Grant Agreement No.: 101096342
Call: HORIZON-JU-SNS-2022
Topic: HORIZON-JU-SNS-2022-STREAM-B-01-04
Type of action: HORIZON-JU-RIA

Holistic, omnipresent, resilient services
for future 6G wireless and computing ecosystems

D4.2 HORSE AI-assisted human-centric Secure and Trustable Orchestration developed (IT-2)

Revision: v.1.0

Work package	WP 4
Task	Task 4.1, Task 4.2, Task 4.3, Task 4.4
Due date	30/06/2025
Submission date	30/06/2025
Deliverable lead	8BELLS
Version	1.0
Authors	Michail Danousis (8BELLS), Eva Rodriguez (UPC), Alice Peremonti (MARTEL), Alexandros Katsarakis (STS), Alex Carrega (CNIT), Xavi Masip (UPC), Fabrizio Granelli (CNIT), Iulisloi Zacarias (TUBS)
Reviewers	Iulisloi Zacarias (TUBS), Fabrizio Granelli (CNIT)

Abstract	D4.2 HORSE AI-Assisted human-centric Secure and Trustable Orchestration, documents the final development of the AI-assisted, human-centric, secure, and trustable orchestration components of the HORSE platform, as part of Work Package 4 (WP4). It details the implementation, debugging, and integration readiness of all orchestration modules, including Smart Monitoring, Pre-processing, the Common Knowledge Base, the RTR framework, ePEM, DOC, and CAS. These components have reached Implementation Target 2 (IT-2) maturity and are ready for full integration and validation in WP5, contributing to a resilient, secure, and intelligent orchestration framework tailored for 5G/6G environments.
Keywords	AI-assisted orchestration, Human-centric security, 5G/6G network resilience, Smart Monitoring, Pre-processing, Knowledge Base, Intent-based security, Threat mitigation, Secure orchestration, Cybersecurity automation

DOCUMENT REVISION HISTORY

Version	Date	Description of change	List of contributor(s)
V0.1	01/04/2025	1st version of the template for comments	Michail Danousis (8BELLS)
V0.2	03/04/2025	1st version of the Table of Contents	Michail Danousis (8BELLS), Xavi Masip (UPC), Fabrizio Granelli (CNIT)
V0.3	10/04/2024	Updates in the HORSE introduction of the deliverable	Michail Danousis (8BELLS), Eva Rodriguez (UPC), Alice Peremonti (MARTEL), Alexandros Katsarakis (STS), Alex Carrega (CNIT)
V0.4	24/04/2024	Updates in the Components' Sections	Michail Danousis (8BELLS), Eva Rodriguez (UPC), Alice Peremonti (MARTEL), Alexandros Katsarakis (STS), Alex Carrega (CNIT)
V0.5	08/05/2025	Refinements of the Components' Sections	Michail Danousis (8BELLS), Eva Rodriguez (UPC), Alice Peremonti (MARTEL), Alexandros Katsarakis (STS), Alex Carrega (CNIT)
V0.6	22/05/2025	Version for internal peer review	Michail Danousis (8BELLS)
V0.7	10/06/2025	Revision from TUBS	Iulisloi Zacarias (TUBS)
V0.8	11/06/2024	Revision form CNIT	Fabrizio Granelli (CNIT)
V1.0	20/06/2024	Quality assessment and final version to be submitted.	Fabrizio Granelli (CNIT)

Disclaimer

Copyright notice

Project co-funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	to specify R, DEM, DEC, DATA, DMP, ETHICS, SECURITY, OTHER*	
Dissemination Level		
PU	*Public, fully open, e.g. web*	**x**
SEN	*Sensitive, limited under the conditions of the Grant Agreement*	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

* *R: Document, report (excluding the periodic and final reports)*
DEM: Demonstrator, pilot, prototype, plan designs
DEC: Websites, patents filing, press & media actions, videos, etc.
DATA: Data sets, microdata, etc
DMP: Data management plan
ETHICS: Deliverables related to ethics issues.
SECURITY: Deliverables related to security issues
OTHER: Software, technical diagram, algorithms, models, etc.

