked with a new module called the Flow Module, as well as the Alert and Notification Module. It is used to store the results of the analysed network flows and security alerts. The second database implementation is a Real-Time DataBase (RTDB) that acts as a cache storage to efficiently expose data from the Communication Bus to the Near Real-Time Data Retrieval API.

The Configuration Manager API has also been updated to manage its relationship with the ThingsBoard service. New endpoints have been created to handle different possibilities, such as continuously monitoring one or a list of ThingsBoard devices, stopping the monitoring of one or a list of them, stopping all the monitoring devices without specifying further details, and requesting information about the status of each of the monitored devices.

Finally, a new module has been developed as mentioned in the previous paragraphs. The Flow Module consumes raw data to analyse the relationship between the network traces, combining an entire data stream into a single summary between the source and the destination of that transaction on the network. The results are sent to the Communication Bus as well as to a database to provide a medium-term history. Both channels will be available for consumption by external components.

2.2.1.2 Design updates

A new design with some changes is shown in Figure 2-7. It is based on the last deliverable 4.3, but adding new capabilities such as the Flow Module, the Medium-term, and Real-Time Data Storage. The Data Aggregation and Normalisation Module evolved to include Logstash [Log26], in order to normalise raw logs to the Elastic Common Schema (ECS) format, and OpenSearch [Ope26], to aggregate the normalised logs. MongoDB [Mon26] is also implemented as a Medium-term Data Storage to save security alerts from the Snort3 IDS [Sno25] and the analysed flows from CICFlowMeter [AppCic]. Redis DB [Red26] is used as a Real-Time Data Storage to acts like an intermediary between the Communication Bus and the Near Real-time Data Retrieval API. Finally, the Flow Module is a new acquisition, although it is considered an "external module" outside of the core of the PMP due to its analytical approach. JSON2PCAP [Jso24] translates network traffic into the correct format for CICFlowMeter to use as a flow analyser.

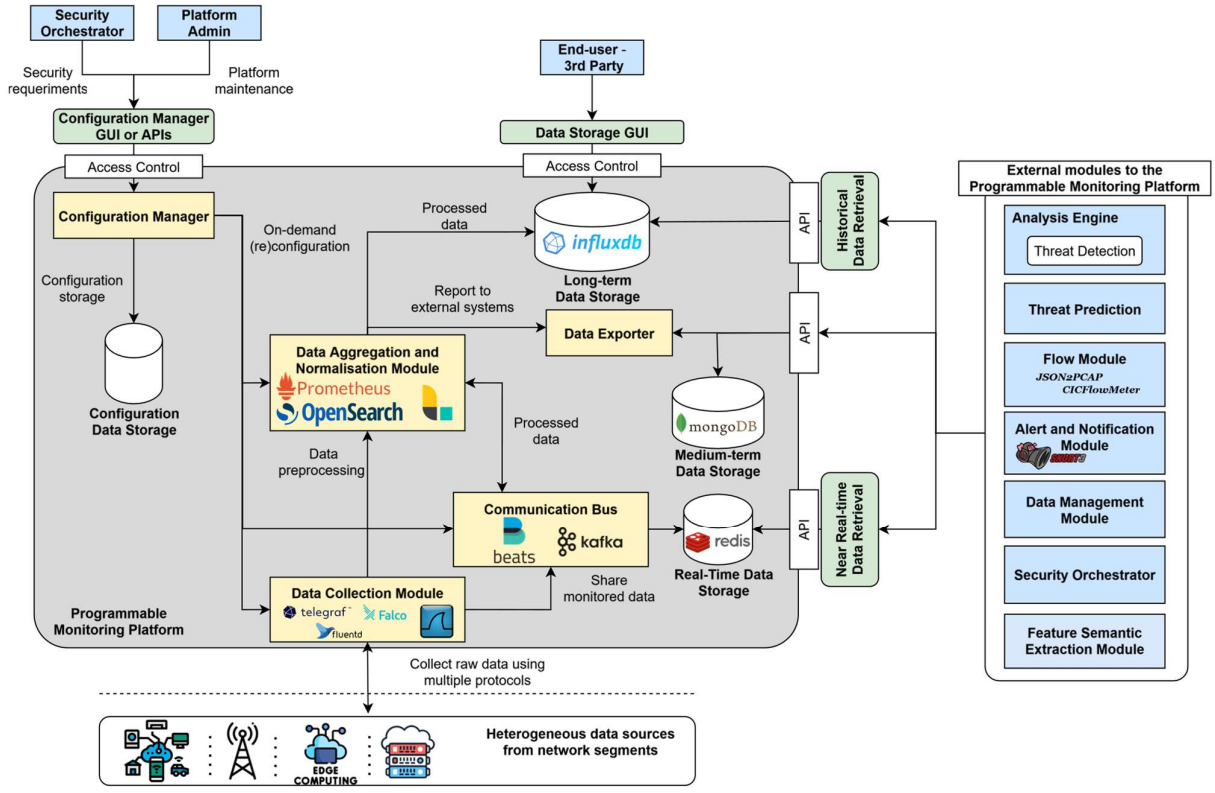


Figure 2-7 Programmable Monitoring Platform design

2.2.1.3 Prototype Implementation

The implementation of PMP includes multiple modifications designed to improve its performance, which involves the addition of new modules and the updating of existing ones. Table 2-9 shows the updated and new modules. In some cases, certain basic settings have been improved, but in others, the way the modules are managed has been completely modified.

Table 2-9 List of PMP implementations

Name	Description	Updated (YES/NO/NEW)
Configuration Module	(Re)configure the tools deployed in each module.	Yes
Data Collection Module	Extracts information from data sources and their third layers (infrastructure, service and network).	Yes
Communication Bus	Transmit information extracted from the Data Collection Module to internal modules and the Near Real-time Data Retrieval API to expose it to external modules.	Yes
Data Aggregation and Normalisation Module	Aggregate the numerical data and normalise the rest of the logs to aggregate them using "machine_id" as the unified key.	New
Data Exporter	Intermediary module to export the processed data	Yes
Databases	Compendium of databases to store long-term data, medium-term data, raw streaming data and configurations.	New
Data Storage GUI	Representation of the data processed via GUI.	Yes
Configuration Manager API	The API REST to configure, update or delete tool configurations. It also manages endpoints for interacting with ThingsBoard platform for information gathering.	Yes
Near Real-time Data Retrieval API	An API REST to expose raw data directly from the Communication Bus.	Yes
Historical Data Retrieval API	An API REST to expose historical data using the "machine_id" identifier.	Yes
Flow Module	Analyse the network information to determine the flows of the traffic and share the information to the Communication Bus.	New

The Configuration Module is designed to comply with the new endpoints of the API. The endpoints have been modified to include Sigma Rules [Sig25] or configurations in JSON. The JSON is directly related to the environment variables used in the tool's configurations.

The Data Collection Module has undergone important changes relating to the shared format of the network traces, changing from a line of a packet in JSON to an entire packet in NDJSON. This modification contributes to a significant improvement in the Communication Bus buffer, as a network packet previously required around 800 JSON lines, but using NDJSON, the entire packet can be compressed into just a single line. It also optimises data consumption by preventing bottlenecks in Kafka [Kaf25] topics. Furthermore, the module has been updated to transmit the data to the Data Aggregation and Normalisation Module using endpoints, as Prometheus [Pro26] needs to collect information directly from the sources. Fluentd [Flu25], Telegraf [Tel25] and Falco [Fal25] tools are responsible for exposing the data in a specific format so that Prometheus can understand it.

The Communication Bus has been improved through the use of compression in Filebeat [Fil25], applying a maximum data retention of 1 day or 1 GiB. On the other hand, in Kafka, the rotation of information into the topics has been modified by size with 256MiB and by time with 1 hour.

The Data Aggregation and Normalisation Module has been developed using Prometheus to scrape the information from the Data Collection Module tools, aggregating numerical information using the MID label as a unified point. The aggregated information pre-processed by Prometheus is written remotely to InfluxDB using down-sampling techniques to condense the data that will be queried and stored efficiently for over a year. It is necessary to create a Telegraf instance to perform remote writing to InfluxDB [Inf26]. This is due to the technical requirements of the new version of InfluxDB that are unrelated to the development of the PMP. This module also covers the normalisation part using Logstash and ECS. The Communication Bus information is normalised by Logstash and then used in OpenSearch to aggregate the logs, also using the MID label now called "host.id" due to issues with the nomenclature of internal tags.

Data Exporter provides access to the Medium-term Data Storage as an intermediary between the API and MongoDB, the tool that represents the storage. When an external component requests information from the Data Exporter API within a specified time period, the Data Exporter module acts as a worker that gathers the information from MongoDB, filters it, and responds to the API request to ensure the MVC pattern.

A revision of databases has been done to include storages for each type of data saved and shared. MongoDB is selected as a Medium-term database to store alerts from Snort3 and flows from CICFlowMeter. Its NoSQL distribution allows data to be saved in BSON, a binary representation of JSON. Snort3 generates the alerts in JSON and CICFlowMeter in CSV, but both can be stored without the need to parse their formats thanks to MongoDB. A new database has been included to share raw data from the Communication Bus to the Near Real-time Data Retrieval API. Kafka provides topics with information, but the exposition through a REST API generates delay, high load, offset problems and bad scalability. In contrast, Redis DB acts as a Real-time database, as a buffer cache in other words. Redis DB uses the same share system than Kafka (publisher/subscriber) and can be connected to Kafka easily to unify the last information provided by the Data Collection Module, serving it through one or multiple websocket associated with an API REST endpoint. Another advantage is that if the node managing Kafka and therefore the associated topics goes down, Redis DB will cache the latest information, ensuring it is always active in case an external component required it. Once the Kafka node is up and running, it will show the updated information again. Moreover, the "redis streams" data structure saves processing load by moving data as bytes instead of loading the data into memory to be transferred from Kafka to the external components and even has data compression to be more space efficient.

Grafana [Gra26] performs the work of the Data Storage GUI. It has been linked to InfluxDB to use the aggregated numerical data from Prometheus, and to OpenSearch to use the normalised log information. MID acts as the join identifier to manage the information from both sources.

Configuration Manager API has been updated to incorporate the endpoints associated with ThingsBoard as shown from Table 2-10 to Table 2-13, and also in the official swagger [PMP25b]. This REST API is requested by ZTSO to collect information from Thingsboard devices. The request is received and managed by the PMP backend in accordance with the Model-View-Controller (MVC) pattern. The backend is responsible for performing validations at the parameters and identification levels. When a device is requested, one thread is created to manage continuous monitoring until the ZTSO sends another request to stop it or even stop all current devices being monitored. The output of Thingsboard devices contains alarms that are transmitted to de Communication Bus generating a Kafka topic per each device with "thingsboard_entityID" nomenclature. The topics can only be created once, so if a device is monitored, stopped and supervised again, it will always use the same topic. The monitoring process concludes with the exposure of alarms to external components through the Near Real-time Data Retrieval API.

Table 2-10 Configuration Manager API - collectDataFromThingsboard operation

Operation name: *ConfigurationManagerThingsboard/collectDataFromThingsboard*		
Description	This endpoint allows the ZTSO to request the PMP to trigger data collection from Thingsboard.	
Input Parameters	**Type**	**Description**
thingsboardsIP	String	IP address of the Thingsboard instance from which data will be collected
thingsboardPort	Integer	Port number of the Thingsboard instance
entityId	String/Array of Strings	A string or an array of string representing the entity id. For example, "784f394c-42b6-435a-983c-b7beff2784f9" or "["71c0e3e0-a8e1-11f0-a091-2d2de22ced6c", "'784f394c-42b6-435a-983c-b7beff2784f9' ", "def456-789-012..."]".
entityType	String	A string value representing the entity type. For example, "DEVICE".
Output Parameters	**Type**	**Description**
200	String	OK with monitoring details.
400	String	Bad request if validation fails.

	401	String	Unauthorized if ThingsBoard authentication fails.
	404	String	Not found if device doesn't exist in ThingsBoard.
	429	String	Too many requests if device limit reached.
	500	String	Internal server error for unexpected errors.
Notes			

Table 2-11 Configuration Manager API - stopMonitoring operation

Operation name: *ConfigurationManagerThingsboard/stopMonitoring*		
Description	This endpoint allows the ZTSO to request the PMP to stop data collection from Thingsboard.	
Input Parameters	**Type**	**Description**
entityId	String/Array of Strings	A string or an array of string representing the entity id. For example, “784f394c-42b6-435a-983c-b7beff2784f9” or “["71c0e3e0-a8e1-11f0-a091-2d2de22ced6c", "'784f394c-42b6-435a-983c-b7beff2784f9' ", "def456-789-012..."]”.
Output Parameters	**Type**	**Description**
200	String	OK with results for each device.
400	String	Bad request if payload is invalid.
Notes		

Table 2-12 Configuration Manager API - stopAllMonitoring operation

Operation name: *ConfigurationManagerThingsboard/stopAllMonitoring*		
Description	This endpoint allows the ZTSO to request the PMP to stop all data collection from Thingsboard.	
Input Parameters	**Type**	**Description**
Confirm	Boolean	A Boolean value to indicate the suspension of all monitored devices. For example, “true”.
Output Parameters	**Type**	**Description**
200	String	OK with results for all stopped devices.
400	String	Bad request if confirmation missing or invalid.
Notes		
The “Confirm” parameter is case sensitive, do not use “True”, “TRUE” or another data type such as string or array of strings.		

Table 2-13 Configuration Manager API - monitoringStatus operation

Operation name: *ConfigurationManagerThingsboard/monitoringStatus*		
Description	This endpoint allows the ZTSO to request the PMP to get status of all monitored devices from Thingsboard.	
Output Parameters	**Type**	**Description**
200	String	OK with list of monitored devices and their status.
Notes		

Historical Data Retrieval API shares the information from the InfluxDB Long-term Data Storage through a REST API. External modules request information about one type of data from a specific user using their MID and a period of time. In this case, the API does not use websockets to transmit information continuously but operates as a polling system.

Furthermore, a new Flow Module has been implemented to analyse the network traffic using CICFlowMeter, which provides the relationship between devices and the network. These metrics can be used by external modules to determine the network behaviour. The module constantly consumes network traces in NDJSON format, which are translated into PCAP format using the JSON2PCAP tool. CICFlowMeter does not allow traffic to be fed in via direct streaming through virtual network interfaces, so this situation has had to be controlled using mini-batches. These mini batches are limited to storing information up to 100KiB or half a second, depending on which comes first. This creates a fast flow of information almost similar to streaming. The resulting flow metrics are stored in a Medium-term Data Storage to keep a record of them. In addition, the metrics are sent to the Communication Bus to be accessible by external modules through the Near Real-time Data Retrieval API.

2.2.2 Semantic-aware Anomaly Detection

This module maintains the core functionalities described in D4.3 and extends them with two new components. The module continues to integrate semantics-aware metrics and data-driven anomaly detection into the AI/ML-driven zero-touch security framework. Table 2-14 summarizes the functionalities.

Table 2-14 List of semantic-aware detection functionalities

Name	Description	Updated (YES/NO/NEW)
Temporal Pattern Recognition with Unsupervised Learning (RANGAN)	GAN-based anomaly detector combined with a transformer architecture for unsupervised detection in multivariate RAN performance data.	No (D4.3 Subsection 2.2.2)
Semantic-Driven Error Quantification and Pareto-Optimal Communication	Quantifies the value of communication for remote estimation using semantics-aware metrics. An efficient algorithm constructs the complete Pareto frontier characterizing optimal trade-offs between estimation performance and communication cost.	Yes (D4.3 Subsection 2.2.2)
Content-Aware Remote Estimation with Prioritized States (AoMA/AoFA)	Introduces Age of Missed Alarm (AoMA) and Age of False Alarm (AoFA) metrics to capture the lasting impacts of different estimation error types in systems with prioritized alarm and normal states. An efficient switching policy is derived via MDP formulation.	New
Error-Aware Joint Sampling for Correlated Multi-Source Monitoring	Proposes an error-aware joint sampling and transmission policy for real-time tracking of two correlated stochastic processes over a shared wireless channel. Closed-form expressions for time-averaged reconstruction error are derived, and an optimization problem minimizing error under sampling cost constraints is solved.	New

The updated and new functionalities are described below:

2.2.2.1 Semantic-Driven Error Quantification and Pareto-Optimal Communication (Insec-SPI / SPLIT) (update)

Building on the prior Insec-SPI framework reported in D4.3, the value-of-communication analysis has been substantially extended. The updated framework introduces a Pareto analysis that characterizes the complete set of policies achieving optimal trade-offs between estimation performance and communication cost. The key structural result is that the Pareto frontier is strictly decreasing, convex, and piecewise linear, and each Pareto-optimal operating point is realizable as a convex combination of two stationary deterministic policies.

Leveraging these properties, an efficient and provably optimal algorithm called SPLIT (Structured Pareto Line Intersection Tracing) has been developed to construct the exact frontier, with time complexity linear in the number of corner points. This provides system designers with a complete lookup table of optimal policies for any communication budget, enabling adaptive scheduling for anomaly alert generation.

2.2.2.2 Content-Aware Remote Estimation with AoMA/AoFA (new)

A new component addresses the limitation of prior semantics-aware metrics that treat all source states equally. In security monitoring contexts, erroneously announcing a normal state when the system is in an alarm state (missed alarm) is significantly more costly than a false alarm. Two new interdependent age metrics are introduced: the Age of Missed Alarm (AoMA) and the Age of False Alarm (AoFA), which capture the lasting impacts of different types of estimation errors. The problem is formulated as a countably infinite-state Markov Decision Process (MDP) with average cost criterion. The optimal policy is shown to have a switching structure with distinct thresholds for each age process. Closed-form performance expressions are derived, and an efficient search algorithm is proposed that avoids the hurdles of classical dynamic programming. For numerical tractability, a finite-state approximate MDP is introduced with provably exponential convergence to the original problem. This component directly supports content-aware anomaly alert prioritization within the platform.

2.2.2.3 Error-Aware Joint Sampling for Correlated Sources (new)

This component extends the semantic-aware framework to multi-source monitoring scenarios where multiple network processes are correlated. The joint evolution of the processes is modeled as a two-dimensional discrete-time Markov chain. An error-aware joint sampling and transmission policy is proposed, where each sampler probabilistically generates samples only when the current process state differs from the most recently reconstructed state at the corresponding monitor. The analysis explicitly leverages inter-source correlation to quantify the marginal value of updates in closed form. An optimization problem minimizing the time-averaged reconstruction error subject to average sampling cost constraints is formulated and solved. This component is relevant to scenarios where multiple network features (e.g., different KPIs from the same RAN segment) are monitored simultaneously and their statistical dependencies are to be exploited for resource-efficient anomaly detection.

2.2.2.4 Design updates

The module design has been extended along two dimensions compared to D4.3. First, the Insec-SPI component has evolved from a single-point Lagrangian optimization to a complete Pareto frontier characterization using the SPLIT algorithm. This provides a richer set of operating points for the anomaly alert scheduling mechanism, allowing dynamic adaptation to varying communication budgets without re-optimization. Second, two new theoretical components have been introduced: (i) the AoMA/AoFA framework that enables content-aware alert prioritization distinguishing between missed alarms and false alarms, and (ii) the error-aware multi-source sampling framework that extends the semantic-aware approach to correlated monitoring scenarios. These extensions enhance the module's ability to support context-dependent and resource-efficient anomaly signalling within the ROBUST-6G security closed loops.

This module is derived from theoretical research and is not intended for direct prototype. As such, the implementation does not involve a defined infrastructure or communication architecture.

2.2.3 Rule-based Threat Detection

2.2.3.1 Main functionalities

The main functionalities of the Alert and Notification Module are the same as those described in deliverable 4.3. The only new feature affects the exposure of the alert notification, as shown in Table 2-15, adding not only the exchange of the alert via Communication Bus, but also the saving of a history in the Medium-term Data Storage located in the PMP system.

Table 2-15 List of Rule-based Threat Detection Functionalities

Name	Description	Updated (YES/NO/NEW)
Alert and Notification Module	Analyses the network traffic using an IDS and notify the security alerts generated.	Yes

2.2.3.2 Design updates

The internal design has been updated to enhance the transmission and performance of the module. The following Figure 2-8 shows the new design using the Medium-term Data Storage and the Near Real-time Data Retrieval to extract the network data. In addition, Filebeat has been removed from the latest design and replaced with a Python script to consume streaming network traffic while sending alerts created by the Analyser Module to the Communications Bus and the Medium-term Data Storage through the Notification Module.

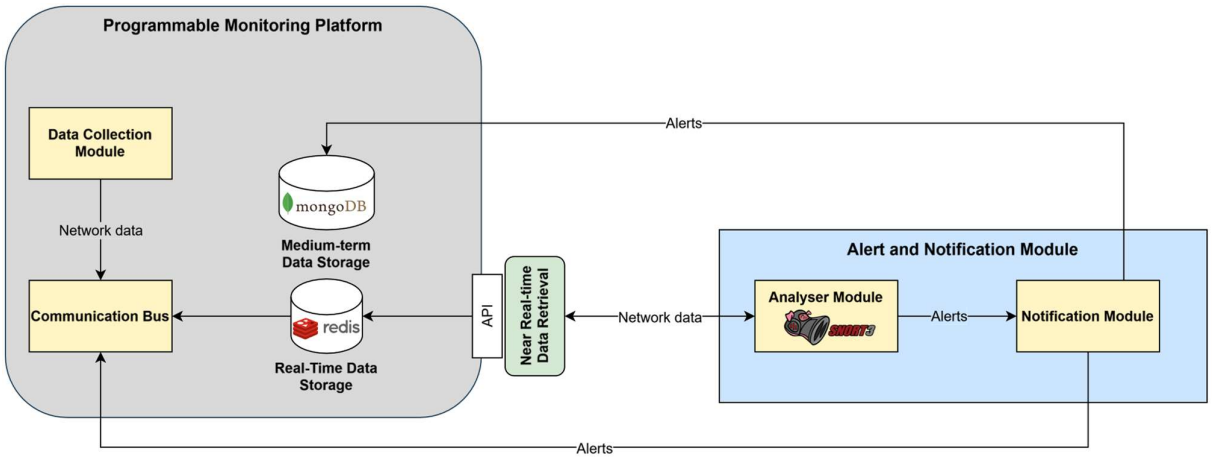


Figure 2-8 Alert and Notification Module design

2.2.3.3 Prototype implementation

This module has been improved by removing the batch processing system, which generated a long response time. The main problem was that the file created in PCAP format after JSON2PCAP translation took a long time to complete, due to the fact that Snort3 could not analyse it until the file descriptor was closed. The solution to this problem was to eliminate the batch processing system and replace it with a continuous streaming one. The network traces collected in NDJSON format are routed to a virtual TUN/TAP interface so that Snort3 reads them in real time from an interface rather than from a file. Thanks to this new method of information transmission, Snort3 can perform its IDS function continuously as if it were installed on the network device itself, generating alerts very quickly. Finally, when an alert is generated, the backend detects and sends it to the Communication Bus and to the Medium-term Data Storage for exposure through the APIs.

2.3 Incident Prediction and Continuous Mitigation

Threat mitigation and prediction are core components of a security orchestrator designed to address cyberattacks on networked systems. Telemetry and network traffic data captured by the Programmable Monitoring Platform (PMP) are first stored within the data fabric. These data are subsequently retrieved and analysed by the respective modules to generate real-time reports and actionable insights for the security orchestrator, as illustrated in Figure 2-9. The functionalities of this Subsection are summarised in Table

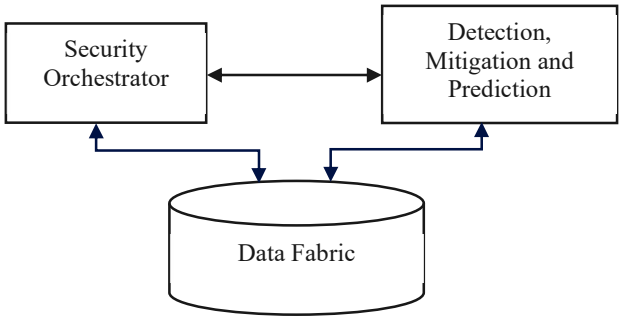


Figure 2-9 Architecture of Threat Prediction and Mitigation

The threat detection, mitigation, and prediction modules are summarized in the next Table 2-16. Specifically, `network_attack_detection()` and `ciot_attack_detection()` are implemented for threat detection; `network_attack_mitigation()` and `ciot_attack_mitigation()` are responsible for mitigation; and `network_attack_prediction()` together with `<iotdev>_attack_prediction()` address threat prediction tasks. A comprehensive description of these modules has been provided in Subsection 2.1.9 of Deliverable 4.3 and is therefore not reiterated here.

Table 2-16 Development of Network Attack Detection, Mitigation and Prediction Modules

Name	Description	Updated (YES/NO/NEW)
`network_attack_detection()`	Detection model built based on CSE-CIC-IDS-2018 Dataset [Ima18]	No (D4.3 Subsection 2.1.9)
`cictoniot_network_attack_detection()`	The detection model that is developed using the CICToN-IoT dataset.	New
`ciot_attack_detection()`	The detection model for iot devices that is developed using combined telemetry datasets of ToN-IoT	No (D4.3 Subsection 2.1.9)
`network_attack_mitigation()`	Mitigation model that is built based on CSE-CIC-IDS-2018 Dataset	No (D4.3 Subsection 2.1.9)
`cictoniot_network_attack_mitigation()`	Mitigation model that is built based on CICToN-IoT dataset.	New
`ciot_attack_mitigation()`	Mitigation model for iot devices developed using combined telemetry datasets of ToN-IoT [MouTon]	No (D4.3, Subsection 2.1.9)
`network_attack_prediction()`	Prediction model that is built using CSE-CIC-IDS-2018 Dataset	No (D4.3 Subsection 2.1.9)
`cictoniot_network_attack_predition()`	Prediction model that is built based CICToN-IoT dataset.	New
`<iotdev>_attack_prediction()`	Future attack prediction on iot devices that is built using telemetry datasets of ToN-IoT	No (D4.3, Subsection 2.1.9)

Furthermore, the newly introduced `cictoniot_network_attack_*()` modules of Table 2-16, where * denotes `detection`, `mitigation`, and `prediction`, are developed to improve the robustness and generalization capability of the proposed AI models across heterogeneous cybersecurity attack scenarios. These modules extend the experimental scope by incorporating the ToN-IoT [MouTon] and cryptomining [Man24] datasets, thereby enabling a more comprehensive evaluation of model performance under diverse attack conditions. ToN-IoT (Telemetry of Network & IoT) [MouTon] is a large-scale, multi-source cybersecurity benchmark designed for intrusion detection and threat analytics in Internet of Things (IoT) and Industrial IoT (IIoT) environments. The dataset was generated within a realistic cyber range that emulates real-world IoT infrastructures, where both normal operational activities and a diverse set of intentionally executed cyberattacks were conducted to capture representative traffic patterns and attack behaviours. However, the processed network dataset of ToN-IoT was originally generated using Zeek, which produces features in a

format that is incompatible with our model, as the model relies on flow-based features extracted using CICFlowMeter [AppCic]. In addition, the dataset does not include cryptomining traffic, which is of particular interest in the use-case study. Therefore, new traffic flow statistics were regenerated from a combined ToN-IoT and cryptomining dataset using CICFlowMeter. The resulting dataset is referred to as the CICToN-IoT dataset and will be discussed in the next subsection.