Executive summary

The HORSE D4.2 deliverable, "AI-assisted Human-centric Secure and Trustable Orchestration developed (IT-2)," serves as the final technical report for Work Package 4 (WP4) of the HORSE project [1]. Building upon D4.1, this document details the refined, debugged, and fully developed versions of the project's AI-assisted human-centric secure and trustable orchestration components, achieving the IT-2 maturity level. The deliverable provides a comprehensive overview of technical enhancements, security measures, and operational guidelines, culminating in a robust, secure, and human-centric orchestration platform ready for final integration in WP5.

This document highlights the final development status of all key components, each playing a distinct role in enabling AI-driven, secure, and human-centric orchestration for next-generation networks:

Smart Monitoring evolved into a central observability hub, with enhanced ingestion, indexing, and fine-grained access control using ElasticSearch.

The **Pre-processing Module** now delivers high-performance, schema-validated data harmonization, supporting both real-time and historical analytics.

A new **Common Knowledge Base (CKB)** was introduced to centralize threat and mitigation intelligence, using Generative AI and LLMs for dynamic updates and prioritization.

The **RTR Framework** advanced its ability to translate structured and natural language security intents into actionable mitigations, with enforcement tracking and performance optimizations.

The **End-to-End Proactive Secure Connectivity Manager (ePEM)** underwent major architectural revisions, adding support for K8S and Proxmox, and improving blueprint lifecycle and NFV management.

Domain Orchestrator Connectors (DOC) now support distributed multicluster operations and improved northbound and southbound interfacing.

The **Compliance Assessment System (CAS)** was enhanced with OPA-based policy enforcement and multi-level compliance reporting.

Collectively, these enhancements position WP4 for successful integration in WP5. All components now exhibit robust performance, security, and interoperability, aligning with HORSE's mission to deliver an intelligent and secure orchestration framework tailored for dynamic 5G/6G network environments.

Table of contents

List of figures

List of tables

Abbreviations

API	Application Programming Interface
CAS	Compliance Assessment Procedures
CKB	Common Knowledge Base
CNIT	Consorzio Nazionale Interuniversitario per le Telecomunicazioni
DFF	Data Fusion Framework
DEME	Decision-Making Engine
DOC	Domain Orchestrator Connectors
DT	Digital Twin
ePEM	End-to-End Proactive Secure Connectivity Manager
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IBI	Intent-Based Interface
IDPS	Intrusion Detection and Prevention System
IMEI	International Mobile Equipment Identity
IT-1	Implementation Target 1
IT-2	Implementation Target 2
JSON	JavaScript Object Notation
K8S	Kubernetes
LLM	Large Language Model
LCM	Lifecycle Management
ML	Machine Learning
NLP	Natural Language Processing
NFV	Network Function Virtualization
OPA	Open Policy Agent
PAG	Privacy-Aware Gateway
PCAP	Packet Capture
PII	Personally Identifiable Information
RBAC	Role-Based Access Control
REST	Representational State Transfer

RTR	**Reliability, Trust, and Resilience Provisioning Framework**
SM	**Smart Monitoring**
SNS	**Smart Networks and Services**
SR	**Source Routing**
STS	**Secure Trustable Services**
TCP	**Transmission Control Protocol**
TLS	**Transport Layer Security**
UE	**User Equipment**
VIM	**Virtualized Infrastructure Manager**
VNF	**Virtualized Network Function**
VyOS	**Virtual Operating System (open-source network OS)**
WP	**Work Package**