2.3.1 CICToN-IoT Dataset

The CICToN-IoT dataset is a network traffic flow statistics dataset generated using CICFlowMeter [AppCic] from raw data of ToN-IoT and cryptomining datasets. This section first describes the process of generating the dataset from the ToN-IoT and cryptomining datasets. Subsequently, the distribution of the dataset and its partitioning into training, validation, and testing sets are presented.

The raw network traffic in ToN-IoT was collected using `netsniff-ng` [Bork] and stored in PCAP format. These PCAP files use the DLT_LINUX_SLL (SLL v2) link-layer format [Tcpd], which is not supported by CICFlowMeter. To overcome this limitation, `scapy` [Scapy] was employed to extract IP-based packets and re-encode them into Ethernet-format PCAP files compatible with CICFlowMeter.

The converted PCAP files were then processed using the `cicflowmeter.sniffer` module to generate unlabeled network flows. Ground-truth labels provided by the ToN-IoT dataset were subsequently aligned with the generated flows using a two-stage strategy:

1. Each unlabeled flow was matched to its corresponding attack instance based on the five-tuple (source IP, source port, destination IP, destination port, and protocol) with a ±15-second temporal tolerance.

2. To further reduce false positives, unmatched flows originating from known attacker IP addresses were removed from the dataset.

Furthermore, Coinhive, Madominer, and Xmrstack are associated with cryptomining activities. Similar to the ToN-IoT dataset, IP-related packets are first extracted using `Scapy` to avoid processing errors and subsequently converted into flow-based statistical features using CICFlowMeter. However, due to the lack of a ground-truth dataset, all records within each respective attack folder are labelled according to the corresponding attack type.

The generated ToN-IoT and Cryptomining datasets are combined to form the CICToN-IoT dataset. The CICToN-IoT dataset comprises records of 12 distinct attack categories in addition to benign traffic. These threats are systematically grouped and mapped onto their respective attack types, denoted as “Attack_Type” in Table . The frequencies, temporal distributions of generated dataset are as shown in Column five of Table and Figure 2-10, respectively.

Table 2-17 Mapping of Label and Attack_Type in CICToN-IoT Dataset

Label	Attack_Type
Benign	Benign
Backdoor	Access
Password	Access
DoS	DoS
DDoS	DDoS
Injection	Web
XSS	Web
Ransomware	Malware
Scanning	Reconnaissance
MITM	MITM
Coinhive	Cryptomining
Madominer	Cryptomining

Xmrstack	Cryptomining

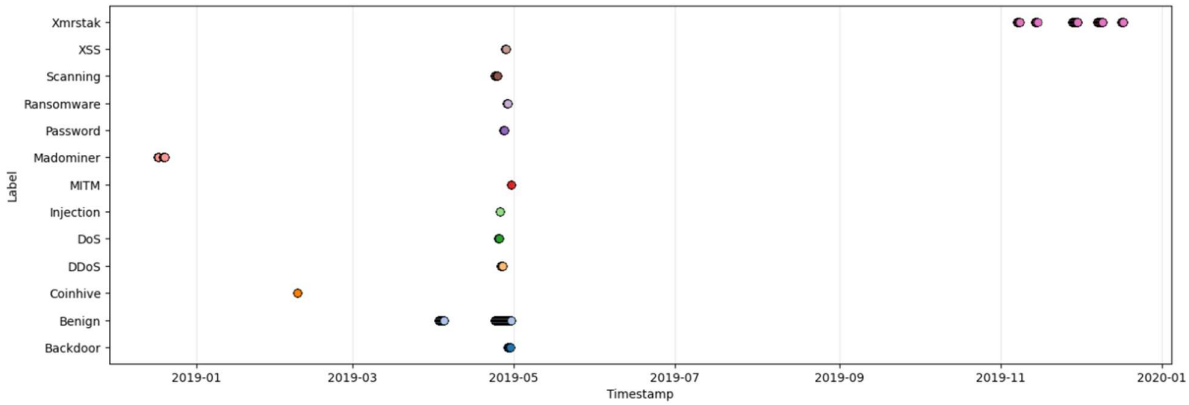


Figure 2-10 Temporal Distribution of Network Attacks in the ToN-IoT and Cryptoming Datasets Generated Using CICFlowmeter.

The network traffic is partitioned based on the temporal sequence of attack occurrences using a 60:20:20 ratio for training, validation, and testing, respectively in sequence. This is to avoid potential data leakage in model building. The temporal segmentation and distribution of these subsets are presented in Table 7-1 to Table 7-3 in Appendix 7.1. In the tables, `window_start` and `window_end` denote the beginning and ending timestamps of each attack experiment window, whereas `attack_start` and `attack_end` indicate the timestamps of the first and last observed attack instances within the corresponding window. Benign traffic is present across all experimental windows, except for the Mandominer, Coinhive, and Xmrstack attacks, which originate from the cryptomining dataset. The benign instances reported in Table 7-1 to Table 7-3 correspond to windows in which only benign traffic is observed, with no concurrent attack activity.

Table 2-18 Attack Frequency in the Training, Validation, and Testing Sets of CICToN-IoT Dataset

Label	Train	Validation	Test	Total
Scanning	4,153,166	1,383,875	1,384,212	6,921,253
DDoS	3,093,563	1,029,967	1,030,768	5,154,298
XSS	1,200,120	399,891	399,944	1,999,955
Benign	653,145	76,201	266,430	995,776
Password	407,542	135,850	135,805	679,197
Injection	264,983	88,214	88,268	441,465
DoS	113,413	37,811	37,797	189,021
Backdoor	10,339	3,446	3,447	17,232
Xmrstak	7,021	2,340	2,342	11,703
Coinhive	5,178	1,726	1,727	8,631
Ransomware	1,669	543	543	2,755
MITM	589	194	191	974
Madominer	363	121	122	606
Total	9,911,091	3,160,179	3,351,596	16,422,866

2.3.2 CICToN-IoT Network Attack Detection

The `cictoniot_network_attack_detection()` module, as shown in Row 3 of Table , is developed based on the CICToN-IoT dataset. This subsection discusses the module's functional specifications, Docker implementation, and model performance.

2.3.2.1 Function Specifications of cictoniot_network_attack_detection()

The function specification is as shown in Table . The function returns a tuple consists of detected attack type and meta data that can be used by security orchestrator for further actions.

Table 2-19 Function Specifications of `cictoniot_network_attack_detection()`

Operation name: `cictoniot_network_attack_detection()`		
Description	**An AI-driven mitigation model that dynamically determines appropriate actions in response to anomalies detected in network traffic of ToN-IoT generated by CICFlowMeter.**	
Input Parameters	**Type**	**Description**
df	**Structured tabular data**	**Data obtain from CICToN-IoT Dataset**
Output Parameters	**Type**	**Description**
Attack_Type	**Str**	**Detected Attack_Type according to Table 2-17**
Meta_df	**Structured tabular data**	**Parameters required for mitigation actions.**
Notes		
NIL		

2.3.2.2 Docker Container

The Docker directory structures for `cictoniot_network_attack_detection`, `cictoniot_network_attack_mitigation`, and `cictoniot_network_attack_prediction` are shown in Figure 2-11(a). The main function, illustrated in Figure 2-11(b), serves as the entry point of the application. It receives input in the form of a CSV file, invokes the respective processing functions, and returns the output in CSV format.

```
r6g/
├── detection/
│   └── __init__.py
├── mitigation/
│   └── __init__.py
├── prediction/
│   └── __init__.py
├── cictoniot_network_attack_detection.py
├── cictoniot_network_attack_mitigation.py
├── cictoniot_network_attack_prediction.py
├── toniot_mappings.json
├── requirements.txt
├── Dockerfile
└── main.py
```

(a)

```python
# ====================================================
# ENTRY POINT
# ====================================================
if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    parser.add_argument(
        "--mode",
        choices=["detection", "mitigation", "prediction"],
        required=True
    )

    parser.add_argument("--input", required=True)
    parser.add_argument("--output", required=True)

    args = parser.parse_args()

    if args.mode == "detection":
        run_detection(args.input, args.output)

    elif args.mode == "mitigation":
        run_mitigation(args.input, args.output)

    elif args.mode == "prediction":
        run_prediction(args.input, args.output)
```

(b)

Figure 2-11 (a) Directory Structure of Docker (b) main.py as Functions' Entry Point

The `test_set.csv` file is the testing dataset derived from the `CICToNIoT` network dataset and serves as the input for evaluating the mitigation function. The dataset comprises 82 features along with a *`Label`* column. The detection command can be executed in PowerShell using the command shown in Figure 2-12(a), where `test_sample.csv` and `det_out.csv` represent the input and output CSV files, respectively. Sample fragments of these CSV files are illustrated in Figure 2-12 (b) and Figure 2-12(c). The first 7 columns of `det_out.csv` form the meta_df, which contains the required metadata for mitigation actions, including the original Attack_Type when the *`Label`* column is provided in `test_sample.csv`. Meanwhile, Detected_Attack represents the predicted attack type from the model.

```
docker run --rm -v "${PWD}:/data" r6g-ids --mode detection --input /data/test_sample.csv --output /data/det_out.csv
```

(a)

Src IP	Dst IP	Src Port	Dst Port	Protocol	...	Subflow Fwd Byts	Subflow Bwd Byts	Label	Anomaly	Attack_Type
192.168.1.35	192.168.1.152	33096	80	6	...	830	821	XSS	1	Web
192.168.1.35	192.168.1.1	36410	53	17	...	237	174	XSS	1	Web
192.168.1.190	203.119.95.53	7528	53	17	...	186	590	Benign	0	Benign

(b)

Flow ID	Src Port	Dst Port	Src IP	Dst IP	Timestamp	Protocol	Attack_Type	Detected_Attack
<M>	33096	80	192.168.1.35	192.168.1.152	2019-04-27 20:30:35	TCP	Web	Web
<M>	36410	53	192.168.1.35	192.168.1.1	2019-04-27 19:47:10	UDP	Web	Web
<M>	7528	53	192.168.1.190	203.119.95.53	2019-04-26 11:49:13	UDP	Benign	Benign

(c)

Figure 2-12 Detection Command and Fragments of Input-Output in CSV
(a) Docker Command (b) test_sample.csv (c) det_out.csv

2.3.2.3 Performance of cictoniot_network_attack_detection()

The classification model of `cictoniot_network_attack_detection()` is XGBoost [Tianqi16] with 400 boosting trees (n_estimators = 400), a maximum tree depth of 7 (max_depth = 7), and a learning rate of 0.3 (learning_rate = 0.3). The model leverage best 10 features of CICToN-IoT dataset obtained using Boruta feature selection as listed in Table . The model achieves 98.15%, 98.15%, 98.19%, and 98.15% for accuracy, F1-score, precision, and recall, respectively, as shown in Table 2-21. The XGBoost model suffers low performance mainly on minority class like MITM where only 191 samples out of over about 3.35 million test records. In addition, as shown in row 9 of Figure 2-13, nearly 21.5% of the cryptomining data are misclassified

as benign. This suggests that the dataset may contain false positive samples that are incorrectly labelled as cryptomining which due to lack of ground-truth from the dataset.

Table 2-20 Best 10 Features Selected by Boruta Feature Selection Technique

Bwd Seg Size Avg	**ACK Flag Cnt**	**Pkt Len Mean**	**TotLen Fwd Pkts**	**Bwd Byts/b Avg**
Pkt Len Max	Bwd Pkt Len Max	Bwd Pkt Len Mean	Fwd Pkt Len Max	Fwd Pkt Len Mean

Table 2-21 Performance of CICToN-IoT Network Attack Detection

	precision	recall	f1-score	support	TP	FP	FN	TN
Benign	0.9665	0.9326	0.9492	266430	248477	8625	17953	3076541
Access	0.8902	0.9826	0.9341	139252	136828	16874	2424	3195470
DoS	0.8969	0.9932	0.9426	37797	37539	4314	258	3309485
DDoS	0.9912	0.9766	0.9838	1030768	1006611	8928	24157	2311900
Web	0.9863	0.9859	0.9861	488212	481351	6673	6861	2856711
Malware	0.9437	0.8637	0.9019	543	469	28	74	3351025
Reconnaissance	0.9889	0.9942	0.9915	1384212	1376144	15458	8068	1951926
MITM	0.3861	0.3194	0.3496	191	61	97	130	3351308
Cryptomining	0.7038	0.5237	0.6005	4191	2195	924	1996	3346481
accuracy			0.9815	3351596				
macro avg	0.8615	0.8413	0.8488	3351596				
weighted avg	0.9819	0.9815	0.9815	3351596				

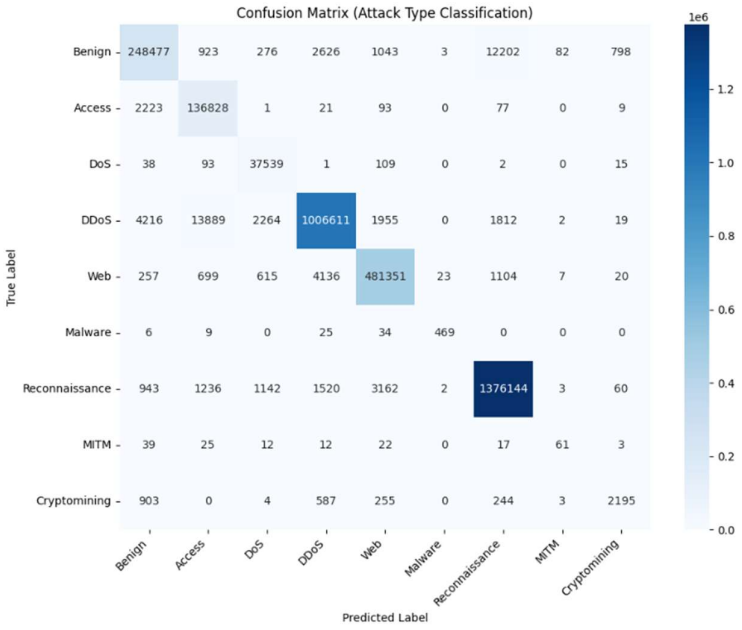


Figure 2-13 Confusion Matrix of `cictoniot_network_attack_detection()`

2.3.3 CICToN-IoT Network Attack Mitigation

The threat mitigation process is illustrated in Figure 2-14. In this process, the data are first cleaned, with missing values imputed and numerical features normalized within the range [0,1]. This is followed by feature selection using a hybrid correlation and mutual information approach. Highly correlated features are removed to minimize redundancy, and for each mitigation action, the ten highest-ranked features with strong relevance

to the target variable and minimal inter-feature correlation are selected. The combined features are as listed in Table 2-22. Finally, a Binary Relevance [Bou04] framework based on the Random Forest [Leo01] classifier is deployed as a multi-label model for the mitigation system. To enhance mitigation effectiveness and address emerging attacks in the ToN-IoT dataset, such as MITM, the mitigation matrix has been updated, as shown in Table 2-23.

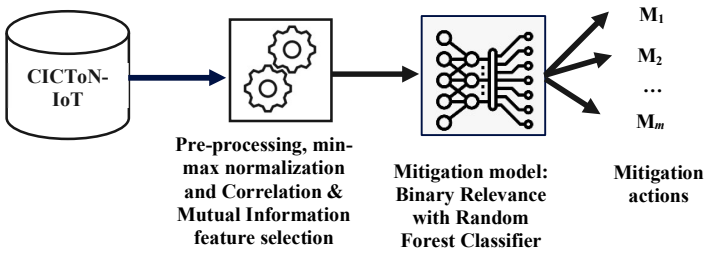


Figure 2-14 Mitigation Process Flow

Table 2-22 Combined Top10 Features of M1 to M16

Bwd IAT Tot	Bwd Pkt Len Max	Bwd Pkt Len Min	Down/Up Ratio	Flow Byts/s	Flow IAT Std
Fwd Pkt Len Max	Fwd Pkt Len Min	Init Bwd Win Byts	Pkt Len Var	Protocol	RST Flag Cnt
Tot Bwd Pkts	Tot Fwd Pkts	TotLen Bwd Pkts	TotLen Fwd Pkts		

Table 2-23 Mitigation Strategies for Different Attack Types

Code	Mitigation of Attack	Access	DoS	DDoS	Web	Malware	Reconn-aissance	MITM	Crypto-mining
M1	Isolating infected system	1*			1*	1		1	1
M2	Close all unused ports via firewall			1			1		
M3	Segmentation / isolation of critical system (to separate VLAN)	1*				1*	1*	1*	1*
M4	Block attacker	1	1		1	1	1	1	1
M5	Blacklist and whitelist attacker	1	1*				1		
M6	Enhance system security credential	1			1*	1		1*	
M7	Rate limiting and throttling to limit the incoming connection attempts	1	1	1	1		1		
M8	Traffic filtering and scrubbing		1*	1			1*		
M9	Perform patching/update	1*			1	1	1*	1*	1
M10	Perform system reinstallation (or IoT device fresh and secure SW image)				1*	1*			1*
M11	IoT platform reset / factory reset					1*			1*

M12	New IoT platform deployment								1*
M13	Account lockout for repeated failures	1			1*				
M14	Implement CAPTCHA				1				
M15	Notify network operator/provider	1*	1*	1*		1*	1*		
M16	Notify vendor	1*			1*	1*		1*	1*
	1 – Mitigation on all Severity Level								
	1* – Mitigation on High Severity Level								

2.3.3.1 Function Specifications of cictoniot_network_attack_mitigation()

The function specifications of `cictoniot_network_attack_mitigation()` is as shown in Table 2-24. The function returns a tuple consists of a set of mitigation action yM1, yM2, … yM16 and meta data that can be used by security orchestrator for mitigation.

Table 2-24 Threat Mitigation of CICToNIoT Network Traffic

Operation name: `cictoniot_network_attack_mitigation()`		
Description	An AI-driven mitigation model that dynamically determines appropriate actions in response to anomalies detected in network traffic of ToN-IoT generated by CICFlowmeter.	
Input Parameters	**Type**	**Description**
df	Structured tabular data	Data obtain from CICToN-IoT Dataset
Output Parameters	**Type**	**Description**
yMi, i=1,2,…16	Boolean vector	Mitigation actions represented in the form of binary vector. A value of 0 indicates no action is required, while a value of 1 signifies that the corresponding mitigation action should be executed.
Meta_df	Structured tabular data	Parameters required for mitigation actions.
Notes		
NIL		

2.3.3.2 Docker Container of cictoniot_network_attack_mitigation()

The Docker directory structures for `cictoniot_network_attack_mitigation` is as shown in Figure 2-11(a). The main function, illustrated in Figure 2-11(b), serves as the entry point of the application. It receives input in the form of a CSV file, invokes the cictoniot network attack mitigation processing functions, and returns the output in CSV format.

The mitigation command can be executed in PowerShell using the command shown in Figure 2-15(a), where `test_sample.csv` and `mit_out.csv` represent the input and output CSV files, respectively. Sample excerpts of these CSV files are illustrated in Figure 2-12(b) and Figure 2-15(b). The first 24 columns of *mit_out.csv* correspond to metadata fields that contain the required information for mitigation actions, including the true mitigation labels (M1, M2, …, M16) when the Label column is provided in `test_sample.csv`. Meanwhile, the binary outputs yM1, yM2, …, yM16 represent the predicted mitigation actions generated by the model. A value of 1 in yMi indicates that mitigation action Mi, as listed in Table , should be executed. Accordingly, the security orchestrator can trigger the appropriate mitigation actions based on the outputs yM1 to yM16.

```
docker run --rm -v "${PWD}:/data" r6g-ids --mode mitigation --input /data/test_sample.csv --output /data/mit_out.csv
```

(a)

Flow ID	Src Port	Dst Port	Src IP	Dst IP	Timestamp	Protocol	Attack_Type	M1	...	yM8	yM9	yM10	yM11	yM12	yM13	yM14	yM15	yM16
<M>	33096	80	192.168.1.35	192.168.1.152	2019-04-27 20:30:35	TCP	Web	1	...	0	1	1	0	0	1	1	0	1
<M>	36410	53	192.168.1.35	192.168.1.1	2019-04-27 19:47:10	UDP	Web	1	...	0	1	1	0	0	1	1	0	1
<M>	7528	53	192.168.1.190	203.119.95.53	2019-04-26 11:49:13	UDP	Benign	0	...	0	0	0	0	0	0	0	0	0

(b)

Figure 2-15 (a) Mitigation Docker's Command (b) Fragment of mit_out.csv

2.3.3.3 Performance Evaluation of Mitigation Function

The CICToN-IoT Network Attack Mitigation Model employs a Binary Relevance multi-label classification technique. In this approach, each mitigation action is modelled as an independent binary classification task and optimized using a Random Forest classifier. As shown in last raw of Table , the overall performance of this mitigation is 98.89%, 98.71%, 98.27% and 99.18% on accuracy, F1, precision and recall, respectively.

Table 2-25 Performance of CICToN-IoT Network Attack Mitigation

Label	precision	recall	f1	accuracy	support	TP	FP	FN	TN
M1	0.9568	0.9922	0.9742	0.9901	632389	627479	28325	4910	2690882
M2	0.9936	0.9892	0.9914	0.9876	2414980	2388966	15457	26014	921159
M3	0.9788	0.9948	0.9867	0.9878	1528389	1520381	32877	8008	1790330
M4	0.9794	0.9984	0.9888	0.9862	2054398	2051114	43116	3284	1254082
M5	0.9785	0.9966	0.9875	0.9882	1561261	1555984	34185	5277	1756150
M6	0.9576	0.9941	0.9755	0.9907	628198	624515	27644	3683	2695754
M7	0.9909	0.9782	0.9845	0.9717	3080241	3012966	27653	67275	243702
M8	0.9934	0.9898	0.9916	0.9877	2452777	2427748	16037	25029	882782
M9	0.9792	0.9983	0.9886	0.9862	2016601	2013096	42857	3505	1292138
M10	0.9858	0.9874	0.9866	0.9961	492946	486739	7031	6207	2851619
M11	0.1060	0.6977	0.1841	0.9913	4734	3303	27855	1431	3319007
M12	0.0925	0.6838	0.1630	0.9912	4191	2866	28111	1325	3319294
M13	0.9567	0.9943	0.9751	0.9905	627464	623899	28240	3565	2695892
M14	0.9892	0.9912	0.9902	0.9971	488212	483930	5281	4282	2858103
M15	0.9894	0.9982	0.9938	0.9903	2592572	2587938	27736	4634	731288
M16	0.9568	0.9922	0.9742	0.9901	632389	627479	28325	4910	2690882
MACRO	0.8678	0.9548	0.8835	0.9889	21211742				
MICRO	0.9804	0.9918	0.9861	0.9889	21211742				
WEIGHTED	**0.9827**	**0.9918**	**0.9871**	**0.9889**	21211742				

2.3.4 CICToN-IoT Future Network Attack Prediction

The future network attack prediction module, as indicated in Table , is designed to forecast potential attacks that may occur within five minutes into the future, based on network activity observed in the preceding five minutes. As illustrated in Figure 2-16, the prediction workflow begins with the CICToN-IoT dataset, which provides labelled network flow statistics comprising both benign and malicious traffic patterns. The first step involves pre-processing, where the raw data are cleaned by removing inconsistencies, handling missing values, and formatting the dataset into a structured form suitable for analysis. Following this, data consolidation (1 record per second) is performed to aggregate network flow statistics into fixed one-second intervals, thereby creating a consistent temporal representation of network activity. The consolidated data are then subjected to Min–Max normalization, which scales each feature to a uniform range to prevent features with larger numerical values from dominating the learning process. Next, the normalized data are organized into time-series stacks

using sliding windows of 300 seconds with a stride of 60 seconds, enabling the model to capture temporal patterns and evolving attack behaviours over time. From these time-series windows, TSFresh feature selection [Maxi18] is applied to automatically extract and statistically evaluate a large set of time-series features, retaining only those that are significantly relevant to attack prediction while eliminating redundant ones. For each future attack type, the top 20 most correlated features are selected. These selected features are subsequently used during model building, where the dataset is prepared and used to train the prediction model. The model adopts a Binary Relevance strategy with a Random Forest classifier, in which separate binary classifiers are trained for each attack category. Finally, the trained model produces future attack predictions (FAT_i, $i \in$Attack_Types), where the subscripts denote specific attack types such as Access, DoS, and DDoS, as defined in Table . The output represents the likelihood of particular attacks occurring within the upcoming five-minute time window, thereby enabling proactive detection and mitigation actions before the attacks actually take place.

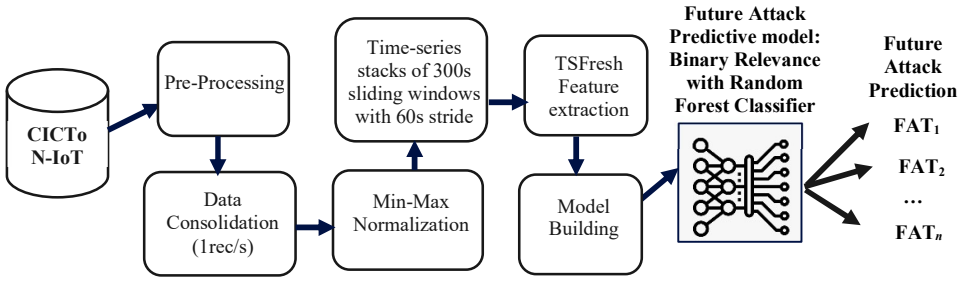


Figure 2-16 Process Flow of Future Attack Prediction

2.3.4.1 *Function Specifications of cictoniot_network_attack_prediction()*

As indicated in Table 2-26, the input to `cictoniot_network_prediction()` consists of statistical network flow features generated by CICFlowMeter. The minimum set of required input features is listed in Table 2-27, representing the union of all raw features selected by the TSFresh algorithm across all attack types (Attack_Type). The function outputs a tuple comprising the predicted future attack indicators, denoted as yFAT_i, where *i* represents the corresponding attack type, and meta_df, a metadata dataframe. The *meta_df* includes the Timestamp, indicating the current observation time, and the true future attack type (FAT_i) when the *Label* column is provided in the input dataset.

Table 2-26 Functionality Specifications of cictoniot_network_attack_prediction() Module

Operation name: `cictoniot_network_attack_prediction()`		
Description	An AI-driven future attack prediction model that determines potential future attack in next 5 minutes.	
Input Parameters	**Type**	**Description**
df	Structured tabular data	Data obtain from CICToN-IoT Dataset
Output Parameters	**Type**	**Description**
yFAT_i, $i \in$Attack_Types	Boolean vector	yFAT_i represented in the form of binary vector. A value of 1 indicates future Attack_Type i is detected, while a value of 0 signifies that the future Attack_Type i is not detected.
Meta_df	Structured tabular data	Timestamp (current time) and True Future Attacks (FAT_i) if Label provided in df.
Notes		
NIL		

Table 2-27 Raw Features Used in Future Attack Predictive Module

Bwd Byts/b Avg	Bwd Header Len	Bwd IAT Min	Bwd IAT Std	Bwd IAT Tot	Bwd PSH Flags
Bwd Pkt Len Max	Bwd Pkt Len Std	Bwd Pkts/b Avg	Down/Up Ratio	Dst Port	FIN Flag Cnt

Flow IAT Mean	Fwd Blk Rate Avg	Fwd Byts/b Avg	Fwd PSH Flags	Fwd Pkt Len Max	Fwd Pkt Len Mean
Fwd Pkt Len Min	Fwd Pkt Len Std	Fwd Pkts/b Avg	Fwd Seg Size Min	Init Fwd Win Byts	PSH Flag Cnt
Pkt Len Max	Pkt Len Min	Pkt Size Avg	Protocol	RST Flag Cnt	SYN Flag Cnt
Subflow Bwd Byts	Subflow Fwd Byts	Tot Fwd Pkts			

2.3.4.2 Docker Container for cictoniot_network_attack_prediction()

The Docker command and the output from cictoniot_network_attack_prediction() module are as shown in Figure 2-17.

```
docker run --rm -v "${PWD}:/data" r6g-ids --mode prediction --input /data/test_sample_pre.csv --output /data/pre_out.csv
```

(a)

FAT_Access	FAT_Cryptomining	FAT_DDoS	FAT_DoS	FAT_MITM	FAT_Malware	FAT_Reconnaissance	FAT_Web	FAT_Benign	Timestamp	yFAT_Access	yFAT_Cryptomining	yFAT_DDoS	yFAT_DoS	yFAT_MITM	yFAT_Malware	yFAT_Reconnaissance	yFAT_Web	yFAT_Benign
0	0	1	0	0	0	0	0	0	26/4/2019 10:57	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 10:57	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 10:58	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 10:58	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 10:59	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 10:59	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 11:00	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 11:00	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 11:01	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 11:01	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 11:02	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	26/4/2019 11:03	0	0	1	0	0	0	0	0	0

(b)

Figure 2-17 (a) Docker Command and (b) Fragment of Output from Network Future Attack Prediction

2.3.4.3 Performance Evaluation of Future Attack Predictive Function

The performance of the Future Attack Type (FAT) prediction model is summarized in Table 2-28. Overall, the model achieved an accuracy of 95.36%, with micro-averaged precision, recall, and F1-score of 0.8806, 0.7272, and 0.7966, respectively. The higher precision compared to recall indicates that the model produces reliable predictions when an attack is identified, although some attack instances remain undetected. The macro F1-score of 0.6953, which is lower than the weighted F1-score of 0.7744, further suggests that predictive performance varies across attack categories due to class imbalance.

The model performs particularly well for FAT_Cryptomining, FAT_Web, and FAT_Access, achieving F1-scores of 0.9799, 0.9664, and 0.9399, respectively. These results indicate that these attack types exhibit distinctive traffic characteristics that can be effectively captured by the model. For FAT_DoS, the model achieved a high recall of 0.9710 but a lower precision of 0.6505, indicating that most DoS attacks are successfully detected but with a higher number of false positives.

In contrast, several attack categories show lower recall, particularly FAT_DDoS (0.3087), FAT_Malware (0.3818), and FAT_MITM (0.1667). This indicates that the model is conservative when predicting these attacks, resulting in fewer false positives but a larger number of missed attacks. The weaker performance for these classes is likely due to limited sample sizes (FAT_Malware and FAT_MITM) and overlapping traffic characteristics with benign flows (FAT_DDoS). Overall, the results demonstrate that the proposed model can effectively predict several major attack types while maintaining high overall classification accuracy.

Table 2-28 Performance of Future Attack Prediction

Future Attack	precision	recall	f1	accuracy	support	TN	FP	FN	TP
FAT_Access	0.9767	0.9057	0.9399	0.9868	371	2885	8	35	336
FAT_Cryptomining	1.0000	0.9606	0.9799	0.9896	862	2402	0	34	828
FAT_DDoS	1.0000	0.3087	0.4718	0.8882	528	2736	0	365	163
FAT_DoS	0.6505	0.9710	0.7791	0.9651	207	2949	108	6	201
FAT_MITM	0.5000	0.1667	0.2500	0.9853	48	3208	8	40	8
FAT_Malware	1.0000	0.3818	0.5526	0.9896	55	3209	0	34	21

FAT_ Reconnaissance	0.8462	0.4724	0.6063	0.9694	163	3087	14	86	77
FAT_Web	0.9536	0.9796	0.9664	0.9939	294	2956	14	6	288
FAT_Benign	0.7810	0.6541	0.7120	0.8143	1145	1909	210	396	749
MICRO	0.8806	0.7272	0.7966	0.9536	3673				
MACRO	0.8564	0.6445	0.6953	0.9536	3673				
WEIGHTED	**0.8926**	**0.7272**	**0.7744**	**0.9536**	3673				

2.4 Zero-Touch Re-Authentication for Scalable 6G Swarm Security

While Sections 2.1 - 2.3 present the main ZTSP building blocks for orchestration, monitoring, detection, prediction, and mitigation, this section addresses a more specific security mechanism for dense 6G swarm environments. Its role is to cover the access-network/physical-layer side of zero-touch security, complementing the broader platform functions introduced earlier. In this sense, the section extends the chapter from general ZTSP enablers to a trust-continuity mechanism based on re-authentication and live channel characteristics.

2.4.1 Context and Problem

Classical PKI-based attestation requires a full multi-phase handshake for every session. In dense 6G swarms, such as UAV fleets or large IoT clusters, frequent reconnections make this approach inefficient. Repeated Diffie–Hellman exchanges and certificate validations introduce unnecessary latency, redundant identity proofs, and increased radio overhead, consuming valuable spectrum.

This work mitigates these inefficiencies by separating initial trust establishment from session continuity. After a single full attestation during onboarding, subsequent sessions are derived from prior cryptographic state and live channel characteristics, eliminating repeated PKI ceremonies while preserving security guarantees.

2.4.2 Zero-Touch Trust Chain

The framework distinguishes two operational phases:

Session 0: Full Attestation. A 4-phase PKI handshake establishes a shared cryptographic root and initial channel fingerprint F_0:

$$K_0 = \mathrm{HKDF}(\mathrm{SDH} \parallel \tau_0 \parallel F_0)$$

where SDH is the Diffie-Hellman secret, τ_0 the attestation token, and F_0 the initial CSI-derived fingerprint.

Session $n \geq 1$: Zero-Touch Renewal. A lightweight ZC pilot exchange extracts a fresh fingerprint $F_n = \Phi(CSI_n)$. If the continuity condition holds, re-authentication proceeds without any PKI re-exchange:

$$\|F_n - F_{n-1}\| \leq \varepsilon \;\rightarrow\; K_n = \mathrm{HKDF}(K_{n-1} \parallel F_n \parallel N_n) \;\rightarrow\; \text{AES-GCM session}$$

$$\|F_n - F_{n-1}\| > \varepsilon \;\rightarrow\; \text{Full re-attestation triggered}$$

The fingerprint vector incorporates AoA profile, AoD, path loss that produce statistically independent fingerprints, making remote forgery physically infeasible.

2.4.3 Security Analysis

The security properties of the proposed mechanism follow from two design choices introduced above: each renewal step is chained to the previous cryptographic state, and each session update depends on both a fresh channel-derived fingerprint and a nonce. Consequently, the most relevant threats are those targeting session continuity, stale credential reuse, and impersonation during renewal. As described in Table 2-29, in this setting, spoofing, replay, and man-in-the-middle attempts fail to establish a valid renewed session, whereas significant cumulative drift is detected and handled by forcing full re-attestation.

Table 2-29: Attacks, Status and Actuated countermeasures

Attack	Status	Countermeasure

Spoofing	BLOCKED	Cannot replicate F_n without occupying the same physical location, different multipath = fingerprint mismatch.
Replay	BLOCKED	K_n depends on live F_n and fresh nonce N_n; captured credentials are cryptographically stale.
Man-in-the-Middle	BLOCKED	Node and AP independently derive F_n; a relay node produces divergent CSI → fingerprint alarm.
Gradual Drift	DETECTED	Cumulative drift $\|F_n - F_0\| > \varepsilon\_global$ forces mandatory full re-attestation with no escalation path.

2.4.4 Conclusion

The proposed Zero-Touch Re-Authentication mechanism enables scalable trust continuity in 6G swarms by chaining session keys to live channel fingerprints. Full attestation is performed only once, while subsequent sessions remain physics-bound and cryptographically linked. Drift detection preserves continuous integrity through a mandatory safety reset, reducing the overhead of classical PKI attestation without weakening security guarantees.

3 Zero-Touch Security Platform component integration

This section details the functional integration of the Zero Touch Security Platform (ZTSP) components and defines their operational workflows. Section 3.1 outlines the workflows from SSLA ingestion to the deployment of Security Services (SSe), while Section 3.2 provides illustrative examples of Security Closed Loops (S-CLs) featuring both rule-based and AI-based detection. While these sections demonstrate the functional logic and interaction between the modules, they represent a preliminary integration and validation phase. The physical integration and operational demonstration will be conducted within Work Package 6 (WP6). Specifically, Use Case 2 (UC2) will serve as the final environment for validating the platform's end-to-end Zero Touch Security Orchestration (ZTSO) capabilities in a real-world scenario.

3.1 Operational workflows of the ZTSP

This section defines the operational workflows required to initialize and manage the ZTSP. The narrative follows the lifecycle of the platform, beginning with the initial setup of the ZTSO (Sections 3.1.1 and 3.1.2), followed by the provisioning of security services based on specific SSLAs (Section 3.1.3), and concluding with service decommissioning and update (Section 3.1.4). It is important to note that the workflows presented here are intended to showcase component interoperability and logic rather than specific attack scenarios or environment-tailored remediations. Furthermore, while the current phase focuses on logical integration, several components remain physically distributed across multiple testbeds and are not yet interconnected at the infrastructure level. A comprehensive evaluation, encompassing physical integration, real-world threat modelling, and performance metrics collection, will be conducted as part of WP6 UC2.

3.1.1 Security Orchestrator Ontology and Knowledge Graph Setup

Figure 3-1 depicts the operational workflow of setup of the ZTSO Ontology Manager. This operation is carried out by an administrator, which has to (i) populate the Ontology Manager TBox, (ii) Populate the Ontology Manager ABox, and (iii) trigger reasoning over the ABox using the rules and definitions of the TBox.

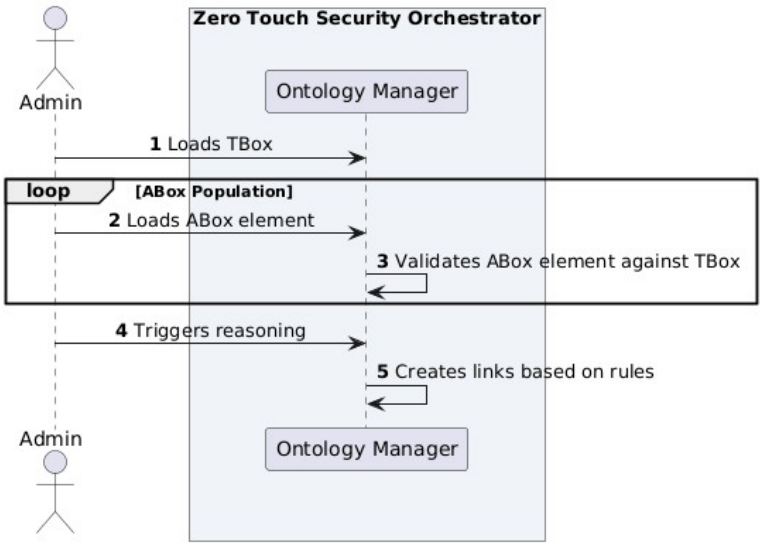


Figure 3-1 TZSO TBox and ABox Setup

All these operations are performed through the Ontology Manager and are further documented below. As a first step, the admin must setup the Ontology Manager ABox and TBox. The TBox Setup is performed by uploading *turtle.ttl* file on the Ontology Management API, reported in Figure 3-2, using the */ontology/upload* method described in [Nex25a].

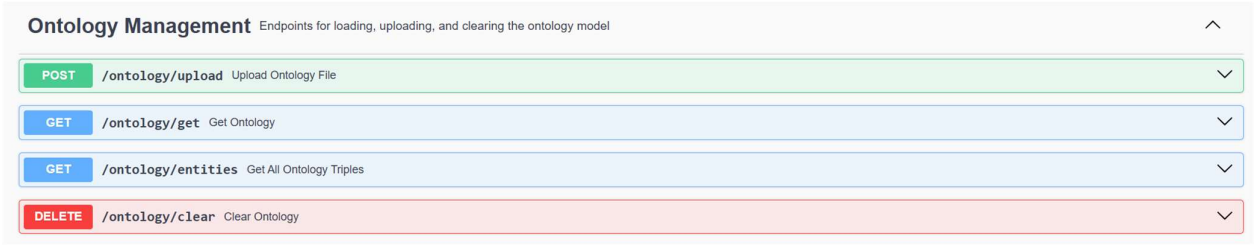


Figure 3-2 Ontology Manager - TBox Management APIs

Once the TBox file is uploaded, it can be visualized from the admin through the Ontology Manager GUI, which enables the admin to dynamically explore the ontology as a graph, as shown in Figure 3-3.

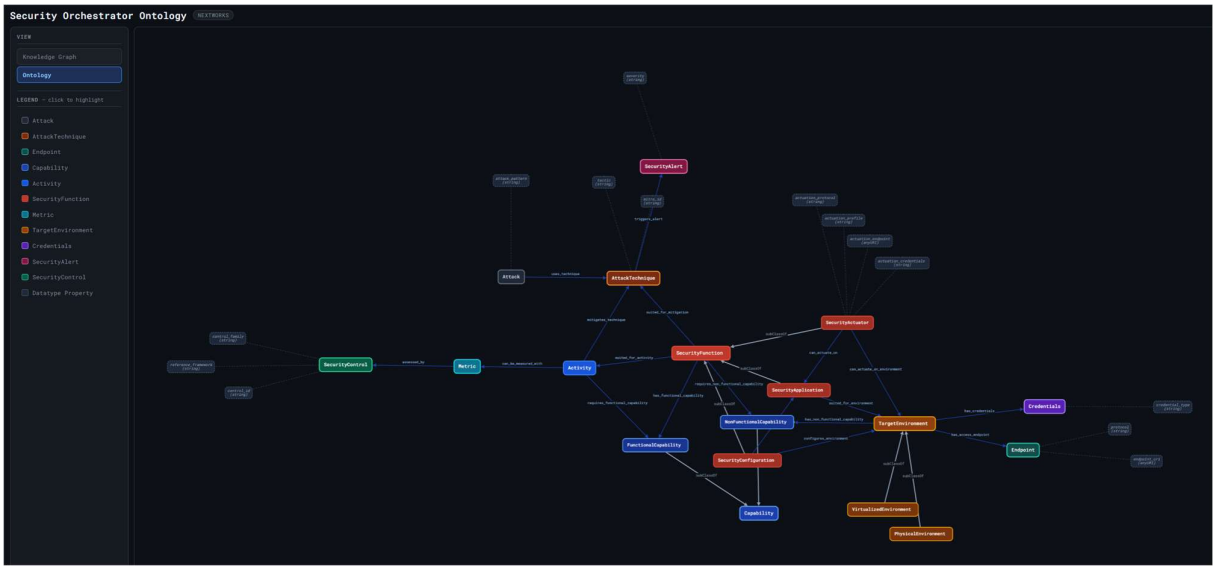


Figure 3-3 Ontology Manager - TBox Visualization from GUI

Once the TBox is loaded, the Ontology Manager is ready to accept incoming requests for the population of the ABox. These requests are accepted through the Knowledge Graph Management API, depicted in Figure 3-4, and described in [Nex25a].

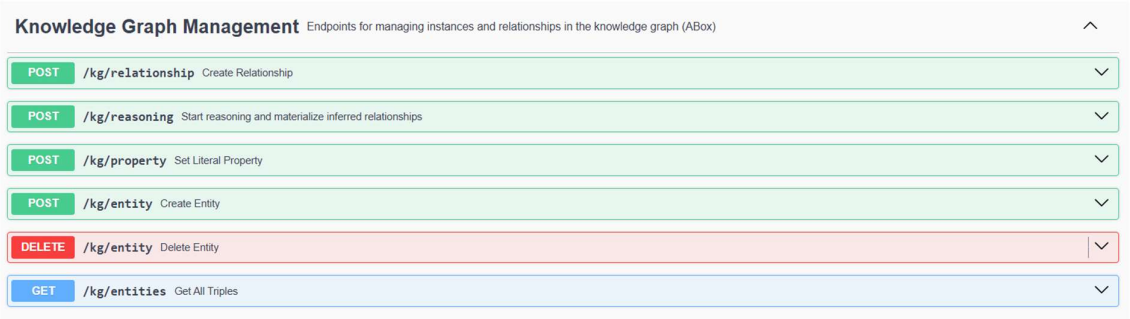


Figure 3-4 Ontology Manager - ABox Management APIs

The admin has to upload entities, set literal properties to entities, and create relationships between entities. While the ROBUST-6G TBox is fixed, the ABox is dynamic and will be updated to demonstrate the ROBUST-6G use cases. However, for the sake of completeness, Figure 3-5 shows the logs of the Ontology Manager while checking if a relationship is compliant with the TBox, during its population. Figure 3-6 depicts the Ontology Manager Knowledge Graph GUI view of a single Security Function before the reasoning is triggered and Figure 3-7 shows the same view but after triggering the reasoning. As can be noted from the images, based

on the ontology rules defined in [R6G24-D41], the *suited_for_activity* and *suited_for_environment* relationships are automatically inferred by the reasoning and the ABox is populated accordingly.

```
=== [Ontology Compliance Check] ===
Validating triple:
   Subject  : https://nextworks.it/security-orchestrator#ProgrammableMonitoringPlatform
   Predicate: https://nextworks.it/security-orchestrator#has_functional_capability
   Object   : https://nextworks.it/security-orchestrator#IoTMonitoringCapability

✅ Match 1 found:
   Domain      : https://nextworks.it/security-orchestrator#SecurityFunction
   Range       : https://nextworks.it/security-orchestrator#FunctionalCapability
   Subject Type: https://nextworks.it/security-orchestrator#SecurityApplication
   Object Type : https://nextworks.it/security-orchestrator#FunctionalCapability

SPARQL Update:
INSERT DATA {
  GRAPH <http://security-orchestrator.nextworks.it/graph/abox/> {
    <https://nextworks.it/security-orchestrator#ProgrammableMonitoringPlatform>
    <https://nextworks.it/security-orchestrator#has_functional_capability>
    <https://nextworks.it/security-orchestrator#IoTMonitoringCapability>
  }
}
```

Figure 3-5 ABox population - example of TBox check

Figure 3-6 depicts the Ontology Manager Knowledge Graph GUI view of a single Security Function before the reasoning is triggered and Figure 3-7 shows the same view but after triggering the reasoning. As can be noted from the images, based on the ontology rules defined in [R6G24-D41], the *suited_for_activity* and *suited_for_environment* relationships are automatically inferred by the reasoning and the ABox is populated accordingly.

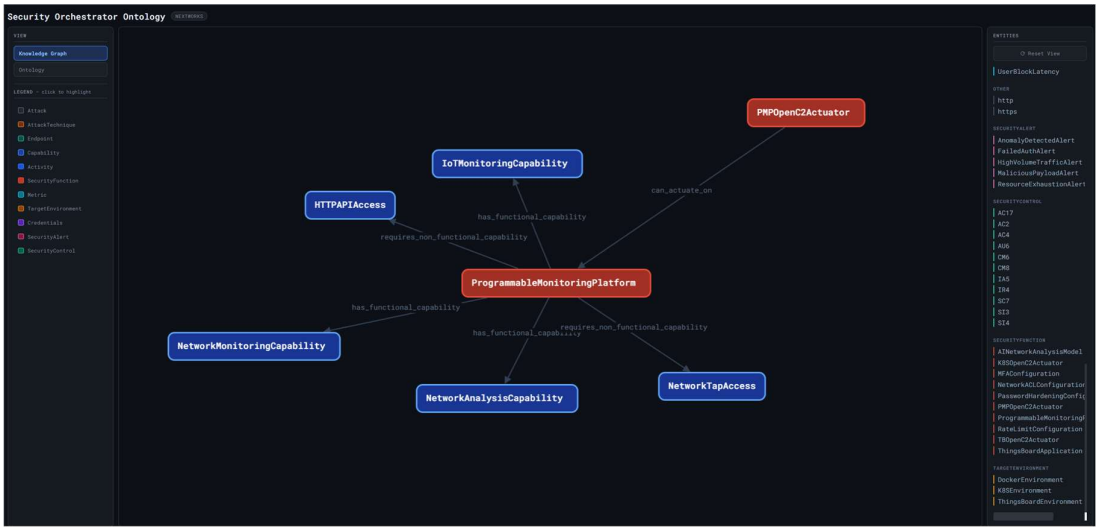


Figure 3-6 ABox - security function view before triggering reasoning

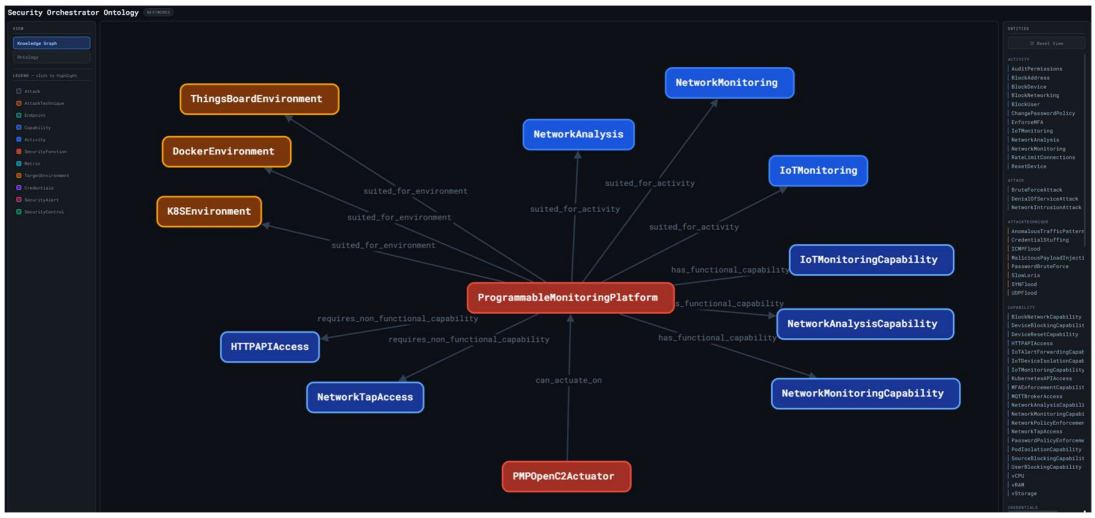


Figure 3-7 ABox - security function view after reasoning

3.1.2 Security Orchestrator Catalogue and Security Resource Orchestrator Setup

Figure 3-8 depicts the operational workflow of setup of Security Functions on the ZTSO Catalogue Manager and the S-RO. This operation is carried out by an Administrator who has to: (i) upload the SF Templates on the S-RO; (ii) upload to the CM the SF descriptors including the S-RO orchestration templates (name and version); As detailed in D4.3 [R6G25-D43], the S-RO has the role to maintain in its service template catalogue the cloud-native descriptors (e.g., Helm Templates) related to SFs, while the CM has the role of storing the security-related metadata of SFs as its Functional Capabilities and the supported Activities. However, it is important to note that SFs can also be uploaded directly to the CM without being registered in the S-RO. This applies to SFs that cannot be physically deployed by the S-RO (e.g., Physical Layer Security Functions). These are still described within the CM to allow for the composition of services that involve these types of non-orchestrable components.

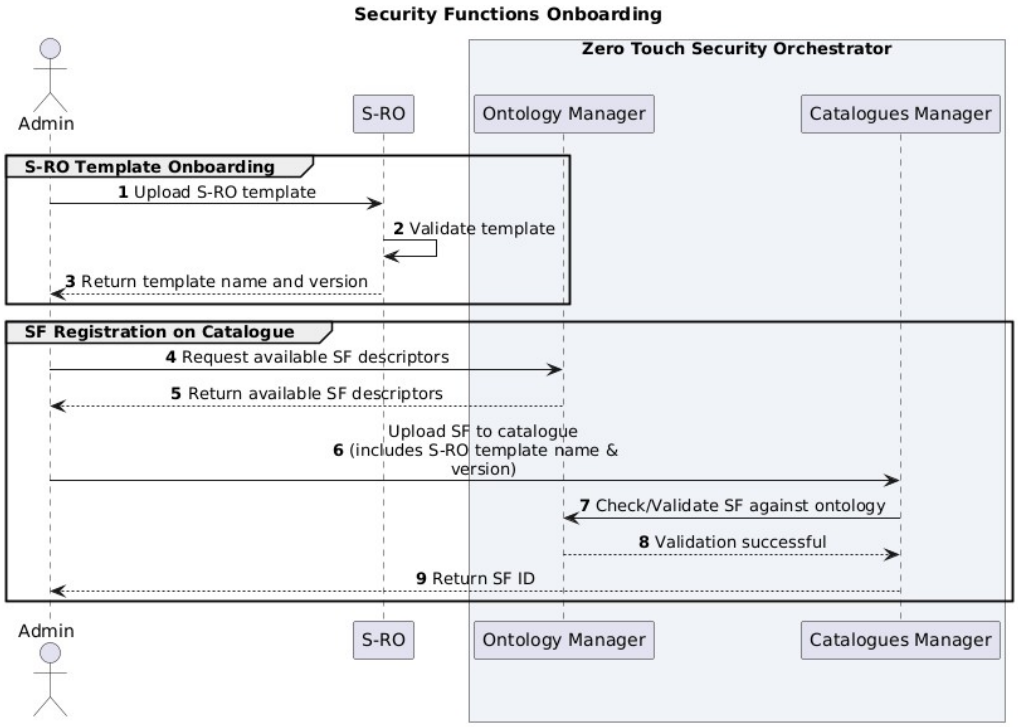


Figure 3-8 ZTSO CM and S-RO Setup of Security Functions

Step (i) can be both performed from the S-RO APIs, described in [Nex24e] or from the S-RO GUI for Service templates upload reported in Figure 3-9. Step (ii) can be performed on the ZTSO Catalogue Manager APIs reported in [Nex24b] and depicted Figure 3-9.