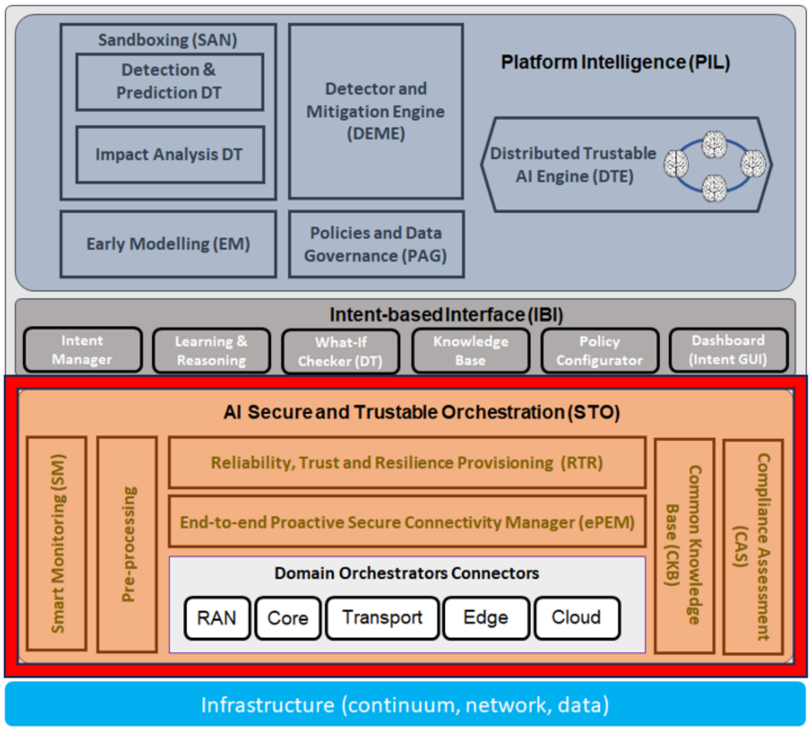
1 About this Document

1.1 Role of this Document

The deliverable, "HORSE AI-assisted Human-centric Secure and Trustable Orchestration (IT-2)", serves as the final technical report for Work Package 4 (WP4) within the HORSE project. As a continuation of D4.1, this deliverable documents the refined, debugged, and fully developed versions of the AI-assisted human-centric secure and trustable orchestration components, achieving in the second iteration (IT-2) of the project. These components, illustrated in Figure 1.1 (highlighted by the red box), include:

- Smart Monitoring Procedures
- Pre-processing Module
- Common Knowledge Base Module
- Reliability, Trust, and Resilience Provisioning Framework
- End-to-End Proactive Secure Connectivity Manager
- Domain Orchestrator Connectors
- Compliance Assessment Procedures

This document details the advancements, debugging efforts, and integration readiness of these components, incorporating feedback from preliminary integration and validation tasks in WP5. The document ensures transparency, accountability, and traceability by providing a comprehensive overview of the technical enhancements, security measures, and operational guidelines, culminating in a robust, secure, and human-centric orchestration platform ready for final integration in WP5.



Sandboxing (SAN)
Detection & Prediction DT
Impact Analysis DT
Detector and Mitigation Engine (DEME)
Platform Intelligence (PIL)
Distributed Trustable AI Engine (DTE)
Early Modelling (EM)
Policies and Data Governance (PAG)
Intent-based Interface (IBI)
Intent Manager
Learning & Reasoning
What-If Checker (DT)
Knowledge Base
Policy Configurator
Dashboard (Intent GUI)
AI Secure and Trustable Orchestration (STO)
Smart Monitoring (SM)
Pre-processing
Reliability, Trust and Resilience Provisioning (RTR)
End-to-end Proactive Secure Connectivity Manager (ePEM)
Domain Orchestrators Connectors
RAN
Core
Transport
Edge
Cloud
Common Knowledge Base (CKB)
Compliance Assessment (CAS)
Infrastructure (continuum, network, data)

Figure 1.1 HORSE Components Interfaces

1.2 Relationship to other HORSE deliverables

D4.2 builds on D4.1 and maintains strong interconnections with other HORSE project deliverables, including:

- **Deliverable D2.4 – HORSE Architectural Design (IT-2):** D2.4 provides the final architectural framework, guiding the development and integration of D4.2 components. The introduction of the Common Knowledge Base Module in D4.2 addresses a gap identified in the updated architecture of D2.4.
- **Deliverable D3.2 – HORSE Platform Intelligence developed (IT-2):** D3.2 enhances the platform's intelligence, complementing the secure orchestration components of D4.2 to enable advanced AI-driven functionalities.
- **Deliverable D4.1 – HORSE AI-assisted Human-centric Secure and Trustable Orchestration developed (IT-1):** D4.2 refines and completes the components introduced in D4.1, incorporating debugging and enhancements based on WP5 feedback.
- **Deliverable D5.2 – HORSE Platform Integration and Validation (IT-2):** D4.2 provides the fully developed components required for final integration and validation in D5.2, ensuring operational readiness.

These relationships highlight the iterative and interconnected nature of the HORSE project, with D4.2 serving as a critical milestone in delivering mature components for the platform.

1.3 Structure of the Document

The document is structured to provide a detailed account of the final development, debugging, and integration of the AI-assisted human-centric secure and trustable orchestration components. Each chapter of the document describes a component of the AI Secure and Trustable Orchestration component and are organized as follows:

Chapter 2: Smart Monitoring Procedures

This chapter details the final development of the Smart Monitoring Procedures, including the shift from the initial Everest plan to a more dynamic approach. It covers the purpose, integration details (e.g., APIs for Elastic and UE location sharing), access control, and changes from IT-1 to IT-2.

Chapter 3: Pre-processing Module

This section examines the finalized Pre-processing Module, detailing its development, security protocols, integration interfaces, access control, and debugging efforts from IT-1 to IT-2.

Chapter 4: Common Knowledge Base Module

This chapter introduces the Common Knowledge Base Module, a new component addressing an architectural gap identified in D2.4. It covers the module's GenAI-powered design, development using a Python framework, security measures (e.g., API server, Pydantic models, .env files), RESTful API integration, and debugging efforts.

Chapter 5: Reliability, Trust, and Resilience Provisioning Framework

This chapter details the finalized framework, including its development, security measures, integration interfaces, access control, deployment guidelines, and changes from IT-1 to IT-2.

Chapter 6: End-to-End Proactive Secure Connectivity Manager

This section focuses on the finalized Connectivity Manager, covering its development, security protocols, integration interfaces, access control, deployment guidelines, and debugging from IT-1 to IT-2.

Chapter 7: Domain Orchestrator Connectors

This chapter elaborates on the finalized Domain Orchestrator Connectors, detailing their functionalities, northbound interfaces, resource management, distributed multicluster support, and changes from IT-1 to IT-2.

Chapter 8: Compliance Assessment Procedures

This section covers the fully developed Compliance Assessment Procedures, including their purpose, pipeline, decision-making process, integration with monitoring mechanisms (e.g., IBI, DEME), blackbox nature, and debugging from IT-1 to IT-2.

Chapter 9: Discussion

The final chapter synthesizes WP4's achievements, highlighting the integration with other components of the HORSE project, and their contributions to the HORSE platform's broader objectives.

2 Smart Monitoring Procedures

Smart Monitoring (SM) is a pivotal component within the HORSE platform, giving us comprehensive oversight of the whole ecosystem. Its main job involves continuously gathering, processing, and securely storing critical operational and security-relevant information. This comprehensive approach to data collection and analysis is essential for achieving high-yield network observability, particularly in advanced network environments such as those envisioned for 6G, where techniques like SRv6-INT enabled network monitoring and measurement are becoming increasingly relevant [2].

SM acts like a hub, collecting data from diverse sources and then exposing this processed information to other HORSE components. These components, are designed to interact with various data spaces (we call them Indexes inside ElasticSearch) to make sure data delivery is robust and structured. The insights we get from this collected information are crucial for HORSE, offering vital understanding of both the deployed ecosystem's structure and, even more importantly, its current operational state. Within HORSE architecture SM acts mainly as a data provider and holder for several components.