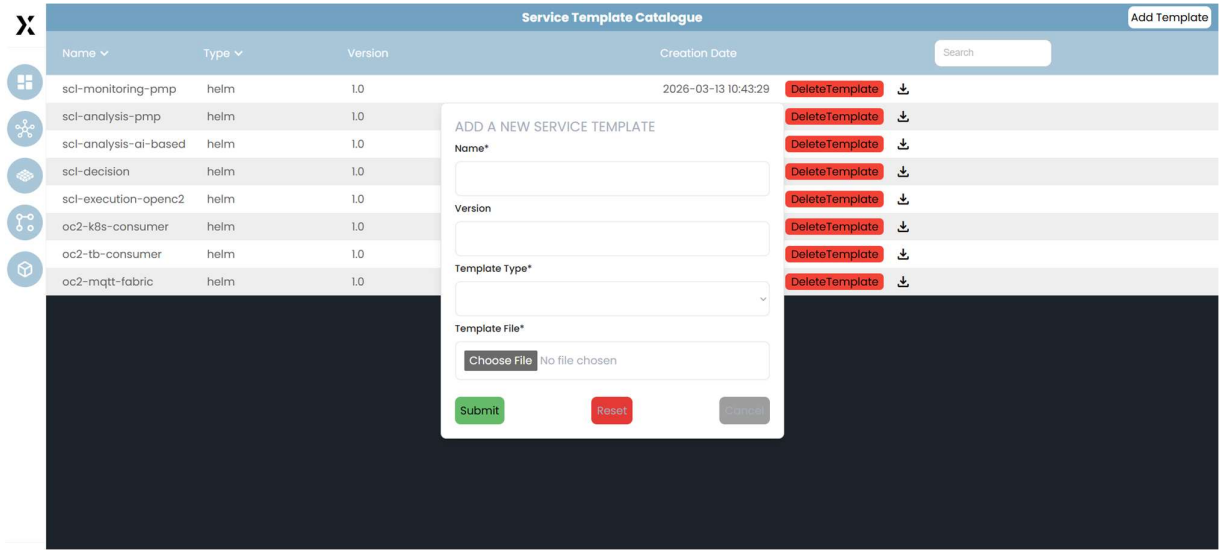


Figure 3-9 S-RO GUI for Service Template Upload

Figure 3-10 ZTSO Catalogue Manager - Security Functions Management APIs

Figure 3-10 depicts the operational workflow of setup of Target Environments on the ZTSO Catalogue Manager and the S-RO. This operation is carried out by an Administrator who has to: (i) upload the Platform template on the S-RO; (ii) upload to the CM the Target Environment descriptors including the retrieved S-RO platform_id. As detailed in D4.3 [R6G25-D43], the S-RO has the role to manage platforms that are defined using cloud native formats (e.g., K8S kubeconfig file), while the CM has the role of storing the security-related metadata of SFs as its NonFunctional Capabilities, Credentials, and Endpoints for interaction. Similarly to SFs, Target Environments can be registered directly in the CM without S-RO involvement. This applies to environments not actively managed by the S-RO (e.g., bare-metal machines). These unmanaged platforms are still registered in the CM as potential environments for security enforcement through specific OpenC2 Actuators.

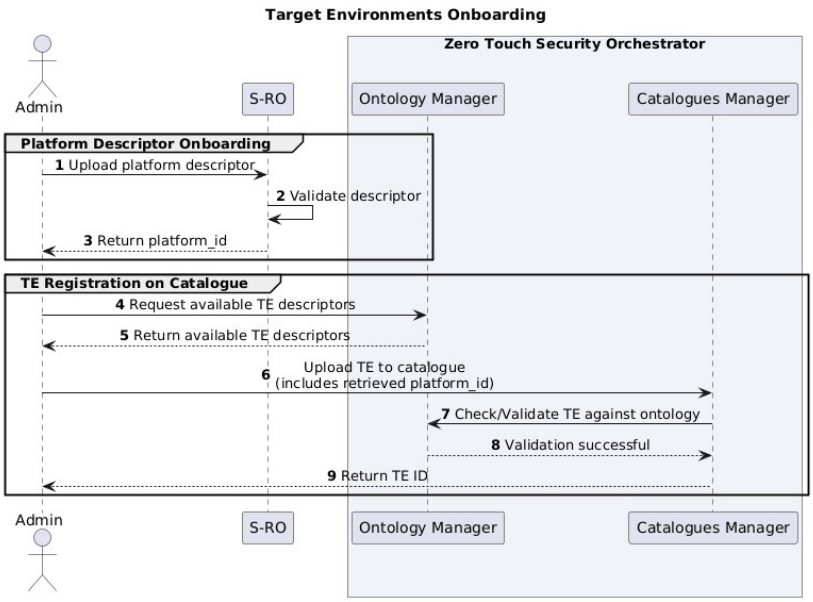


Figure 3-11 ZTSO CM and S-RO Setup of Target Environments

Step (i) can be both performed from the S-RO APIs, described in [Nex24e] or from the S-RO GUI for Platforms Management reported in Figure 3-12. Step (ii) can be performed on the ZTSO Catalogue Manager APIs reported in [Nex24b] and depicted in Figure 3-13.

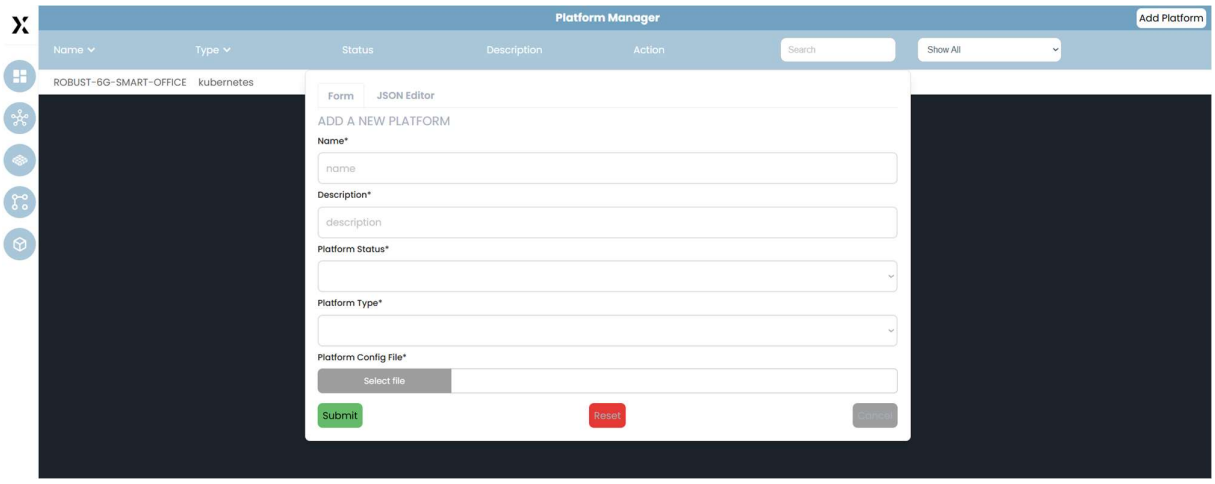


Figure 3-12 S-RO GUI for Platforms Upload

Figure 3-13 ZTSO Catalogue Manager - Target Environment Management APIs

Once the SFs and TEs are uploaded correctly in both the CM and the S-RO, the last step (depicted in Figure 3-14) encompasses the management of infrastructures. Specifically, this step involves logically linking Target Environments to specific Infrastructures, and subsequently associating Security Functions with those Target Environments. While this linking often occurs automatically at orchestration time, such as when an SF is dynamically deployed on a TE to enable a requested security capability, it can also be executed manually. As previously established, not all Security Functions and Target Environments can be orchestrated by the ZTSP but still, they can be part of a Security Service. Therefore, an Administrator can manually perform this linking to configure environments that cannot be managed by the S-RO (e.g., bare metal servers) and associate them with non-orchestrable SFs. This ensures that even static, physical, or unmanaged assets are properly mapped within the overall topology and available for security policy enforcement.

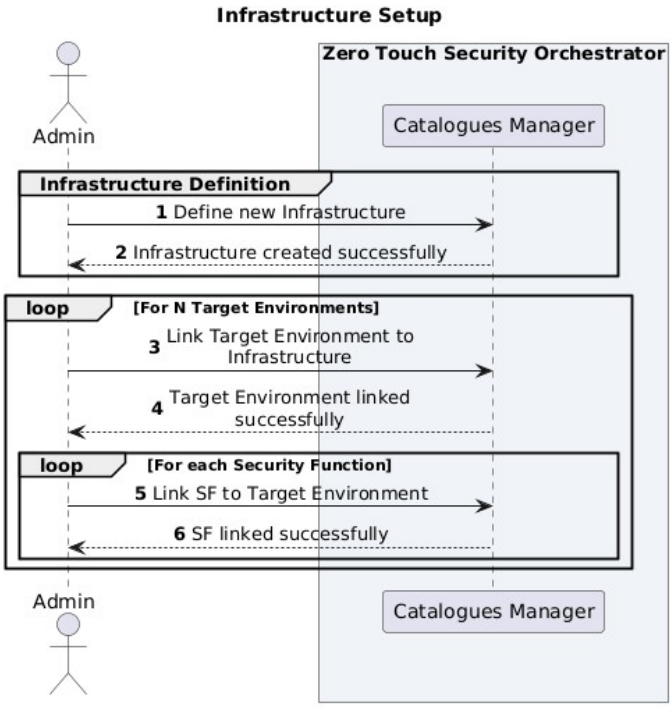


Figure 3-14 ZTSO CM Setup of Infrastructures

The steps depicted in Figure 3-14 can be performed on the ZTSO Catalogue Manager APIs reported in [Nex24b] and depicted in Figure 3-15.

Figure 3-15 ZTSO Catalogue Manager - Infrastructure Management APIs

3.1.3 Security Closed Loops Descriptors and Functions Descriptors Setup

This section reports the final configuration step of the ZTSP: the setup of the Security Closed Loop (S-CL) Management component. As stated in D4.3 [R6G25-D43], this component is logically divided into two main parts: (i) the S-CL Catalog, which maintains the metadata for the S-CL Functions (which are linked to their respective orchestration templates in the S-RO) as well as the S-CL descriptors, which are composed of multiple interconnected functions; (ii) the S-CL Lifecycle Management (LCM), engine responsible for the runtime management, execution, and orchestration of the closed loops, linked to the ZTSO-SCM. Figure 3-16 illustrates the onboarding process carried out by an Administrator. During this workflow, the individual S-CL functions and the composite S-CL descriptors are uploaded to the manager and validated. Once successfully registered, these S-CLs can be utilized to automate security services by dynamically ingesting configuration parameters at runtime. This setup process can be executed either programmatically via the S-CL Manager APIs detailed in [Nex25c], or manually through the dedicated GUIs.

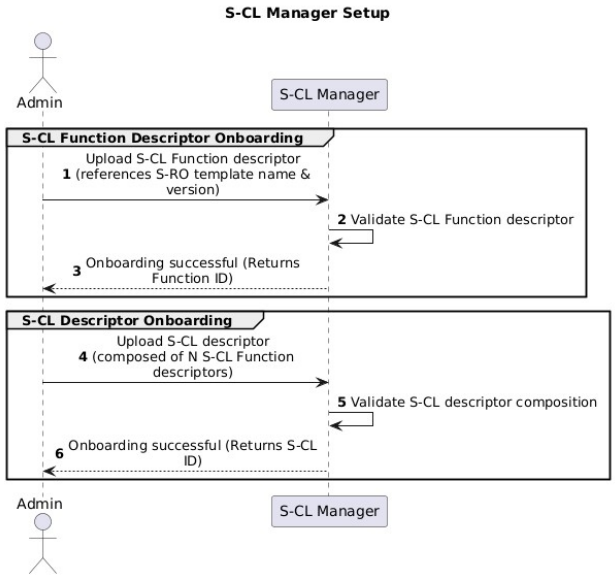


Figure 3-16 S-CL Manager – Setup of S-CL Functions and S-CL Descriptors

Figure 3-17 provides an example of an S-CL Function, Figure 3-19 and Figure 3-19 depict respectively the dedicated interfaces for uploading and managing S-CL Functions and S-CL descriptors. Finally, Figure 3-20 presents a detailed view of a fully configured S-CL descriptor composed of four distinct functions working together to form a complete closed loop.


```
{
   "description": "Security execution function, acts as CACAO OpenC2 Producer.",
   "name": "execution-scl",
   "version": "1.0",
   "orchestrated": true,
   "sharable": false,
   "deploymentConstraints": [],
   "externalOrchestrator": null,
   "orchestratorTemplate": {
      "url": null,
      "name": "scl-execution-openc2",
      "version": "1.0"
   },
   "configurationAttributes": [
   ],
   "inputParameters": [
      "valkeyHost",
      "valkeyPort"
   ],
   "outputParameters": [
   ],
   "replicas": 1,
   "requirements": {
      "cpu": 0.5,
      "memory": 512,
      "storage": 0
   },
   "functionDescriptorType": "EXECUTION"
}
```


Figure 3-17 Example of S-CL Function Descriptor

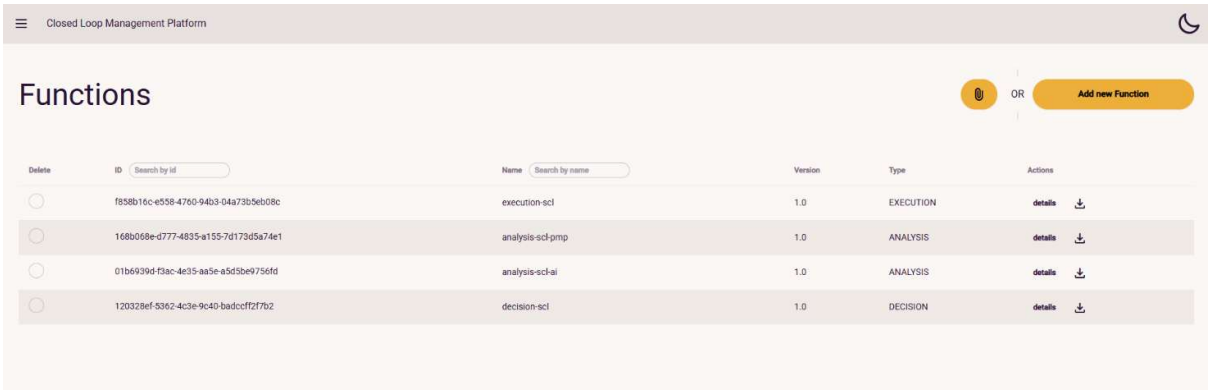


Figure 3-18 S-CL Manager - GUI for S-CL Functions descriptors management

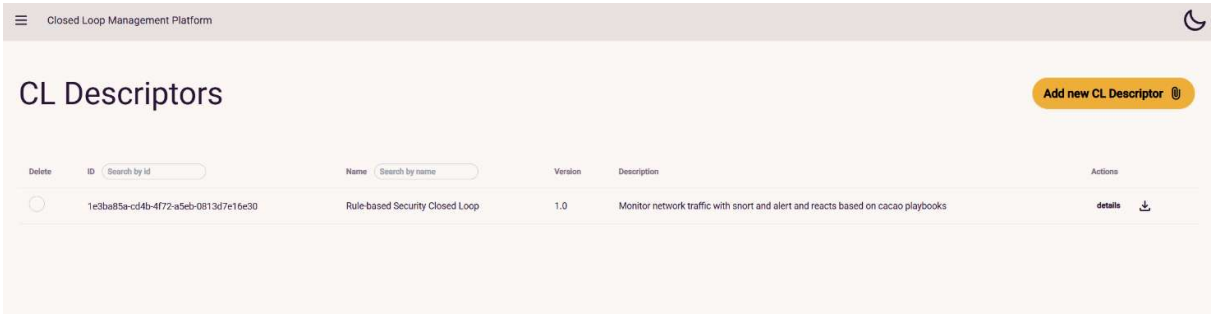


Figure 3-19 S-CL Manager - GUI for S-CL descriptors management

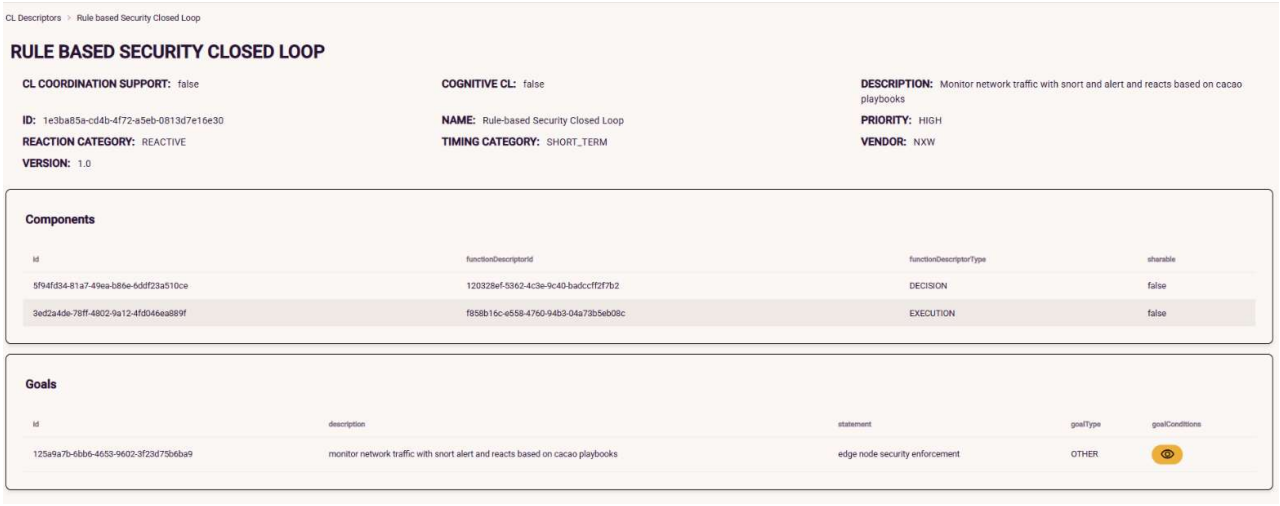


Figure 3-20 S-CL Manager - Complete view of a S-CL descriptor

3.1.4 Security Service Provisioning

The Security Service provisioning is applied by the ZTSO's components alongside an optional and external generative AI service to ensure the automation of the maintenance of this security service. While GenAI4SOAR significantly accelerates the provisioning process by automatically selecting SFs and generating CACAO playbooks, its integration is entirely optional. In scenarios where the platform cannot rely on this external GenAI service, the security service can still be enforced manually. Importantly, whether the process is entirely manual or AI-assisted, all steps involving SF selection and playbook generation are designed as entry points for human validation. This ensures a robust "human-in-the-loop" approach, allowing an Administrator to review, modify, or approve the AI-generated selections before final enforcement. The first part of the security service provisioning starts from a client request and terminates with semantic-driven

selection of the right Security Functions and the Gen-AI assisted generation of CACAO Playbooks for automated responses. This phase can be further divided into three main steps that are described in this section.

SSLA ingestion and validation

As depicted in Figure 3-21, the objective of this phase is to validate and manage security policies coming from clients:

1. A client (user, consumer, web client) sends a security policy to the Policy Manager with an infrastructure ID, to enforce security requirements on this infrastructure.
2. This first component detects the type of policy and sends it to the proper component to validate the format and content of the policy. For example, if the policy type matches the SSLA standard format, the Policy Manager sends the policy to the SSLA Manager.
3. This policy is validated, parsed and stored for security policy management in future steps.
4. The Policy Manager gets the parsed security policy as a result.

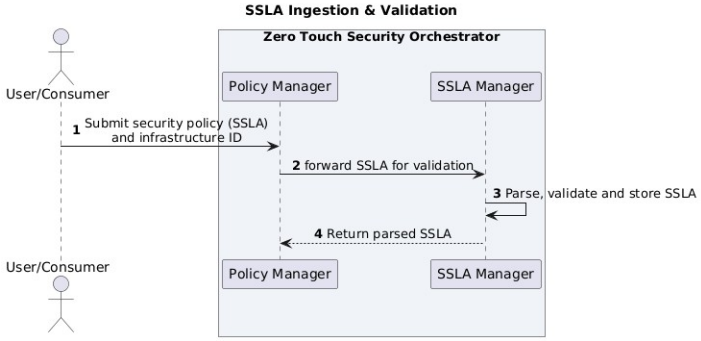


Figure 3-21 Security Service Provisioning - SSLA Ingestion and Validation

The Policy Manager is addressed by Rest clients by committing to the API depicted in Figure 3-22. The Policy Manager handles the SSLA by committing to the API of the SSLA Manager depicted in Figure 3-23.

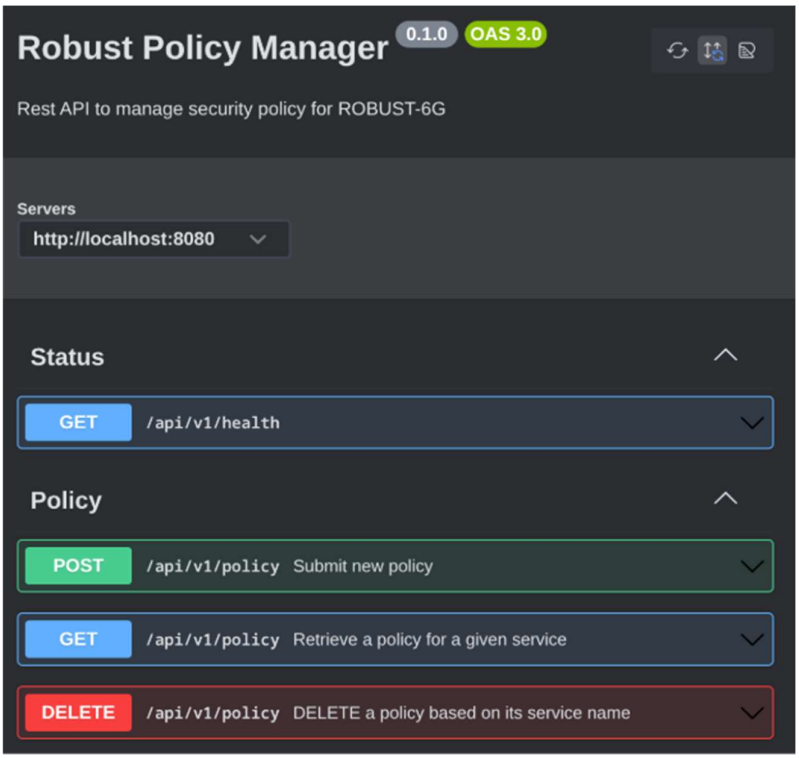


Figure 3-22: OpenAPI Specification of the Policy Manager REST Interface

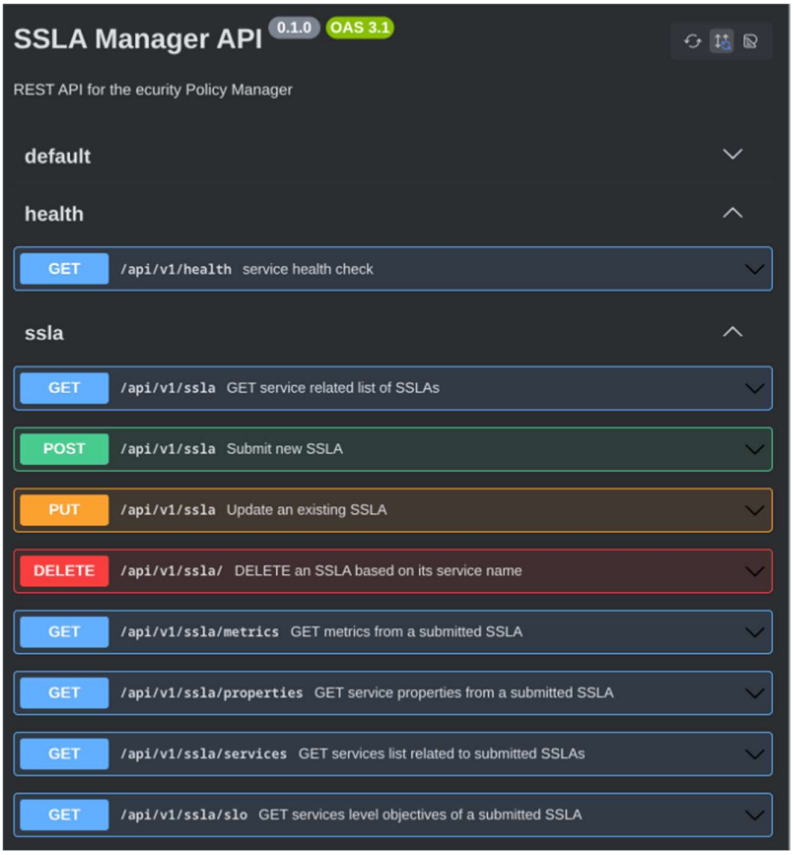


Figure 3-23: OpenAPI specification of the SSLA Manager REST interface

Security Function Selection and Context Retrieval

As depicted in Figure 3-24, the objective of this phase is to retrieve the context of a security policy about enforceable security functions and target infrastructure to deploy them:

1. The Policy Manager sends a request to get the supported security metrics and activities from the Ontology Manager. A security metric is composed with a name (e.g. *multi_factor_authentication*), and a set of possible values (e.g., [2, 3, 4]), while activities are composed of metrics and define a higher level of security detail (e.g., authentication)
2. The Ontology Manager responds with supported activities and metrics.
3. The Policy Manager sends the parsed Security Level Objectives (SLO), activities with metrics and the infrastructure ID to the Security Context Manager. SLO defines the quantified requirement for a metric (e.g., *multi_factor_authentication=2*).
4. The Security Context Manager selects appropriate security functions that match the activities, metrics and objectives. The Security Context Manager then sends a request to the Catalogue Manager containing the infrastructure ID to get more context information.
5. The Catalogue Manager responds with the targeted environments (e.g., Kubernetes) by the infrastructure, the existing functions and the deployable functions.
6. The Security Context Manager persists the activities, metrics, SLOs, existing and selected functions about the infrastructure to manage it for future phases.
7. The Security Context Manager responds to the Policy Manager with this infrastructure context. The context can also include existing information about the same infrastructure, functions or CACAO playbooks applied to this context. The CACAO playbook may have been generated in the past, as described in the next phase.

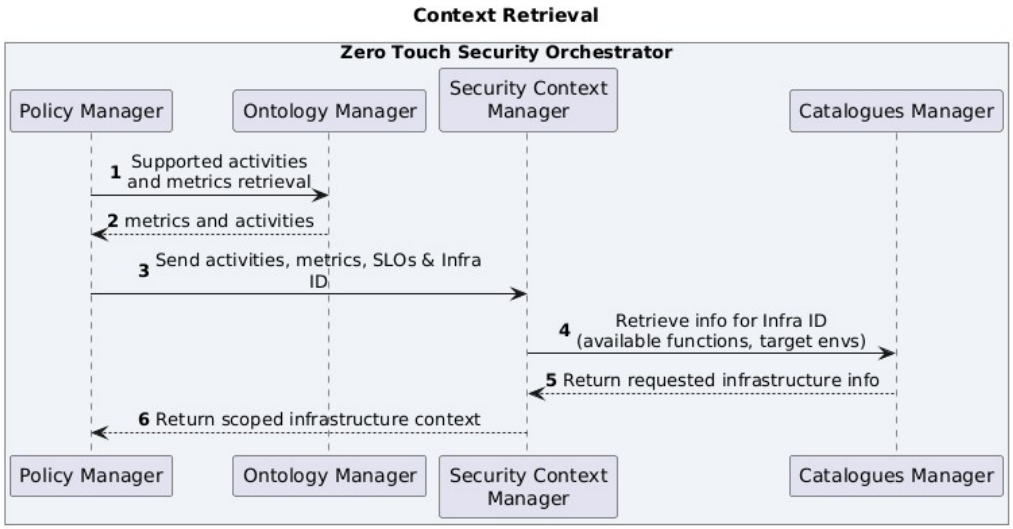


Figure 3-24 Security Service Provisioning - Context retrieval

Security Functions Selection and Playbook Generation

As depicted in Figure 3-25, the objective of this phase is to select the right SFs from the retrieved previously retrieved context and generate a security maintenance playbook, using a generative AI service, applied to enforced security functions in an infrastructure:

1. The Policy Manager sends all the information about the policy, the selected security functions and the targeted infrastructure to the GenAI Gateway component.
2. The GenAI Gateway parses this information into a high-level textual format and sends it to an external generative AI service, asking to generate a CACAO playbook to maintain these security functions against modifications and threats.
3. The GenAI Gateway gets a CACAO playbook to maintain the policy in return.
4. The GenAI Gateway sends the new CACAO playbook to the Security Context Manager, with the infrastructure ID, to enrich the context of the former infrastructure for potential future phases.

As previously discussed, steps 4 and 5 represent the dedicated entry points for human-in-the-loop validation. Furthermore, the broader sequence spanning steps 2 through 7, which encompasses the selection of Security Functions (SFs) and the generation of the CACAO playbook, can be: (i) entirely be carried out by a human, (ii) carried out by a human in the selection of the SF with reliance on GenAI for CACAO Playbook generation, or (iii) totally automated through the utilize of GenAI.

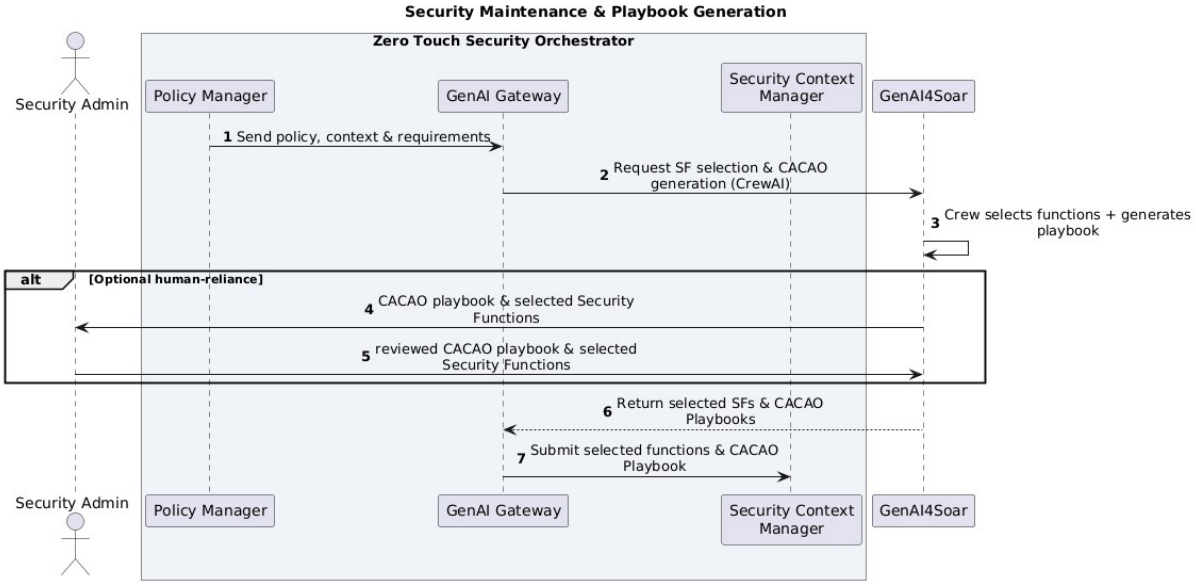


Figure 3-25 Security Service Provisioning - Security Functions selection and Playbook Generation

Once this phase is complete, the plan is now ready to be associated with a service descriptor through the SCM Service Lifecycle Manager component. As depicted in Figure 3-26, following the semantic and AI-assisted selection of SFs and generation of the CACAO playbook, the Administrator is now responsible for building a Vertical Service Blueprint (VSB) that describes how these SFs and selected S-CL Functions are linked and communicate with one another. This operation is envisioned to be carried out by an Administrator in the context of ROBUST-6G, but it can be easily automated though the usage of GenAI with human supervision.

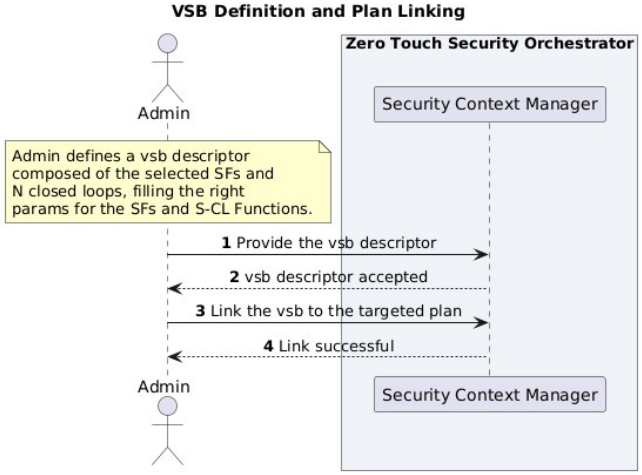


Figure 3-26 Security Service Provisioning - VSB definition and Plan linking

The operation of VSB uploading (steps 1-2) can be done through the SCM API for VSB management reported in [Nex25c] and depicted in Figure 3-27 or via the dedicated GUI for VSB uploading and management reported in Figure 3-28. Finally, Figure 3-30, reports an example of VSB composed of two Security Functions and linked to a CL previously uploaded in the S-CLM component.

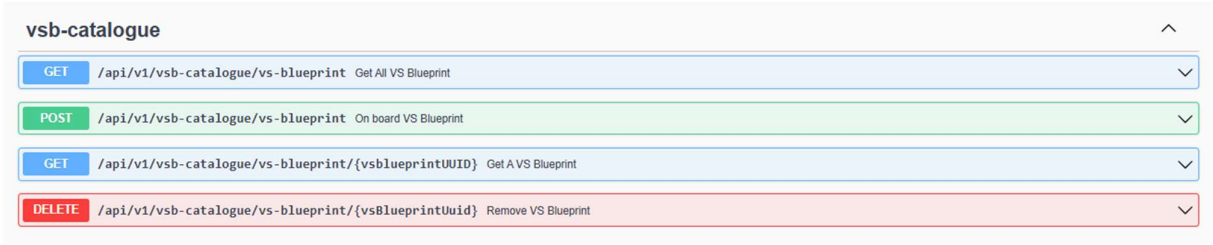


Figure 3-27 ZTSO Security Context Manager - VSB Management APIs

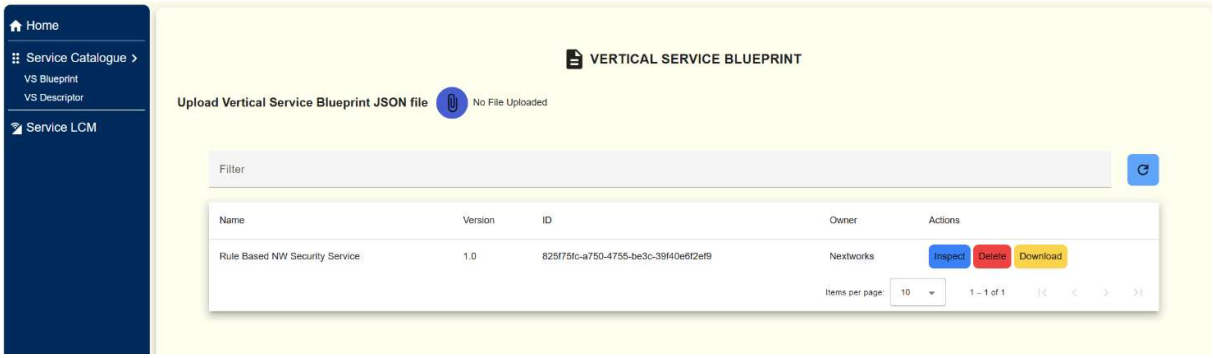


Figure 3-28 SCM GUI for VSBs management

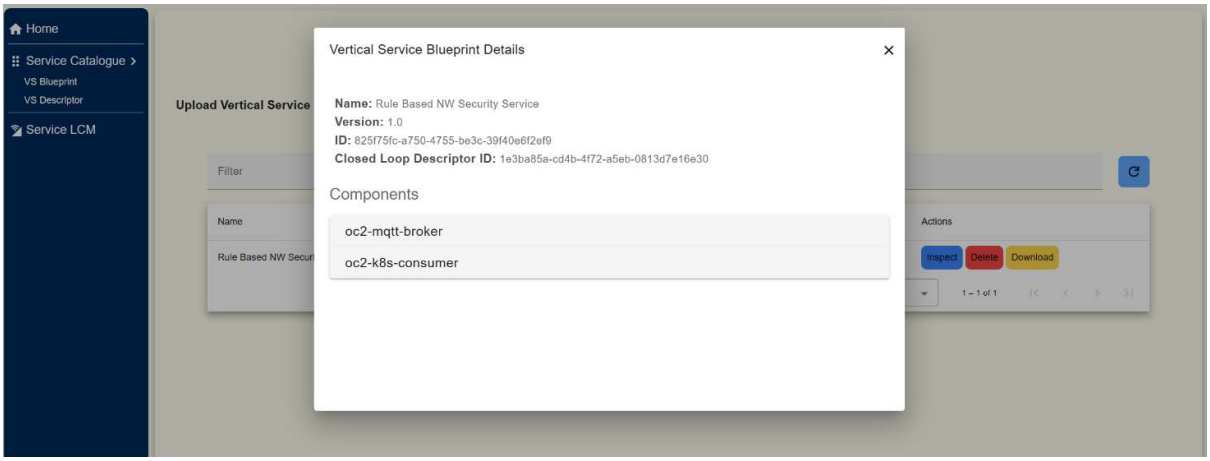


Figure 3-29 Example of VSB for NW security with linked SCL

The operation of VSB linking to a plan (steps 3-5) can be done through the SCM API for Plans management reported in [Nex25c] and depicted in Figure 3-30.

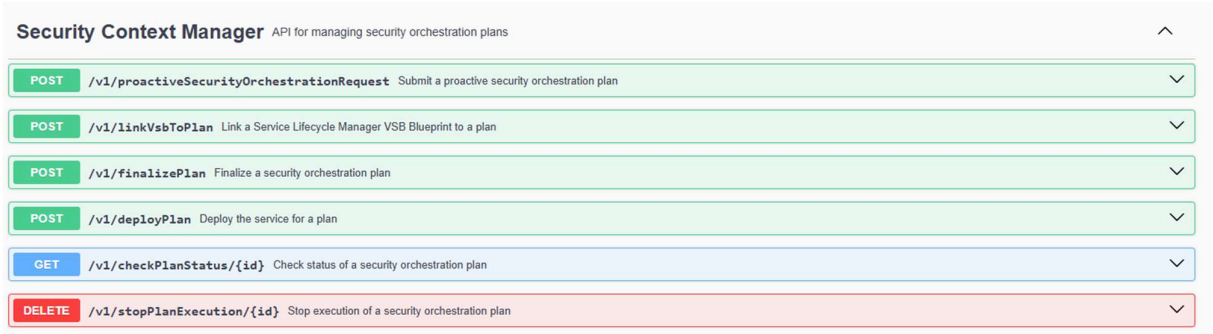


Figure 3-30 ZTSO Security Context Manager - Plans Management API

Once the plan is linked to its VSB, the final step of Security Service Provisioning is the deployment of the Service associated with the plan through the same SCM API for Plans management reported in Figure 3-30 by

providing all the configuration parameters defined in the VSB. This step, reported in Figure 3-31, starts from the Administrator initiating the execution of the plan via the SCM. The deployment follows a strict sequential enforcement logic. First, the SCM requests the S-RO to deploy the selected SFs; once provisioned and configured on the infrastructure, the S-RO acknowledges the successful deployment back to the SCM. Only after the underlying SFs are fully deployed and operational does the SCM trigger the S-CLM to deploy the associated closed loops. The S-CLM then instructs the S-RO to deploy the specific S-CL Functions onto the infrastructure. Once these functions are successfully configured, the deployment acknowledgments cascade back from the S-RO to the S-CLM, and finally to the SCM, marking the completion of Security Service deployment.

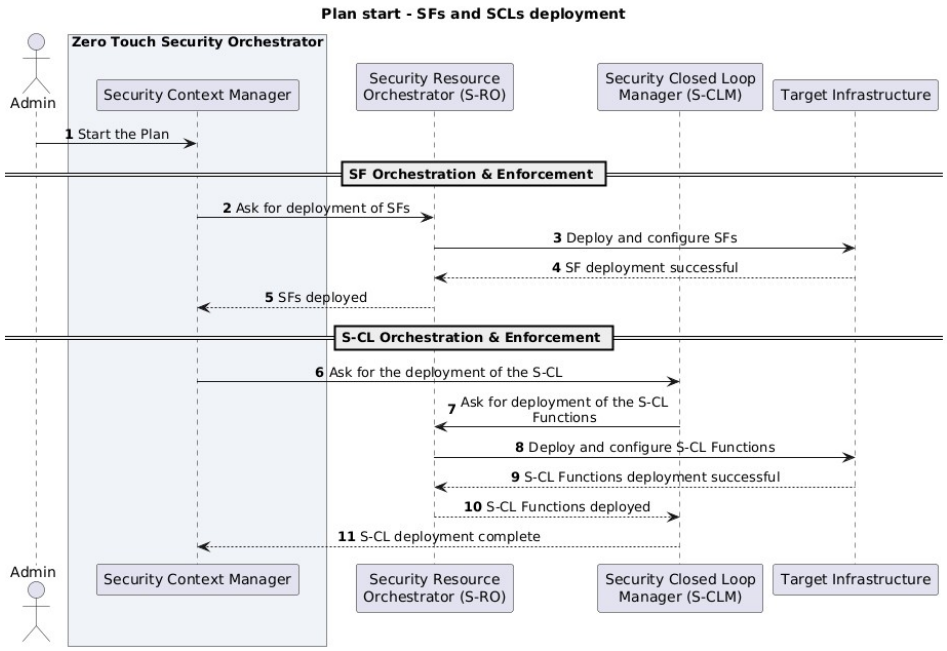


Figure 3-31 Security Service Provisioning - Security Service deployment

Once the Service results are instantiated from the SCM API described in [Nex25f] and the GUI depicted in Figure 3-32 the status of the service can be analysed. Figure Figure 3-33 reports both the components related to the service descriptor of figure Figure 3-29 deployed in the target infrastructure, and the Security Closed Loop Functions related to the Security Closed Loop reported in Figure 3-20. The actual execution of the loop, in this case, rule-based for the detection of threats on a K8S cluster, is described as example in section 3.2.1.

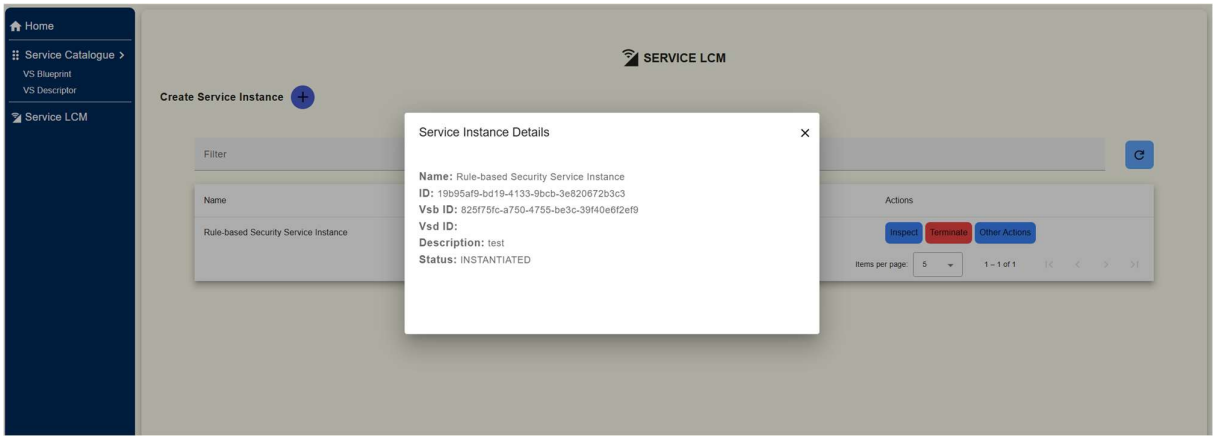


Figure 3-32 SCM GUI for Security Service Management

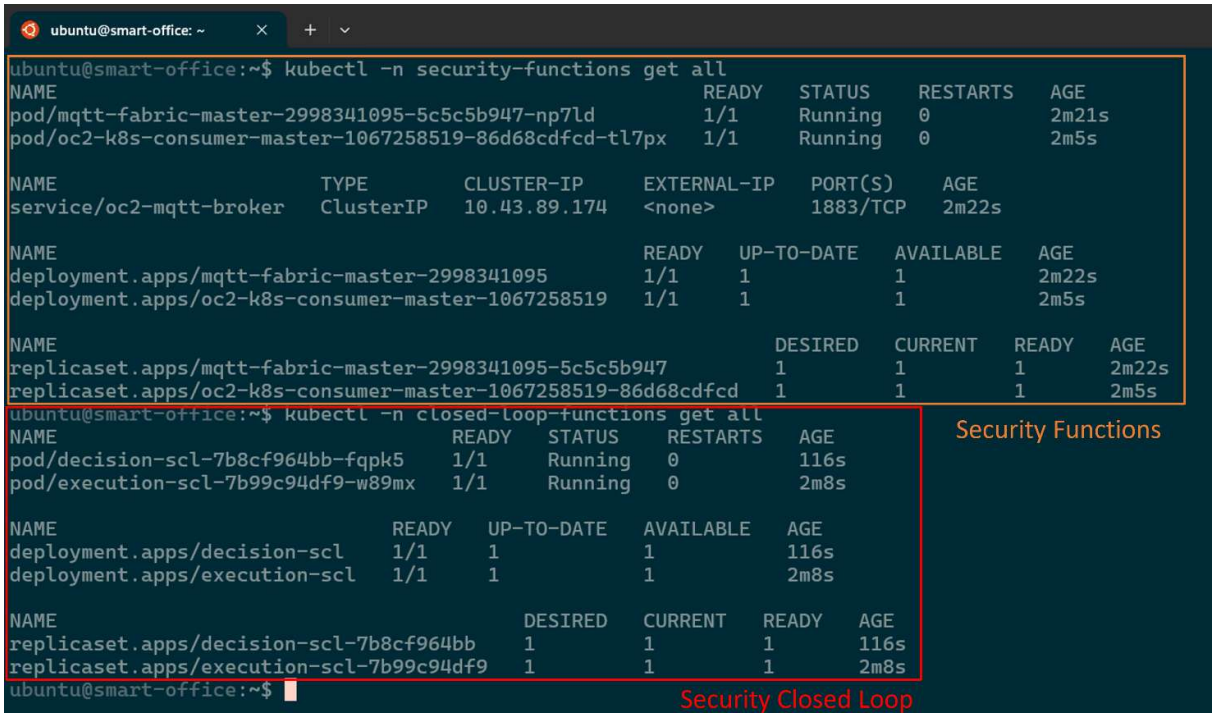


Figure 3-33 Security Functions and Security Closed Loop Deployed

3.1.5 Security Service Decommissioning

The decommissioning of security services, as depicted in Figure 3-34, starts from an explicit request from ZTSP consumer. To initiate the removal of a deployed service, the user submits a request to the Policy Manager to delete an existing security policy. The Policy Manager then notifies the Security Context Manager (SCM) to decommission the entire security service associated with that specific policy. This triggers a sequential termination workflow involving the Security Resource Orchestrator (S-RO) and the Security Closed Loop Manager (S-CLM) to ensure all components are cleanly removed from the target infrastructure. This decommissioning phase operates following a strict sequential enforcement logic, essentially acting as the reverse of the provisioning process. As reported in Figure 3-34, once the SCM receives the deletion request from the Policy Manager, it first requests the S-RO to terminate the deployed Security Functions (SFs). The S-RO removes and unconfigures these SFs from the Target Infrastructure and acknowledges the successful termination back to the SCM. Following the removal of the underlying SFs, the SCM triggers the S-CLM to terminate the associated security closed loops (S-CL). The S-CLM then instructs the S-RO to terminate the specific S-CL Functions from the infrastructure. Once these functions are successfully removed, the termination acknowledgments cascade back from the S-RO to the S-CLM, and finally to the SCM. To conclude the process, the SCM updates the internal service status to "terminated".

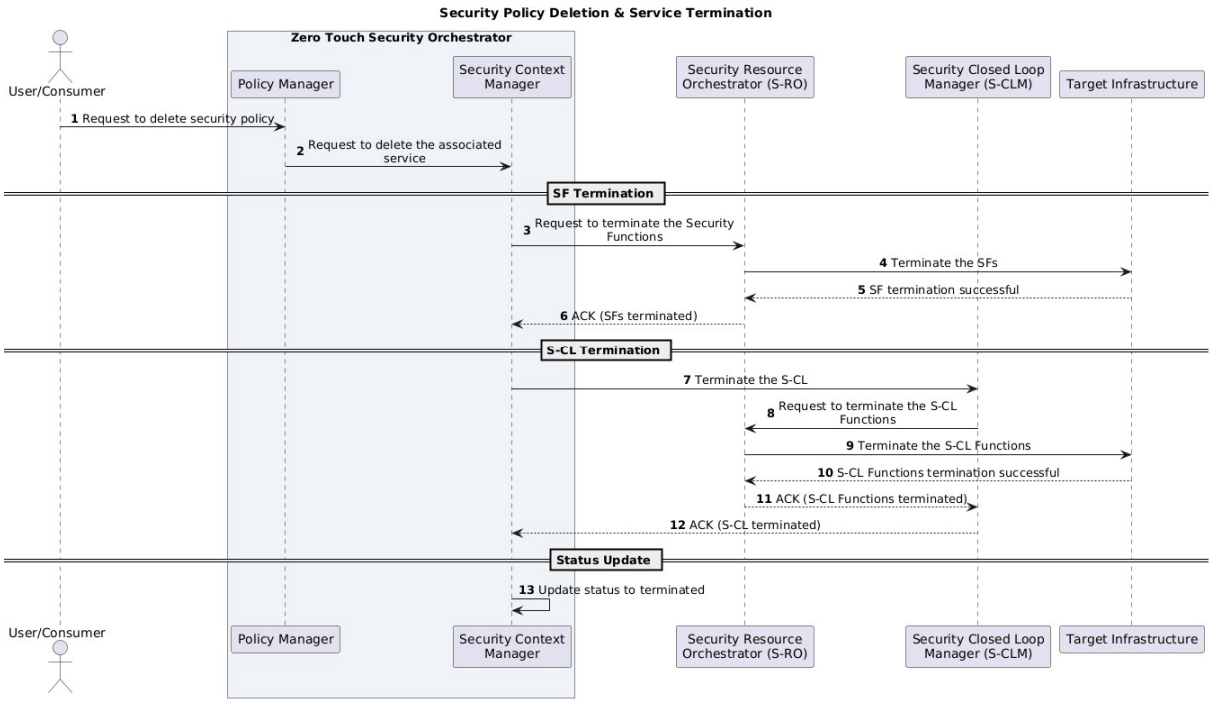


Figure 3-34 Security Service decommissioning workflow

As can be seen from Figure 3-35, after the process described in Figure 3-34 is completed, the associated service results TERMINATED, and so, also the Closed Loop associated as can be seen from Figure 3-36.

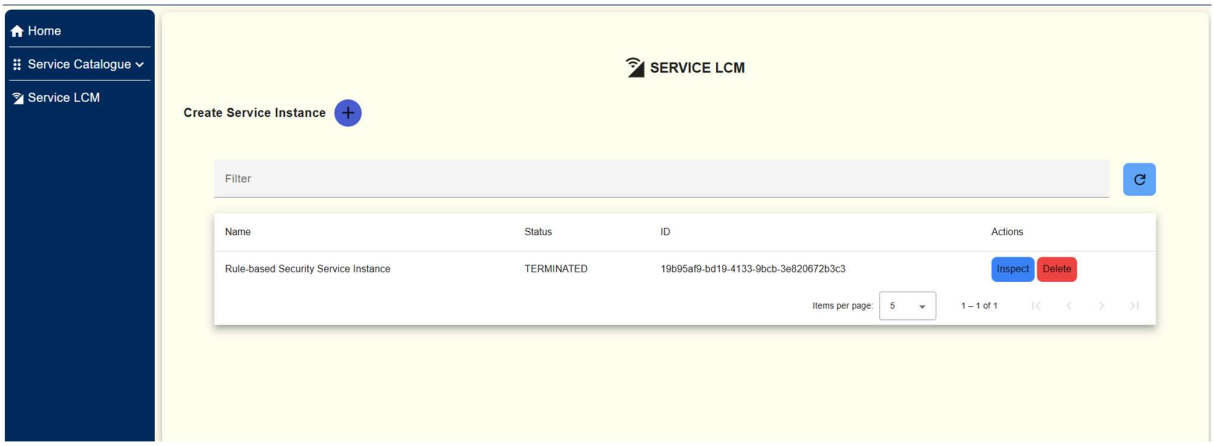


Figure 3-35 Terminated Service Instance

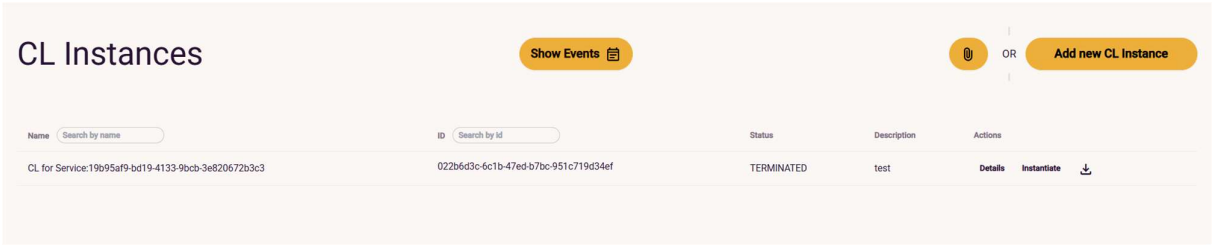


Figure 3-36 Terminated S-CL Instance

3.2 Security closed-loops execution examples

This section reports three examples of Security Closed Loops (S-CLs) operating within the context of a Security Service (SSe). We underline that these examples are not linked to the scenarios that will be part of the WP6 UC2. Rather, their scope is to demonstrate how S-CLs can provide rule-based detection (Section 3.2.1) as well as AI-based detection (Section 3.2.2) and prediction capabilities (Section 0). The following sections report the functional integration among the different components of SSes; however, this should be considered as a preliminary integration step. Consequently, the screenshots provided in the following sections reflect manual configurations and may contain discrepancies in terms of timestamps, IP addresses, or metadata. These variations exist because this initial validation was not performed in a unified testbed. A comprehensive, unified demonstration within a single environment will be presented as part of the WP6 Work Package and its associated Use Cases.

In this context, the tested Security Service is composed of the following Security Functions:

- **Programmable Monitoring Platform (PMP) (**Section 2.2.1): Configured to capture network traffic. In the rule-based scenario, it analyses the traffic directly to produce Snort alerts. For AI-based detection and prediction, it produces network flows in the CICFlowMeter format.
- **Threat Detection and Prediction Modules** (Sections 2.3.2 and 2.3.4): capable of ingesting data in the CICFlowMeter format and producing network alerts for detected or predicted attacks.
- **OpenC2 Consumer for Kubernetes (K8s):** Actuates on the K8s API to conduct several mitigation and investigation operations. Examples include QUERY pod (to discover which pod is creating malicious traffic given its cluster IP), DENY pod_network (to create a network policy blocking ingress and egress traffic), and DENY ip (to create a rule on the cluster ingress to blacklist a specific IP).
- **MQTT Broker:** Deployed once each time an OpenC2 consumer is selected as a security function. This ensures adherence to the Publish/Subscribe (PUB/SUB) pattern of the OpenC2 protocol and supports the coexistence of multiple consumers and producers.

These functions are coordinated and glued together through three distinct closed loops, which are composed of the following operational Security Closed Loop Functions:

- **Analysis:** Used only in the AI-based loops, this stage pulls the network flows produced by the PMP and provides them to the AI threat detection and prediction models for inference.
- **Knowledge:** Stores the remediation workflows and loop configuration parameters (e.g., PMP configurations, AI module configurations, and secrets).
- **Decision:** Given an alert raised by the analysis stage, this component selects and parses the correct remediation workflow from those stored in the knowledge function.
- **Execution:** Acts as a CACAO playbook interpreter and OpenC2 producer. It executes the playbook by storing and updating variables, delegating the final execution commands to the appropriate OpenC2 consumers.

These examples of loops have the goal of demonstrating, in a simplified scenario, how the established Security Service can maintain the security posture through rule- and AI-based detection and prediction of threats, followed by an automated response through CACAO playbooks and OpenC2 executions. The playbook used for these examples is a simple workflow for DoS remediation on Kubernetes, reported in Figure 3-37. The playbook is composed of the following three steps:

1. *Investigate IP***:** The OpenC2 producer asks the OpenC2 consumer to investigate the source IP address. This step determines whether the IP belongs to an internal cluster pod (internal attack) or originates from the outside by executing a query on the IP address.

2. *Branch Logic***:** If the investigated IP is internal, the workflow proceeds to Step 3a. Otherwise, it triggers Step 3b.

3. Based on the Branch logic

    a. (*Internal Mitigation*): A NetworkPolicy is created for the pod originating the attack, effectively isolating it by blocking all ingress and egress traffic.

b. (*External Mitigation*): A new rule is created on the cluster ingress (e.g., Nginx) to blacklist the external IP address.

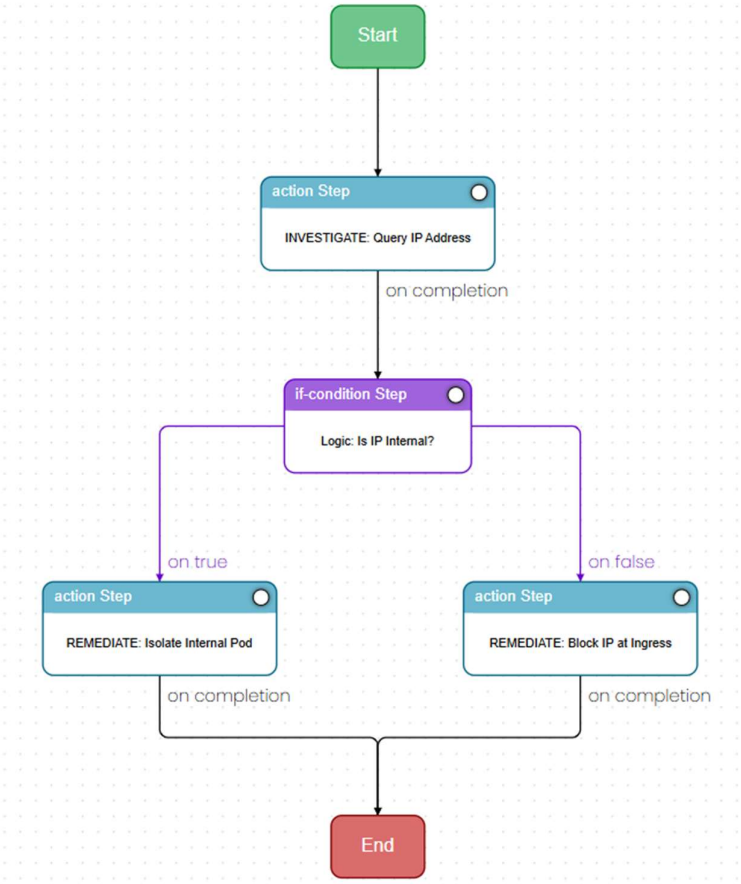


Figure 3-37 Example CACAO Playbook for DoS remediation

All these operations are made available through the OpenC2 K8s consumer. The complete set of supported operations, mapped by their OpenC2 Action and Target Resource, is further detailed in Table 3-1.

Table 3-1 Available Operations in the OpenC2 Kubernetes Consumer

OpenC2 Action	Target Resource	Description
query	pods	Queries a namespace to return a list of deployed pods along with their status, IP address, and node allocation.
query	ip_addr	Investigates a specific IP address by scanning the cluster to determine if it is assigned to an internal Pod or is external to the cluster.
get	pod	Retrieves the tail logs from a specified pod for observability and investigation.
set	deployment	Scales a specific deployment to a requested number of replicas.
restart	deployment	Triggers a rolling restart of a deployment by patching its annotations.
delete	pod	Executes a hard kill by deleting a specified pod from a namespace.
deny	pod	Isolates a targeted pod by dynamically generating a NetworkPolicy that blocks all ingress and egress traffic for its specific labels.
allow	pod	Removes a previously applied isolation NetworkPolicy, restoring standard network connectivity to the pod.
deny	ipv4_connection	Blocks an external source IP by patching the Ingress resource with a Nginx configuration snippet that returns an HTTP 403 Forbidden status.

update	ingress	Applies adaptive security responses by updating the Ingress resource to enable ModSecurity WAF rules or enforce request rate limiting.

3.2.1 Rule-based reactive loop

Figure 3-38 depicts the SSe that enables rule-based, zero-touch remediation to DoS attacks. As represented in the figure, in this type of SSe the only S-CL Functions that are needed are the *Decision, Execution* and *Knowledge*, while the *Monitoring* and *Analysis* stages are covered by the Programmable Monitoring Platform.

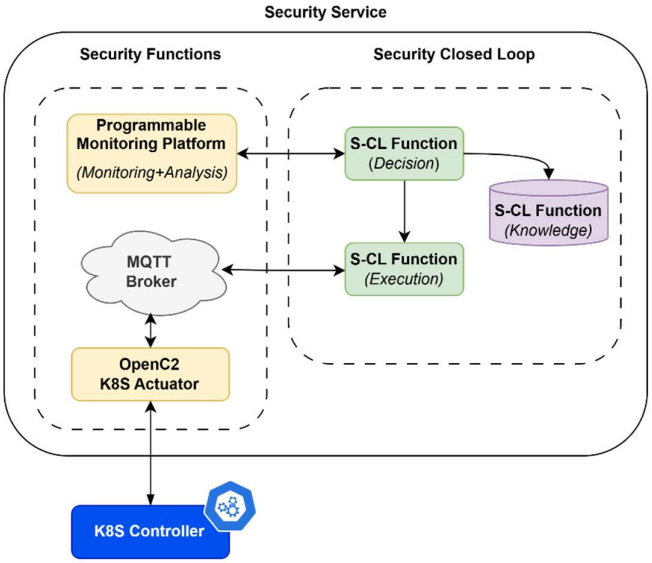


Figure 3-38 Security Service with rule-based Security Closed Loop

Figure 3-39 depicts the preliminary steps executed at SSe setup-time. In this case, the only setup step required concerns the PMP setup, which is performed through the PMP Configuration Manager API and is further described in this section. It is important to notice that, in the figure, the CLF-Decision function sets up the PMP and then starts listening for alerts; however, more generally, the ZTSO is responsible for configuring the PMP through the Configuration Manager API during different steps of service provisioning. At this stage of implementation, this is done through a dedicated API call from the CLF-Decision that, at setup time, retrieves the correct configuration parameters from the Knowledge Function based on the goal and performs the API call. Nevertheless, the logical configuration of the PMP is always the responsibility of the ZTSO and, in this case, is actuated through an S-CL Function. Another strategy currently under discussion, which will be explored during the demonstration of the ROBUST-6G UC2 in WP6, is the definition of an OpenC2 actuator for the PMP and the configuration from the SCM at service provisioning time through such an actuator.

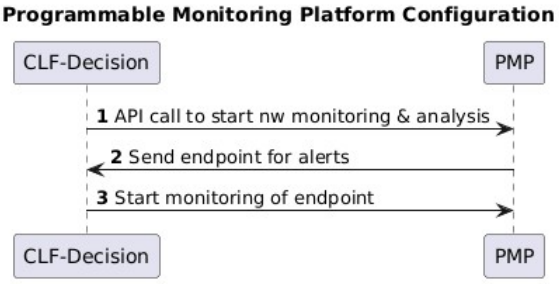


Figure 3-39 Programmable Monitoring Platform setup for monitoring and analysis

Steps 1 and 2 of the figure shows the request to deploy a monitoring and analysis security tool (e.g. Snort3) IDS to generate alerts located into the Alert Module. The HTTP POST request received by the endpoint */ConfigurationManager/DeploySecurityTool* includes parameters such as the pod (device) IP to be deployed,

a tool name, and the sigma rules or the configuration, only one of the two configuration options. A response is generated such as shows Table 3-2 with the IP:Port and the topic to consume the alerts using a Kafka consumer.

Table 3-2 Configuration Manager endpoint for security tool deployment

Operation name: *ConfigurationManager/DeploySecurityTool*		
Description	This endpoint allows the ZTSO to request the deployment of a security tool.	
Input Parameters	**Type**	**Description**
podIP	String	A string with the IP of the POD or the device.
toolName	String	Name of the security tool to be deployed.
sigma_rules	Array of String	List of configuration sigma rules.
Configuration	Array of String	List of configuration options using environment variables.
Output Parameters	**Type**	**Description**
200	String	Request successfully. You can collect the information in IP:Port -> kafka_robust6g-node1.lan:9094. Deployment: { “module”: “alert_module”, “tool”: “snort3”, “kafka_broker”: “kafka_robust6g-node1.lan:9094”, “topic” “snort_alerts”}
400	String	Bad request if payload is invalid.
500	String	Internal Server Error
Notes		
The configuration can only be set with sigma_rules or configuration parameters, not both.		

Once the Alert Module of the PMP is deployed, the Snort3 IDS system will trigger alerts when an attack is detected. This detection is notified through the topic name configured in the HTTP request, exposing the alerts in JSON format into the Kafka topic. The detection and notification technical process is further described in the following figures. Figure 3-40 shows the start of the Configuration Manager API, which is not yet containerised but works correctly for exchanging HTTP messages. It is implemented with FastAPI [Ram18] and Uvicorn [Uvi26] on port 8000, and the requests shown refer to Figure 3-41 and Figure 3-42. The first one is a health check of the API, and the second request is associated with the endpoint */ConfigurationManager/DeploySecurityTool* and the data from Figure 3-41. Figure 3-42 shows a response with the module and tool implemented, the IP:Port of the Kafka broker, and the name of the topic for consuming alerts.

```
user@user:~/Escritorio/ROBUST-6G_PMP$ python3 APIs/ConfigurationManager/configuration_manager_api.py
INFO:     Started server process [10558]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     127.0.0.1:48102 - "GET / HTTP/1.1" 200 OK
INFO:     127.0.0.1:48116 - "POST /ConfigurationManager/DeploySecurityTool HTTP/1.1" 200 OK
```

Figure 3-40 Implementation of the Configuration Manager API and request management for Alert Module

```
curl -X POST http://localhost:8000/ConfigurationManager/DeploySecurityTool -H "Content-Type: application/json" -d
'{
    "podIP": "10.0.2.15",
    "toolName": "snort3",
    "sigma_rules": [],
    "configuration": ["SNORT_KAFKA_TOPIC_OUT=snort_alerts","SNORT_ALERT_TAP_IFACE=tap0"]
}'
```

Figure 3-41 POST Request to deploy the Security Tool Snort3

```
user@user:~/Escritorio/ROBUST-6G_PMP$ sh Tests/configuration_manager_api_test.sh
Tests/configuration_manager_api_test.sh: 5:
Test script for Configuration Manager API endpoints
Execute this after starting the API
: not found
-e ================================
-e Testing Configuration Manager API
-e ================================

-e [1] Testing Health Check Endpoint...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   100  100   100    0     0  42087      0 --:--:-- --:--:-- --:--:-- 50000
{
  "message": "Configuration Manager API is running",
  "kafka_bootstrap": "kafka_robust6g-node1.lan:9094"
}
-e

-e [2] Testing DeploySecurityTool Endpoint...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   425  100   255  100   170     88     59  0:00:02  0:00:02 --:--:--   147
{
  "status": "success",
  "message": "Request successfully. You can collect the information in IP:PORT -> kafka_robust6g-node1.lan:9094",
  "deployment": {
    "module": "alert_module",
    "tool": "snort3",
    "kafka_broker": "kafka_robust6g-node1.lan:9094",
    "topic": "snort_alerts"
  }
}
```

Figure 3-42 Request and response for Security Tool Snort3

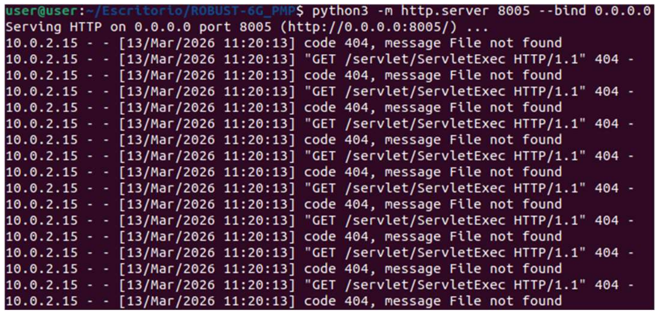
Once the request is received, the endpoint contacts the PMP backend to deploy the required tool, as demonstrated in Figure 3-43 with the *alert_module_robust6g* container. The remaining containers are already up and running, as indicated in the "Created" and "Status" columns.

```
user@user:~/Escritorio/ROBUST-6G_PMP$ sudo docker container ls
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS                    PORTS
NAMES
c81f62ce4b52   alert_module_robust6g:latest   "/usr/bin/python3 /h…"   51 seconds ago   Up 51 seconds
alert_module_robust6g
e04af87d00b9   elastic/filebeat:8.16.2        "/usr/bin/tini -- /u…"   17 minutes ago   Up 9 minutes
filebeat_robust6g
2c28fdd796d4   tshark_robust6g:latest         "/usr/bin/python3 /u…"   17 minutes ago   Up 9 minutes
tshark_robust6g
f3eb03c5ba03   mongo:4.4.29                   "docker-entrypoint.s…"   17 minutes ago   Up 9 minutes (healthy)    0.0.0.0:27017->2
mongodb_robust6g
3451266cc34a   apache/kafka:3.9.0             "/__cacert_entrypoin…"   17 minutes ago   Up 7 minutes (healthy)    0.0.0.0:9092->90
::]:9094->9094/tcp, 0.0.0.0:29092->29092/tcp, [::]:29092->29092/tcp   kafka_robust6g
19aa487dcd31   device_info_robust6g:latest    "python3 /usr/local/…"   17 minutes ago   Up 9 minutes              0.0.0.0:9999->99
device_info_robust6g
```

Figure 3-43 Deployment of the Alert Module container in the PMP environment

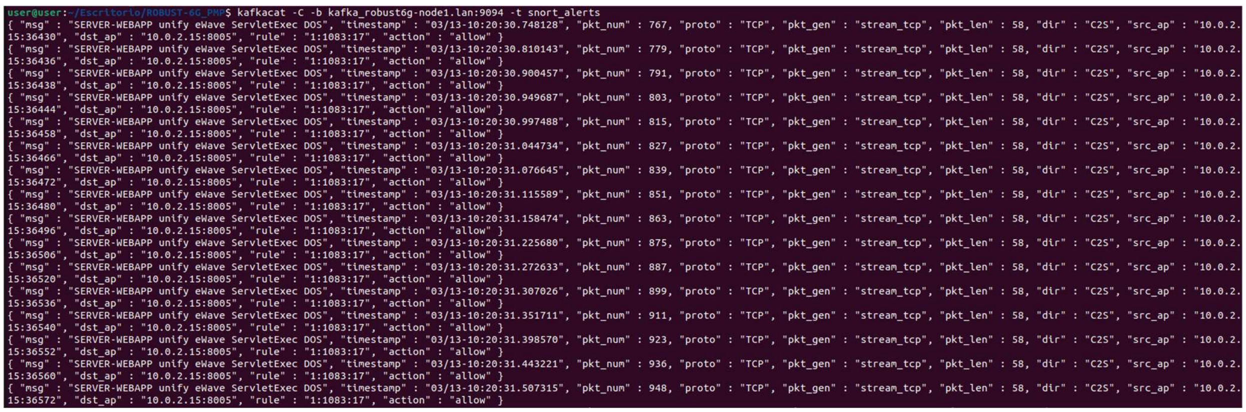
To simulate a DoS attack, a Python server is created on port 8005 and remains listening, as shown Figure 3-44. The system receives a large volume of HTTP request asking for a vulnerable resource [CVE20001025]. In this case, the attack is simulated and access to the resource is prevented so as not to overload the system; however bearing this in mind, a DoS occur in a real environment. In Figure 3-45 the Snort3 alert "msg:"SERVER-WEBAPP unify eWave ServletExec DOS"; flow:to_server,established; http_uri; content:"/servlet/ServletExec",fast_pattern,nocase; metadata:ruleset community; service:http; reference:bugtraq,1868; reference:cve,2000-1025; classtype:web-application-activity; sid:1083; rev:17;)" is triggered and presented for each request sent by the attacker. The attacker strikes at 11:20:13 (hours, minutes, seconds) in UTC+1 format, and the alerts are detected from 10:20:30 (hours, minutes, seconds) in UTC+0

format, which means that detection takes around 17 seconds. Part of this delay is due to transmission latency and hardware resource limitations, as the test is carried out on a virtual machine with minimum requirements.



```
user@user:~/Escritorio/ROBUST-6G_PMP$ python3 -m http.server 8005 --bind 0.0.0.0
Serving HTTP on 0.0.0.0 port 8005 (http://0.0.0.0:8005/) ...
10.0.2.15 - - [13/Mar/2026 11:20:13] code 404, message File not found
10.0.2.15 - - [13/Mar/2026 11:20:13] "GET /servlet/ServletExec HTTP/1.1" 404 -
10.0.2.15 - - [13/Mar/2026 11:20:13] code 404, message File not found
10.0.2.15 - - [13/Mar/2026 11:20:13] "GET /servlet/ServletExec HTTP/1.1" 404 -
10.0.2.15 - - [13/Mar/2026 11:20:13] code 404, message File not found
10.0.2.15 - - [13/Mar/2026 11:20:13] "GET /servlet/ServletExec HTTP/1.1" 404 -
10.0.2.15 - - [13/Mar/2026 11:20:13] code 404, message File not found
10.0.2.15 - - [13/Mar/2026 11:20:13] "GET /servlet/ServletExec HTTP/1.1" 404 -
10.0.2.15 - - [13/Mar/2026 11:20:13] code 404, message File not found
10.0.2.15 - - [13/Mar/2026 11:20:13] "GET /servlet/ServletExec HTTP/1.1" 404 -
10.0.2.15 - - [13/Mar/2026 11:20:13] code 404, message File not found
10.0.2.15 - - [13/Mar/2026 11:20:13] "GET /servlet/ServletExec HTTP/1.1" 404 -
10.0.2.15 - - [13/Mar/2026 11:20:13] code 404, message File not found
10.0.2.15 - - [13/Mar/2026 11:20:13] "GET /servlet/ServletExec HTTP/1.1" 404 -
10.0.2.15 - - [13/Mar/2026 11:20:13] code 404, message File not found
10.0.2.15 - - [13/Mar/2026 11:20:13] "GET /servlet/ServletExec HTTP/1.1" 404 -
10.0.2.15 - - [13/Mar/2026 11:20:13] code 404, message File not found
```

Figure 3-44 DoS ServletExec attack to HTTP server



```
user@user:~/Escritorio/ROBUST-6G_PMP$ kafkacat -C -b kafka_robust6g-node1.lan:9094 -t snort_alerts
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:30.748128", "pkt_num" : 767, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36430", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:30.810143", "pkt_num" : 779, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36436", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:30.900457", "pkt_num" : 791, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36438", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:30.949687", "pkt_num" : 803, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36444", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:30.997488", "pkt_num" : 815, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36458", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.044734", "pkt_num" : 827, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36466", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.076645", "pkt_num" : 839, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36472", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.115589", "pkt_num" : 851, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36480", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.158474", "pkt_num" : 863, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36496", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.225680", "pkt_num" : 875, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36506", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.272633", "pkt_num" : 887, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36520", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.307026", "pkt_num" : 899, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36536", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.351711", "pkt_num" : 911, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36540", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.398570", "pkt_num" : 923, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36552", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.443221", "pkt_num" : 936, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36560", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
{ "msg" : "SERVER-WEBAPP unify eWave ServletExec DOS", "timestamp" : "03/13-10:20:31.507315", "pkt_num" : 948, "proto" : "TCP", "pkt_gen" : "stream_tcp", "pkt_len" : 58, "dir" : "C2S", "src_ap" : "10.0.2.
15:36572", "dst_ap" : "10.0.2.15:8005", "rule" : "1:1083:17", "action" : "allow" }
```

Figure 3-45 DoS ServletExec alerts in Kafka broker

After the setup is complete, Figure 3-46 showcases the actual execution of the S-CL. As illustrated in Figure 3-47, the Decision function receives the alert from the PMP (in this case, a DoS alert). It then utilizes a key-value mapping to retrieve the appropriate remediation playbook from the Knowledge base. After validating the playbook, the Decision function passes it to the Execution function. Figure 3-48 displays the Execution function acting as an OpenC2 producer. It leverages an OpenC2 target agent to send all relevant OpenC2 commands to the deployed security functions during the mitigation process. Finally, once the mitigation is complete, a full execution trace is archived in the Knowledge. Figure 3-49 showcases the resulting network policy created to block the attacker by the Execution function during this mitigation.

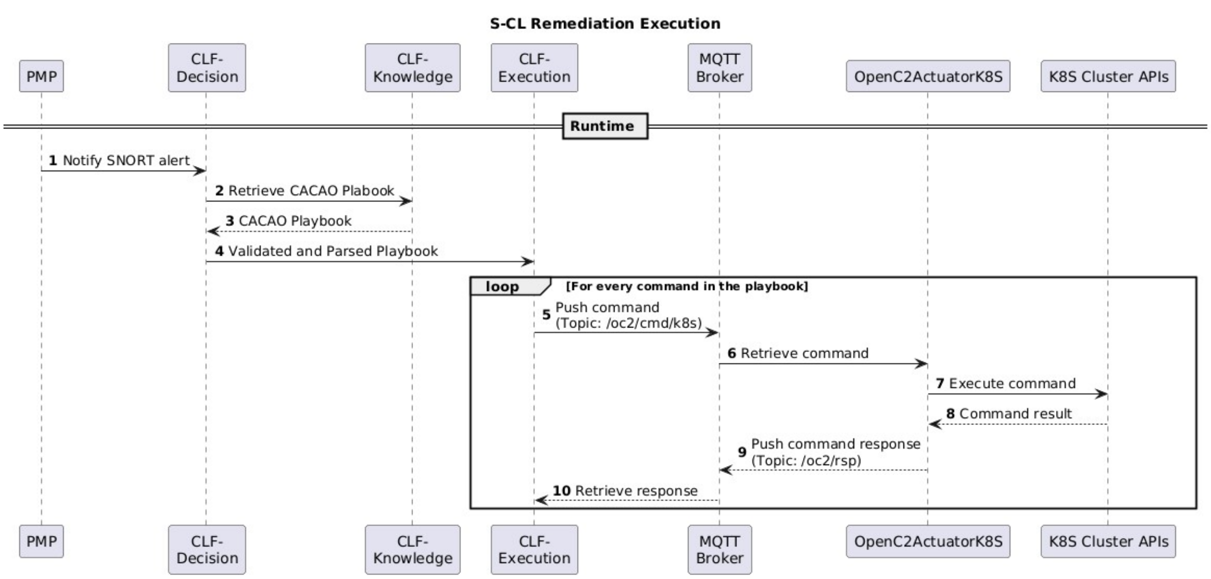


Figure 3-46 Rule Based S-CL Remediation Execution

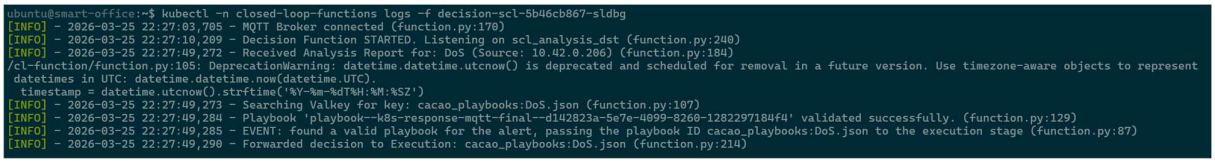


Figure 3-47 Decision Function logs - Playbook retrieval and validation

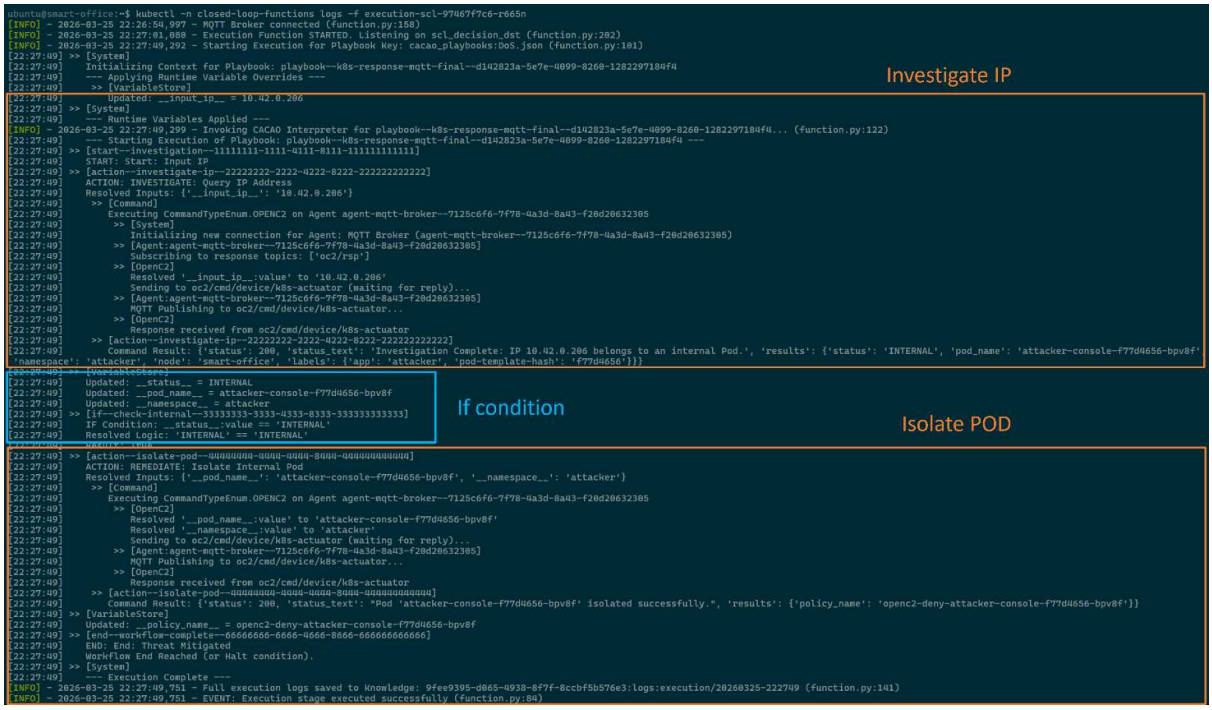


Figure 3-48 Execution Function logs - Playbook execution with OpenC2

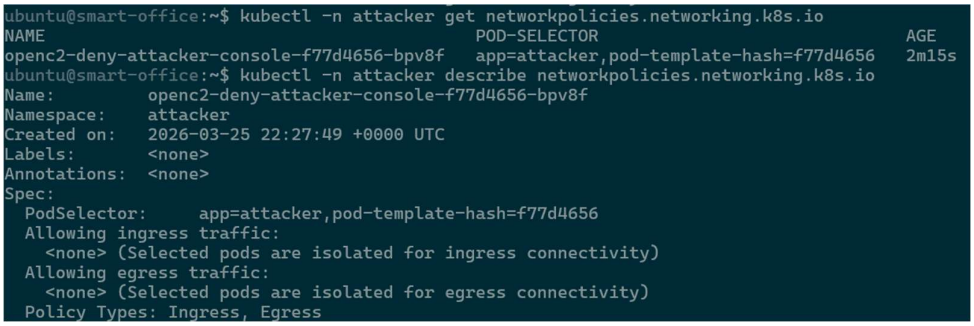


Figure 3-49 Results of the remediation - Network Policy generated

3.2.2 AI-driven reactive loop

Figure 3-50 shows the SSe that provides AI-based threat detection and zero-touch remediation to DoS attacks. As represented in this figure, for this type of SSe, with respect to the rule-based reactive loop, the *Analysis* function was added as intermediate step in between the *Monitoring*, covered by the Programmable Monitoring Platform, and the *Decision.* The role of the *Analysis* function is to act as intermediary in between the data generated from the PMP and the Threat Detection Module deployed and used as analysis algorithms. The steps of *Decision*, *Knowledge* and *Execution* are the same of the rule-based reactive loop, and so, this section will mostly focus on the phases from raw data ingestion to AI-driven detection of the threat.

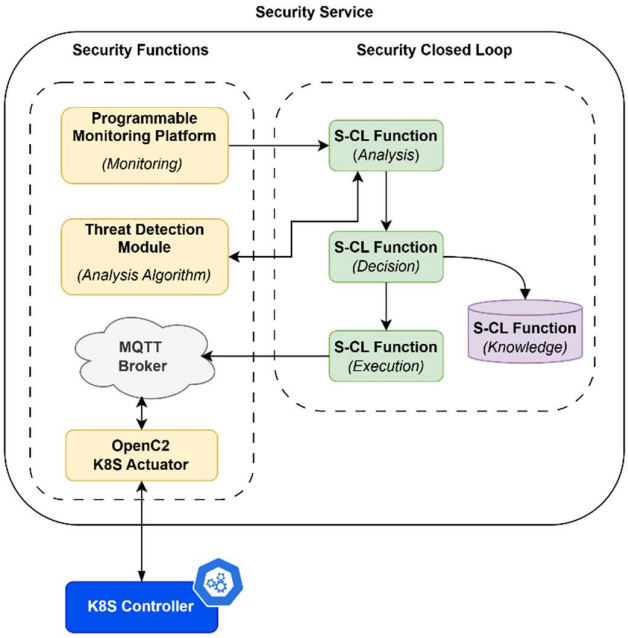


Figure 3-50 Security Service with AI-based detection Security Closed Loop

As in the previous section, the PMP needs to be properly configured to enable the collection and exposure of network flows that are compatible with the input format of the Threat Detection Module. Differently from the rule-based loop, this time is the *Analysis* function that oversees this configuration, but as previously discussed, is always the ZTSO that logically performs this configuration at setup time.

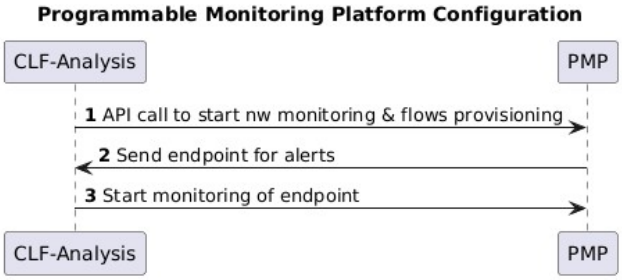


Figure 3-51 Programmable Monitoring Platform setup for monitoring

Step 1 and 2 shows how the endpoint */ConfigurationManager/DeployFlowTool* behaves when a request is made from the ZTSO to deploy the Flow Module, using the nomenclature of Table 3-3 to be configured. The endpoint responds with the appropriate message and information about where data can be extracted. In this case, the tool used to generate the network flows is CICFlowMeter, but internally it is also referred to as the general module, hence "flow_module" is used as both the module and the tool.

Table 3-3 Configuration Manager endpoint for flow tool deployment

Operation name: */ConfigurationManager/DeployFlowTool*		
Description	This endpoint allows the ZTSO to request the deployment of a flow generator tool.	
Input Parameters	**Type**	**Description**
podIP	String	A string with the IP of the POD or the device.
toolName	String	Name of the security tool to be deployed.
sigma_rules	Array of String	List of configuration sigma rules.

	Configuration	Array of String	List of configuration options using environment variables.
Output Parameters		**Type**	**Description**
	200	String	Request successfully. You can collect the information in IP:Port -> kafka_robust6g-node1.lan:9094. Deployment: { “module”: “flow_module”, “tool”: “flow_module”, “kafka_broker”: “kafka_robust6g-node1.lan:9094”, “topic” “cic_flow”}
	400	String	Bad request if payload is invalid.
	500	String	Internal Server Error
Notes			
The configuration can only be set with sigma_rules or configuration parameters, not both.			

As in the previous Section, the Configuration Manager API is initiated to receive an HTTP deployment message, as shown in Figure 3-52. Figure 3-53 shows the HTTP request made by the ZTSO to the endpoint */ConfigurationManager/DeployFlowTool* requesting the deployment of the “flow_module” tool, which has the same name as the module, even though CICFlowMeter is implemented as the main flow analysis tool. Figure 3-54 shows the endpoint’s response with the Kafka broker and the topic for consuming the flows.

```
user@user:~/Escritorio/ROBUST-6G_PMP$ python3 APIs/ConfigurationManager/configuration_manager_api.py
INFO:     Started server process [19874]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     127.0.0.1:49038 - "GET / HTTP/1.1" 200 OK
INFO:     127.0.0.1:49048 - "POST /ConfigurationManager/DeployFlowTool HTTP/1.1" 200 OK
```

Figure 3-52 Implementation of the Configuration Manager API and request management for Flow Module

```
curl -X POST http://localhost:8000/ConfigurationManager/DeployFlowTool -H "Content-Type: application/json" -d
'{
    "podIP": "10.0.2.15",
    "toolName": "flow_module",
    "sigma_rules": [],
    "configuration": ["CIC_KAFKA_BASE_TOPIC_OUT=cic_flow"]
}'
```

Figure 3-53 POST Request to deploy the Flow Tool flow_module

```
user@user:~/Escritorio/ROBUST-6G_PMP$ sh Tests/configuration_manager_api_test.sh
Tests/configuration_manager_api_test.sh: 5:
Test script for Configuration Manager API endpoints
Execute this after starting the API
: not found
-e ================================
-e Testing Configuration Manager API
-e ================================

-e [1] Testing Health Check Endpoint...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   100  100   100    0     0   5153      0 --:--:-- --:--:-- --:--:--  5263
{
  "message": "Configuration Manager API is running",
  "kafka_bootstrap": "kafka_robust6g-node1.lan:9094"
}
-e

-e [3] Testing DeployFlowTool Endpoint...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   395  100   255  100   140     84     46  0:00:03  0:00:03 --:--:--   130
{
  "status": "success",
  "message": "Request successfully. You can collect the information in IP:PORT -> kafka_robust6g-node1.lan:9094",
  "deployment": {
    "module": "flow_module",
    "tool": "flow_module",
    "kafka_broker": "kafka_robust6g-node1.lan:9094",
    "topic": "cic_flow"
  }
}
```

Figure 3-54 Request and response for Flow Tool flow_module

Figure 3-55 shows a successfully deployment of the Flow Module. The remaining containers are already up and running, as indicated in the “Created” and “Status” columns. Finally, Figure 3-56 shows the data flows generated by the CICFlowMeter analysis, published in CSV format to the relevant Kafka broker using the topic name.

```
user@user:~/Escritorio/ROBUST-6G_PMP$ sudo docker container ls
CONTAINER ID   IMAGE                         COMMAND                  CREATED          STATUS                    PORTS
                                                                      NAMES
a681290c5d98   flow_module_robust6g:latest   "/usr/bin/python3 /h…"   36 seconds ago   Up 35 seconds
                                                                      flow_module_robust6g
e04af87d00b9   elastic/filebeat:8.16.2       "/usr/bin/tini -- /u…"   36 minutes ago   Up 28 minutes
                                                                      filebeat_robust6g
2c28fdd796d4   tshark_robust6g:latest        "/usr/bin/python3 /u…"   36 minutes ago   Up 28 minutes
                                                                      tshark_robust6g
f3eb03c5ba03   mongo:4.4.29                  "docker-entrypoint.s…"   36 minutes ago   Up 28 minutes (healthy)   0.0.0.0:27017->27017/tcp, [::]:27017->27017/tcp
                                                                      mongodb_robust6g
3451266cc34a   apache/kafka:3.9.0            "/__cacert_entrypoin…"   36 minutes ago   Up 26 minutes (healthy)   0.0.0.0:9092->9092/tcp, [::]:9092->9092/tcp, 0.0.0.0:9094->9094/tcp, [
::]:9094->9094/tcp, 0.0.0.0:29092->29092/tcp, [::]:29092->29092/tcp   kafka_robust6g
19aa487dcd31   device_info_robust6g:latest   "python3 /usr/local/…"   36 minutes ago   Up 28 minutes             0.0.0.0:9999->9999/tcp, [::]:9999->9999/tcp
                                                                      device_info_robust6g
```

Figure 3-55 Deployment of the Flow Module container in the PMP environment

```
user@user:~/Escritorio/ROBUST-6G_PMP$ kafkacat -C -b kafka_robust6g-node1.lan:9094 -t cic_flow
127.0.0.1,127.0.0.53,64064,53,17,2026-03-12 09:47:50,52748.0,5516.796845378023,37.91612952149844,18.95806476074922,18.95806476074922,1,1,92,199,92,92,92.0,0.0,199,199,199.0,0.0,199,92,145.5,53.5,2862.25,8
,8,8,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1.0,145.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,92.0,199.0,0,1,1,92,199

127.0.0.1,127.0.0.53,48421,53,17,2026-03-12 09:47:50,7801.0,39353.92898346366,256.37738751442123,128.18869375721061,128.18869375721061,1,1,92,215,92,92,92.0,0.0,215,215,215.0,0.0,215,92,153.5,61.5,3782.25
,8,8,8,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1.0,153.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,92.0,215.0,0,1,1,92,215

10.0.2.15,155.54.1.1,56552,53,17,2026-03-12 09:47:50,0.0,0,0,0,0,1,0,103,0,103,103,103.0,0.0,0,0,0,0,0.0,103,103,103.0,0.0,0,0.0,8,0,8,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.0,103.0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,103.0,0,0,1,0,103,0

10.0.2.15,155.54.1.1,60876,53,17,2026-03-12 09:47:50,0.0,0,0,0,0,1,0,103,0,103,103,103.0,0.0,0,0,0,0,0.0,103,103,103.0,0.0,0,0.0,8,0,8,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.0,103.0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,103.0,0,0,1,0,103,0

155.54.1.1,10.0.2.15,53,60876,17,2026-03-12 09:47:50,0.0,0,0,0,0,1,0,226,0,226,226,226.0,0.0,0,0,0,0,0.0,226,226,226.0,0.0,0,0.0,8,0,8,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.0,226.0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,226.0,0,0,1,0,226,0

10.0.2.15,40.79.197.35,40196,443,6,2026-03-12 09:47:50,252923.99999999997,743.3062896364125,11.861270579304456,7.907513719536304,3.953756859768152,2,1,128,60,74,54,64.0,10.0,60,60,60.0,0.0,74,54,62.666666
666666664,8.379870059984356,70.2222222222221,40,20,20,0,126462.0,252577.0,347.0,126115.0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,2,0,0,0.5,62.666666666666664,64240,65535,0,0,0,0,0,0,0,0,0,0,0,0,0,64.0,60.0
,0,2,1,128,60

155.54.1.1,10.0.2.15,53,56552,17,2026-03-12 09:47:50,0.0,0,0,0,0,1,0,210,0,210,210,210.0,0.0,0,0,0,0,0.0,210,210,210.0,0.0,0,0.0,8,0,8,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.0,210.0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,210.0,0,0,1,0,210,0

10.0.2.15,155.54.1.1,45093,53,17,2026-03-12 09:47:50,24687.0,9397.658686758212,81.0142990237777,40.50714951188885,40.50714951188885,1,1,116,116,116,116,116.0,0.0,116,116,116.0,0.0,116,116,116.0,0.0,0,0.0,8,
8,8,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1.0,116.0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,116.0,116.0,0,1,1,116,116

10.0.2.15,40.79.197.35,40210,443,6,2026-03-12 09:47:50,0.0,0,0,0,0,1,0,74,0,74,74,74.0,0.0,0,0,0,0,0.0,74,74,74.0,0.0,0,0.0,20,0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0.0,74.0,64240,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,74.0,0,0,1,0,74,0
```

Figure 3-56 Network flows in Kafka broker

Figure 3-57 depicts the AI-Driven reactive S-CL execution, starting from the collection of network traces from the PMP and finishing with automated execution of a CACAO remediation playbook though OpenC2. While the steps after step 4 are exactly the same of the ones described in Section 3.2.1, the real difference in this loop is the integration of the Analysis function and the threat Detection modules.

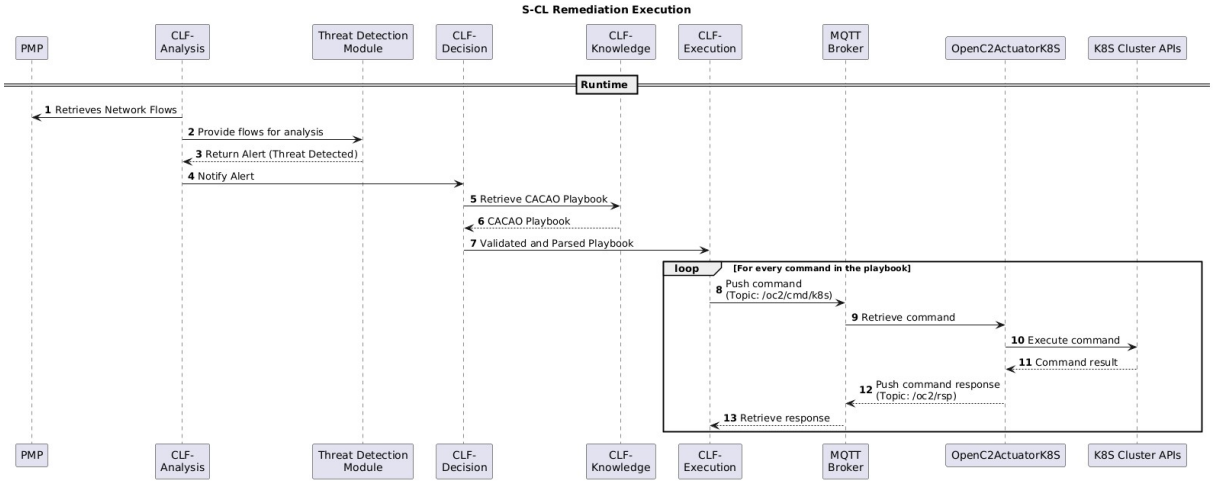


Figure 3-57 AI-Driven reactive S-CL Remediation Execution

In this S-CL, the scope of the Analysis function is to retrieve periodically (based on parameters provided at configuration time) the network flows from the PMP, in the format depicted in Figure 3-56 and provide them to the Threat Detection Module, on the */infer* endpoint. Figure 3-58 reports about the Threat Detection module deployed as Security Function and Figure 3-59 reports the logs of the S-CL Function of analysis retrieving the data from the PMP and passing them to the Threat Detection module for inference. Once this alert is retrieved, steps from 4 to 13 are the same ones described in Section 3.2.1.

```
ubuntu@smart-office:~$ kubectl -n security-functions get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/threat-detector-6dc545456f-4m465    1/1     Running   0          8m1s

NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/threat-detector    ClusterIP   10.43.209.179   <none>        8000/TCP   8m1s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/threat-detector    1/1     1            1           8m1s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/threat-detector-6dc545456f    1         1         1       8m1s
```

Figure 3-58 Threat detector module deployed as Security Function

```
ubuntu@smart-office:~/security-functions$ kubectl -n closed-loop-functions logs -f deployment/analysis-scl-5b46cb867-sl0bg -c axon-module
[INFO] - 2026-03-26 10:41:22.005Z - MQTT Broker connected (analysis_function.py:91)
[INFO] - 2026-03-26 10:41:22.005Z - Analysis Function STARTED. (analysis_function.py:92)
[INFO] - 2026-03-26 10:41:22.005Z - Listening to the PMP for flows (analysis_function.py:76)
[INFO] - 2026-03-26 10:41:27.006Z - Flows retrieved, passing to analysis function (analysis_function.py:86)
[INFO] - 2026-03-26 10:41:27.516Z - Analysis Report for [{"Flow ID": "<M>", "Src Port": 48994, "Dst Port": 53, "Src IP": "192.168.1.31", "Dst IP": "192.168.1.1", "Timestamp"
: "2019-04-25T09:12:56", "Protocol": "UDP", "Detected_Attack": "DoS"}] (analysis_function.py:52)
[INFO] - 2026-03-26 10:41:27.517Z - Parsed Alert for DoS, src IP 192.168.1.31 - passing to the Decision Function [dst=192.168.1.1 src_port=48994 dst_port=53 proto=UDP flow_t
s=2019-04-25T09:12:56] (analysis_function.py:63)
```

Figure 3-59 Logs of Analysis Function using the Threat Detection Module

3.2.3 AI-driven predictive loop

Figure 3-60 depicts the SSe that provides AI-based threat prediction and zero-touch remediation to DoS attacks. As shown in the figure, in this type of SSe, with respect to the AI-based threat detection reactive loop the only component that changed is the Threat Detection module that was substituted by the Threat Prediction one.

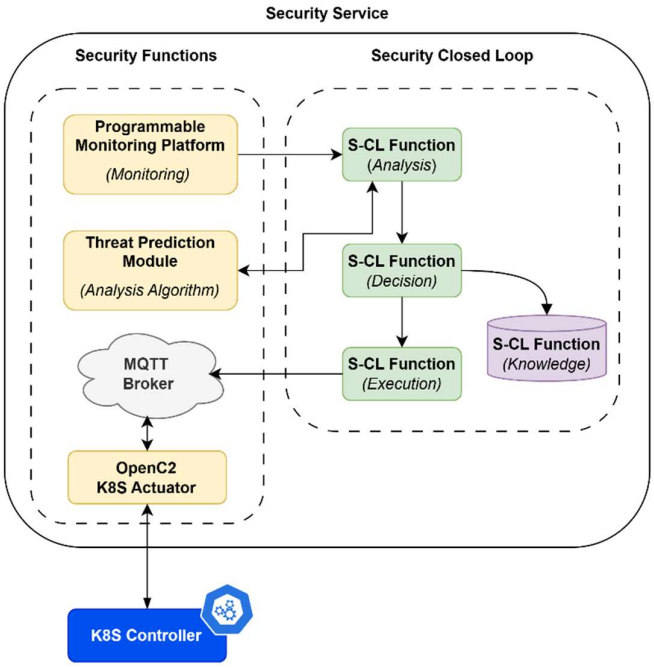


Figure 3-60 Security Service with AI-based predictive Security Closed Loop

Figure 3-57 depicts the AI-Driven predictive S-CL execution, starting from the collection of network traces from the PMP and finishing with automated execution of a CACAO remediation playbook though OpenC2. In this case, the PMP configuration is exactly the same reported in the previous section, and also, all the steps from 4 to 13 are the same of the previous sections.

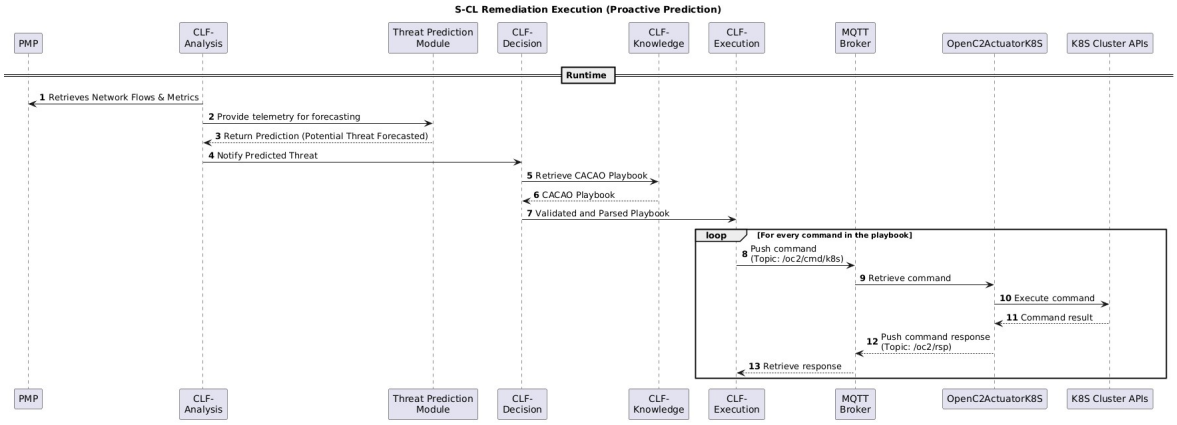


Figure 3-61 AI-Driven predictive S-CL Remediation Execution

The only thing that differs with respect to the previous example is the role of the Analysis function and the threat prediction module. In this S-CL, the scope of the Analysis function is to retrieve every five minutes (based on parameters provided at configuration time and on the characteristics of the threat prediction module described in Section 2.3.4) the network flows from the PMP, in the format depicted in Figure 3-56 and provide them to the Threat Prediction Module, on the */infer* endpoint. Figure 3-62 reports about the Threat Prediction module deployed as Security Function and Figure 3-63 reports the logs of the S-CL Function of analysis retrieving the data from the PMP and passing them to the Threat Prediction module for inference. Once this alert is retrieved, steps from 4 to 13 are the same ones described in Section 3.2.1. Is it worth to notice that future attack prediction is based on a binary relevance algorithm, where each threat type is detected independently, including the benign class. Both the benign and DoS labels may take a value of 1 simultaneously, indicating that benign and DoS traffic coexist within the same interval.

```
ubuntu@smart-office:~/security-functions$ kubectl -n security-functions get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/threat-predictor-6b654fdb85-hk8k9   1/1     Running   0          6s

NAME                        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/threat-predictor    ClusterIP   10.43.155.77   <none>        8000/TCP   7s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/threat-predictor    1/1     1            1           7s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/threat-predictor-6b654fdb85    1         1         1       7s
```

Figure 3-62 Threat Prediction module deployed as Security Function

```
ubuntu@smart-office:~/security-functions$ kubectl -n closed-loop-functions logs -f analysis-scl-4j78fg561-lu1bd-b888cfb5d-vwc55
Defaulted container "axon-module" out of: axon-module, pip-install (init)
[INFO] - 2026-03-26 11:02:32.720Z - MQTT Broker connected (analysis_function.py:64)
[INFO] - 2026-03-26 11:02:32.720Z - Analysis Function STARTED. (analysis_function.py:65)
[INFO] - 2026-03-26 11:02:32.721Z - Listening to the PMP for flows (analysis_function.py:72)
[INFO] - 2026-03-26 11:02:37.908Z - Flows retrieved, passing to prediction function (analysis_function.py:82)
[INFO] - 2026-03-26 11:02:40.968Z - Prediction Report for [{"Timestamp": "2019-04-25T09:17:55", "Future_Attack_Type_Access": 0, "Future_Attack_Type_Cryptomining": 0, "Future
_Attack_Type_DDoS": 0, "Future_Attack_Type_DoS": 1, "Future_Attack_Type_MITM": 0, "Future_Attack_Type_Malware": 0, "Future_Attack_Type_Reconnaissance": 0, "Future_Attack_Typ
e_Web": 0, "Future_Attack_Type_Benign": 1}] (analysis_function.py:47)
[INFO] - 2026-03-26 11:02:40.968Z - Predicted possible DoS attack [ts=2019-04-25T09:17:55] - passing to the Decision Function (analysis_function.py:57)
```

Figure 3-63 Logs of Analysis Function using the Threat Predictor Module

3.3 Additional Considerations on the ZTSP

As already discussed in Sections 3.1 and 3.2, the primary goal of Section 3 is to showcase practical, functional integration execution examples of the ZTSP. It is important to emphasize that these are preliminary examples and are not strictly linked to the final use case scenarios. Furthermore, these sections demonstrated three baseline models of security closed loops: rule-based detection, AI-based detection, and AI-based predictive loops. These models serve as the foundation for the comprehensive, unified demonstrations planned for WP6. Together, these loops validate the system's capacity to maintain a strong security posture and enable zero-touch automation within the provisioned security services. Additional considerations regarding the deployment and capabilities of the ZTSO include:

- **Integration of Physical (PHY) Layer Security Components:** while the current execution examples do not explicitly feature PHY layer security components, incorporating them is primarily a matter of modelling within the ZTSO's KG. By defining OpenC2 actuators within the KG and the catalogue, the system can orchestrate actuators for any component that exposes a set of APIs. A concrete example of this PHY-level integration will be fully detailed and demonstrated in WP6.

- **Analysis Algorithm Flexibility:** the examples currently utilize the specific Threat Detection and Threat Prediction modules developed in WP4. However, the architecture is inherently agnostic; any compatible AI algorithm can be loaded into the catalogue, respecting the KG schema, and the S-RO for runtime orchestration. Future work in WP6 will highlight this flexibility by showcasing the integration between the ZTSO and the ROBUST-6G Global Model Repository (GMR) for model retrieval.

- **Adaptability of SCL Mechanisms:** the role of the S-CL Functions, as explained in Section 3.2, is highly flexible and can be tailored to various use cases and operational settings. Depending on the scenario, a S-CL can deploy up to four operational S-CL Functions (Monitoring, Analysis, Decision, and Execution) to act as intermediate coordination between Security Functions. An example is in the rule-based loop, where the analysis simply collects the analysis alerts from the PMP, or in the AI-based loop, where the analysis stage acts as an intermediary stage between the data source and the actual analysis algorithm. In this context, the decision function and/or the execution function can also be linked to other security functions: for instance, the decision module can actually make use of the Network Attack Mitigation module presented in Section 2.3.3, while the execution function in all the presented loops acts as an OpenC2 producer, outsourcing the actuation to external OpenC2 consumers.

4 ROBUST-6G Objective 4 fulfilment

The Zero-Touch Security Platform and the related works have been designed and developed in line with ROBUST-6G Objectives. In particular, WP4 is committed to fulfilling Objective 4:

"Automatic, zero-touch, security, and resource management for trusted and certified services among multiple stakeholders in distributed dynamic scenarios"

Each ROBUST-6G objective includes a set of Quantifiable Targets (QT). Part of them is addressed in this document, the rest in WP6, as several functionalities need to be demonstrated through the project's PoCs. A global overview of WP4 achievements will be provided in the Final Management Report at the end of the Project (M30). Table 4-1 below lists the entries addressed by this document.

Table 4-1 WP4 Quantifiable Targets

QT ID	Description	Status*	How	Reference
4.1	Increase the security orchestration efficiency by 10% through the use of AI.	A	No quantified baseline could be found in the literature to compare our own metrics measurement. This lack of data is due to closed source datasets from studies and papers. Our solution to fulfil this objective is to rely on the WP6 Use case 2 demonstrator to measure the efficiency metrics with the usage of AI building blocks of WP4, then without it. In this way, we will be able to compare the gain of efficiency based on our own implementations.	See the clarification paragraph below the table
4.2	Ensure that the Holistic Control, Orchestration, and Management plane provides a quality of service that meets or exceeds the demanded levels in the security SLAs at least 95% of cases.	A	A security service is not provisioned if the target environment cannot guarantee the adequate security posture.	[R6G24-D41], [R6G25-D43], Section 2.1.1
4.3	Improve the efficiency of the security management system by reducing the response time to potential threats by at least 30% using AI and explainable AI.	A	The Robust 6G ZTSP integrates AI agents based on large language model in GenAISOAR, along with AI/ML-based detection and mitigation models, to enable real-time threat response and short-term attack prediction. Although validation is ongoing, the approach is expected to improve remediation efficiency by about 30%, consistent with Bono et al. [Bono24].	Sections 2.1.2, 2.3
4.4	Specification and implementation of pervasive cognitive security closed loops across several domains in the fashion of a nervous	A	Cognitive (AI-driven) CLs implemented have been implemented. The loops are considered multi-target having	Section 3.2

	system with a support of reflexes, peripheral, and central closed loops.		their actuation based on OpenC2.	
4.5	Number of Managed domains > 3	PA	Edge/FarEdge and Cloud domains are targeted through the S-RO, Physical Layer Security management will be addressed in the ongoing integrations of WP6.	Section 2.1.4
4.6	Minimum number of closed-loops coordinated = 5	PA	CL coordination designed and implemented but not fully operationale. Collaboration between Loops to be demonstrated in UC2 PoCs.	[R6G25-D43] Section 2.1.1

*** A=Achieved; NA= Not Achieved; PA=Partially Achieved.**

To clarify objective 4.1, the definition of "efficiency" was taken from [Enoch18], which defines metrics used to measure efficient orchestration:

- Mean Time to Detect an Incident (MTTD)
- Mean Time to Respond (MTTR)
- Volume of Alerts Analysed and Unanalysed (number per day)
- Volume of tickets closed (number per day)
- Volume of incidents detected within (per pre-defined timeframe using timestamps)
- Mean time spent on operations by analysts
- Mean time spent on each ticket by analysts

Adding to these ones, we can also add true-positive rate for accuracy and measure all these metrics to compare our zero-touch security architecture with a baseline available in the literature.

Beyond the objective 4 and its QT, WP4 is also committed to fulfil a set of sub-objectives specific for the work package (See Table 4-2 below).

Table 4-2 WP4 sub-objectives fulfilment

SO ID	Description	Status*	How	Reference
4.1	Provide an advanced AI/ML-driven platform to orchestrate security in 6G system, implementing Zero-Touch and Security-as-a-Service paradigms.	A	Policy Management, Semantic modeling, AI-driven orchestration and remediations, Zero-touch automation	Sections 2.1.1, 2.1.2, 2.1.3
4.2	Implement a threat detection and alarm notification mechanisms based on pervasive monitoring of 6G systems	A	PMP, Semantic-Aware/Rule-based Anomaly detection, dedicated CLs (Analysis stage)	Sections 2.2.1, 2.2.2, 2.2.3
4.3	Implement a threat-mitigation mechanism that can adapt to the different forms of attacks, driven by AI/ML predictions and inputs from all layers	A	AI-driven Threat prediction and mitigation modules. Inputs from different layers collected through the Programmable Monitoring Platform.	Sections 2.3.1, 2.3.2, 2.2.1.
4.4	Guarantee a high-level of efficiency and robustness in resource management in the case of threats towards services and applications	A	Resource Orchestrator, Risk-averse Resource Management Framework	Sections 2.1.4, 2.1.5

*** A=Achieved; NA= Not Achieved; PA=Partially Achieved.**

5 Conclusions

This document presented the final advancements of the Zero-Touch Security Platform at M27 (the end of WP4), covering several aspects related to the design, implementation, and integration of the different platform's module. In the initial part, the design and the implementation has been finalised per each module of the platform. In particular, the document focused on reporting the deltas of those two aspects with respect to the status reported in D4.3.

The second part of the document has been dedicated to the integration of the different modules. In the sense, a set of operation workflows has been defined in in form of sequence diagrams, covering the set of steps required for i) provisioning a security service (proactive security orchestration) and security service decommissioning, and ii) implementing security automation through security CLs (reactive/predictive orchestration). In particular, the document provided three examples of CLs, one reactive rule-based and two AI-based, one of them exploiting predictive inference. A description of the different steps has been provided for each sequence diagram and, most important, the actual interactions between the different modules has been demonstrated through a set of screenshots captured during the integration activities and tests.

What reported so far, including the original contribution of past WP4 deliverable (D4.1, D4.3), represents the concrete realisation of the 4 key functionalities (or pillars) of the ZTSP identified by the work package to achieve its internal and project objectives: i) security service and resource orchestration, ii) programmable pervasive monitoring, iii) threat/anomaly detection, and prediction iv) closed-loop-based security automation. Security Service Orchestrator, Resource Orchestrator, and Programmable Monitoring Platform are the components that implement (i) and (ii). Their design and implementation have been finalised, complemented with a new set of functionalities not previously reported and their interactions have also been demonstrated. Furthermore, the contribution of these set of components is complemented by the scientific works related to the risk-adverse resource management. Pillar (iii) is achieved by the implementation of specific AI-driven modules i.e., the Threat Mitigation Modules, that provides both anomaly reactive and predictive inference and the Threat Mitigation Module which allow the dynamic selection of the mitigation actions. The PMP must also be considered in this sense as it provides rule-based anomaly detection functionalities, along with the other scientific work related to the semantic-aware anomaly detection. The Security CL management modules, PMP, AI-driven anomaly detection/prediction, and a set of dedicated CL stages implements the fourth pillar: the ZTSP implemented and demonstrated the capability of creating dynamically security CLs as part of given security services.

The WP4 achievements discussed in this document are in line with the project's objectives (Objective 4 in particular), KPIs, and work package sub-objectives. In this regard, not all the Quantifiable Targets (KPIs) defined in WP4 have been addressed in this document. In part due to the fact that some of them overlap with those related to UC2, in part because in some cases a dedicated measurement campaign would be required. In both cases, the aim is to achieve the remaining QTs in the context of WP6, where the UCs will be demonstrated. In this regard, more integration activities, test and measurement campaigns are foreseen, along with possible refinements of the ZTSP for integration purposes.

6 References

[R6G24-D41] Deliverable D4.1: Security Automation for 6G. Horizon Europe Project, Grant Agreement No. 101139068.

[R6G25-D43] Deliverable D4.3: ROBUST-6G AI/ML Driven Zero-Touch Security Management Platform Consolidated Design. Horizon Europe Project, Grant Agreement No. 101139068.

[Sig25] Sigma rules, 2025, Online access on September 23rd, 2025: https://sigmahq.io/docs/guide/about .

[PMP25a] PMP official GitHub repository, 2025, Online access on September 23rd, 2025: https://github.com/CyberDataLab/ROBUST-6G_PMP/ .

[PMP25b] PMP swagger APIs, 2025, Online access on September 23rd, 2025: https://cyberdatalab.github.io/ROBUST-6G_PMP/ .

[Mid25] GitHub of Machine ID repository, 2025, Online access on September 23rd, 2025: https://github.com/doroved/mid .

[Tel25] Telegraf official website, 2025, Online access on September 23rd, 2025: https://www.influxdata.com/time-series-platform/telegraf/ .

[Flu25] Fluentd official website, 2025, Online access on September 23rd, 2025: https://www.fluentd.org/ .

[Fal25] Falco official website, 2025, Online access on September 23rd, 2025: https://falco.org/ .

[Fil25] Filbeat official website, 2025, Online access on September 23rd, 2025: https://www.elastic.co/beats/filebeat .

[Kaf25] Apache Kafka official website, 2025, Online access on September 23rd, 2025: https://kafka.apache.org/ .

[Jso24] JSON2PCAP official GitHub repository, 2024, Online access on September 23rd, 2025: https://github.com/H21lab/json2pcap .

[Sno25] Snort3 official website, 2025, Online access on September 23rd, 2025: https://www.snort.org/

[Ope26] OpenSearch official page, 2026, Online access on 24th, 2026: https://opensearch.org/

[Mon26] MongoDB official page, 2026, Online access on 24th, 2026: https://www.mongodb.com/

[Red26] Redis official page, 2026, Online access on 24th, 2026: https://redis.io/

[Pro26] Prometheus official page, 2026, Online access on 24th, 2026: https://prometheus.io/

[Inf26] InfluxDB official page, 2026, Online access on 24th, 2026: https://prometheus.io/

[Log26] Logstash official page, 2026, Online access on 24th, 2026: https://www.elastic.co/es/logstash

[Gra26] Grafana official page, 2026, Online access on 24th, 2026: https://grafana.com/

[Uvi26] Uvicorn official page, 2026, Online access on 24th, 2026: https://uvicorn.dev/

[Wsag11] Web Services Agreement Specification (WS-Agreement), 2011, https://ogf.org/documents/GFD.192.pdf

[ssla25a] SSLA Manager openAPI specifications, ssla-manager-rest/openapi.yaml at main · ThalesGroup/ssla-manager-rest GitHub, Online access on September 23rd, 2025: ssla-manager-rest/openapi.yaml at main · ThalesGroup/ssla-manager-rest GitHub

[Nex25a] Nextworks (Italy). (2025). *Zero-Touch-Security-Orchestrator Ontology Manager API*. Zenodo. Online access on September 23rd, 2025: https://doi.org/10.5281/zenodo.16925313

[Nex25b] Nextworks (Italy). (2025). *Zero-Touch-Security-Orchestrator Catalogues Manager API*. Zenodo. Online access on September 23rd, 2025: https://doi.org/10.5281/zenodo.16925327

[Nex25c] Nextworks (Italy). (2024). Closed Loop Governance - Catalogue API. Zenodo. Online access on September 23rd, 2025: https://doi.org/10.5281/zenodo.14193484

[Nex25d] Nextworks (Italy). (2024). Closed Loop Governance LCM API. Zenodo. Online access on September 23rd, 2025: https://doi.org/10.5281/zenodo.14193635

[Nex25e] Nextworks (Italy). (2024). Multi-cluster extreme-edge resource orchestration API. Zenodo. Online access on September 23rd, 2025: https://doi.org/10.5281/zenodo.14193659

[Nex25f] Nextworks (Italy). (2026). *Zero-Touch-Security-Orchestrator Security Context Manager API* Zenodo. Online access on March 15th 2026: https://doi.org/10.5281/zenodo.19002101

[AppCic]Applications: CICFlowMeter / CIC-AB. Available online: https://www.unb.ca/cic/research/applications.html (accessed on 7 December 2024)

[Bou04] Boutell, M. R., Luo, J., Shen, X., & Brown, C. M. (2004). Learning multi-label scene classification. Pattern recognition, 37(9), 1757-1771.

[Man24] Mannella, L., Canavese, D., & Regano, L. (2024, April). Detecting Cryptomining Traffic in IoT Networks. In Proceedings of the 9th International Conference on Smart and Sustainable Technologies–SpliTech 2024. Institute of Electrical and Electronics Engineers (IEEE).

[MouTon] Moustafa, N. ToN_IoT Datasets. Available online: https://research.unsw.edu.au/projects/toniot-datasets (accessed on 2 October 2024).

[Ram18] Ramírez, Sebastián. 2018. *FastAPI*. FastAPI Project. Online access on September 23rd, 2025: https://fastapi.tiangolo.com/

[CACAO23] CACAO Security Playbooks v2.0 - OASIS Open,Online access on September 23rd, 2025: https://www.oasis-open.org/standard/cacao-security-playbooks-v2-0/

[OpenC222] OASIS Open Command and Control (OpenC2) TC | OASIS, Online access on September 23rd, 2025: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=openc2

[CVE20234914] NVD - CVE-2023-4914, Online access on September 23rd , 2025: https://nvd.nist.gov/vuln/detail/CVE-2023-4914

[CVE20001025] eWave ServletExec CVE-2000-1025, Online access on 13rd, 2026: https://www.cvedetails.com/cve/CVE-2000-1025/

[SPR25] Spring Framework. Online access on September 23rd, 2025: https://spring.io/projects/spring-framework

[ZenGAI426] GenAI4SOAR agents configuration for CACAO playbook generation, 2026, https://zenodo.org/records/18962917?preview=1&token=eyJhbGciOiJIUzUxMiJ9.eyJpZCI6IjZiZDI3YmNkLTg5ODUtNDQ4Yy05NzU2LTk2MDA0MDJkNzc0NiIsImRhdGEiOnt9LCJyYW5kb20iOiIxYWRiYjE2ZTE1NmM1N2M0YmMyMDZhYzM1M2FkZTQ0MSJ9.NLbA8s03M_D_5O1G2-s94-tQRx2p9AxWdTQAoGzm7WPXIErhTCR5IKfnAq395q_wmhr3rUSlNdfhF6ikxgkBcw

[Bork] Borkmann, D. (2009). *netsniff-ng: A high performance Linux networking toolkit*. Retrieved from http://www.netsniff-ng.org/

[Bono24] Bono, J., Grana, J., & Xu, A. (2024). Generative AI and security operations center productivity: evidence from live operations. arXiv preprint arXiv:2411.03116.

[Tcpd] The Tcpdump Group. (n.d.). *Link-layer header types*. tcpdump/libpcap. Retrieved from https://www.tcpdump.org/linktypes.html

[Scapy] Biondi, P. (2024). *Scapy: Packet manipulation tool and network protocol toolkit*. Retrieved from https://scapy.net

[Tianqi16] Tianqi Chen, & Carlos Guestrin. (2016). *XGBoost: A scalable tree boosting system*. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16) (pp. 785–794). https://doi.org/10.1145/2939672.2939785

[Leo01] Leo Breiman. (2001). *Random forests*. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

[Maxi18] Maximilian Christ, Nils Braun, Julius Neuffer, Andreas W. Kempa-Liehr, & Oliver K. Ernst. (2018). *Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package)*. **Neurocomputing, 307**, 72–77. https://doi.org/10.1016/j.neucom.2018.03.067

[Enoch18] Enoch Agyepong, Yulia Cherdantseva,Philipp Reinecke & Pete Burnap. (2019). Challenges and performance metrics for security operations center analysts: a systematic review: Journal of Cyber Security Technology: Vol 4 , No 3. https://www.tandfonline.com/doi/full/10.1080/23742917.2019.1698178

7 Annex I

7.1 Temporal Split of Network Attacks in Training, Validation and Testing Datasets

Table 7-1Table 7-3 present the detailed temporal distribution of threat labels in the CICToN-IoT dataset across the training, validation, and testing sets. *Window_start* and *window_end* indicate the time interval during which the data is collected, whereas *attack_start* and *attack_end* denote the timestamps of the first and last detected threats within each window interval.

Table 7-1 Temporal Split of Network Attacks for Training Dataset

Label	window_start	attack_start	attack_end	window_end
Madominer	2018-12-17 09:38:15	2018-12-17 09:38:15	2018-12-19 14:53:04	2018-12-19 14:53:04
Coinhive	2019-02-08 04:53:39	2019-02-08 04:53:39	2019-02-08 06:12:56	2019-02-08 06:12:56
Benign	2019-04-02 17:45:58	2019-04-02 17:45:58	2019-04-03 19:33:41	2019-04-03 19:33:41
Scanning	2019-04-23 20:10:09	2019-04-23 20:10:09	2019-04-24 11:25:45	2019-04-24 11:25:45
DoS	2019-04-25 02:13:41	2019-04-25 02:13:41	2019-04-25 07:34:56	2019-04-25 07:34:56
Injection	2019-04-25 13:49:17	2019-04-25 13:49:19	2019-04-25 16:27:30	2019-04-25 16:27:30
DDoS	2019-04-25 22:48:46	2019-04-25 22:48:47	2019-04-26 08:17:58	2019-04-26 08:17:58
Password	2019-04-26 21:44:49	2019-04-26 21:45:47	2019-04-27 09:53:33	2019-04-27 09:53:33
XSS	2019-04-27 12:54:22	2019-04-27 12:54:22	2019-04-27 18:00:24	2019-04-27 18:00:24
Ransomware	2019-04-28 11:49:50	2019-04-28 11:49:53	2019-04-28 13:00:11	2019-04-28 13:00:11
Backdoor	2019-04-28 14:53:49	2019-04-28 15:29:59	2019-04-29 09:13:12	2019-04-29 09:13:12
MITM	2019-04-29 20:20:41	2019-04-29 20:20:41	2019-04-29 22:12:09	2019-04-29 22:12:09
Xmrstak	2019-11-07 03:49:49	2019-11-07 03:49:49	2019-12-08 05:46:55	2019-12-08 05:46:55

Table 7-2 Temporal Split of Network Attacks for Validation Dataset

Label	window_start	attack_start	attack_end	window_end
Madominer	2018-12-19 14:55:58	2018-12-19 14:55:58	2018-12-19 19:34:05	2018-12-19 19:34:05
Coinhive	2019-02-08 06:12:59	2019-02-08 06:12:59	2019-02-08 06:38:54	2019-02-08 06:38:54
Benign	2019-04-03 19:33:42	2019-04-03 19:33:42	2019-04-04 05:20:45	2019-04-04 05:20:45
Scanning	2019-04-24 11:25:46	2019-04-24 11:25:46	2019-04-24 12:13:28	2019-04-24 12:13:28
DoS	2019-04-25 07:34:57	2019-04-25 07:34:57	2019-04-25 09:12:55	2019-04-25 09:12:55
Injection	2019-04-25 16:27:31	2019-04-25 16:27:31	2019-04-25 17:17:44	2019-04-25 17:17:44
DDoS	2019-04-26 08:17:59	2019-04-26 08:17:59	2019-04-26 10:57:14	2019-04-26 10:57:14
Password	2019-04-27 09:53:34	2019-04-27 09:53:34	2019-04-27 10:49:37	2019-04-27 10:49:37
XSS	2019-04-27 18:00:25	2019-04-27 18:00:25	2019-04-27 19:19:33	2019-04-27 19:19:33
Ransomware	2019-04-28 13:00:12	2019-04-28 13:00:12	2019-04-28 13:10:03	2019-04-28 13:10:03
Backdoor	2019-04-29 09:13:15	2019-04-29 09:13:15	2019-04-29 13:29:41	2019-04-29 13:29:41
MITM	2019-04-29 22:12:10	2019-04-29 22:12:10	2019-04-29 22:13:42	2019-04-29 22:13:42
Xmrstak	2019-12-08 05:47:25	2019-12-08 05:47:25	2019-12-09 07:44:21	2019-12-09 07:44:21

Table 7-3 Temporal Split of Network Attacks for Testing Dataset

Label	window_start	attack_start	attack_end	window_end
Madominer	2018-12-19 19:36:24	2018-12-19 19:36:24	2018-12-20 01:35:00	2018-12-20 01:35:00
Coinhive	2019-02-08 06:38:55	2019-02-08 06:38:55	2019-02-08 07:04:30	2019-02-08 07:04:30
Benign	2019-04-04 05:21:00	2019-04-04 05:21:00	2019-04-04 15:50:10	2019-04-04 15:50:10
Scanning	2019-04-24 12:13:29	2019-04-24 12:13:29	2019-04-24 17:22:44	2019-04-24 17:56:37
DoS	2019-04-25 09:12:56	2019-04-25 09:12:56	2019-04-25 10:50:20	2019-04-25 10:50:20

Injection	2019-04-25 17:17:45	2019-04-25 17:17:45	2019-04-25 18:14:19	2019-04-25 18:14:19
DDoS	2019-04-26 10:57:15	2019-04-26 10:57:15	2019-04-26 15:33:32	2019-04-26 15:33:32
Password	2019-04-27 10:49:38	2019-04-27 10:49:38	2019-04-27 11:47:44	2019-04-27 11:47:44
XSS	2019-04-27 19:19:34	2019-04-27 19:19:34	2019-04-27 20:40:52	2019-04-27 20:40:52
Ransomware	2019-04-28 13:10:05	2019-04-28 13:10:19	2019-04-28 14:05:25	2019-04-28 14:32:39
Backdoor	2019-04-29 13:29:42	2019-04-29 13:29:42	2019-04-29 17:45:29	2019-04-29 17:45:29
MITM	2019-04-29 22:13:43	2019-04-29 22:13:43	2019-04-29 22:43:15	2019-04-29 22:45:55
Xmrstak	2019-12-09 07:45:42	2019-12-09 07:45:42	2019-12-17 12:03:34	2019-12-17 12:03:34