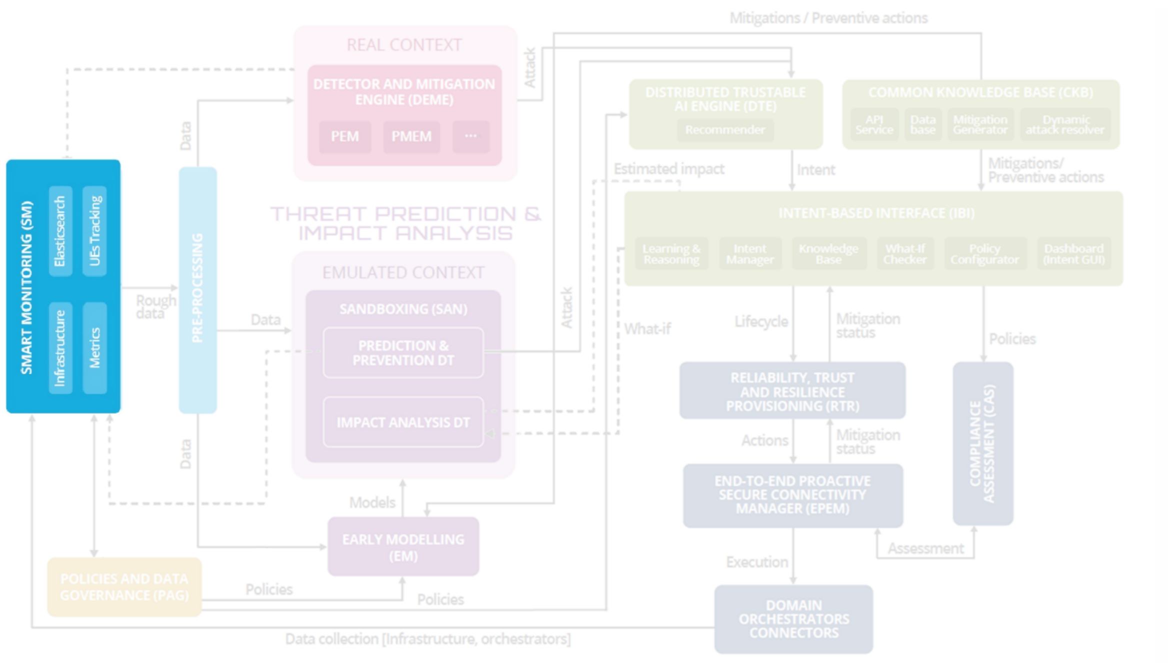


Figure 2.1 Smart Monitoring component highlighted in the HORSE architecture

2.1 Overview and Final Development Details

The HORSE platform is fundamentally subjected upon the efficient and secure management of its critical data, which includes network traffic, system logs, and internal feedback. This information is often disparate and widely distributed, a challenge that SM is specifically designed to address [3]. SM establishes a unified collection point for HORSE, providing comprehensive monitoring data within the demanding context 6G environment. Given the immense volume of data in these advanced network spaces and the imperative to prevent data loss (especially from a security perspective) SM's role in collecting data from various system points and offering a secure interface for critical data storage is important.

The architectural changes of Smart Monitoring reflects a strategic adaptation to the dynamic requirements of the HORSE platform. Initially, the approach considered leveraging Sphynx's EVEREST. However, a thorough analysis of system needs led to the decision that SM's role should be more dynamic. By focusing on the ingestion of raw data and integrating the Pre-processing module as a "bridge" to the main architecture. This design ensures that SM can operate without requiring real-time knowledge of the system's active components. This strategic decoupling allows SM to specialize in high-volume, high-velocity data ingestion, fundamental formatting (for ElasticSearch), and of course storage. The Pre-processing module, acting as the "bridge" then handles the dynamic transformation and standardization of this data based on the current requirements of downstream modules, such as DEME and DT. This architectural refinement enhances the overall scalability and resilience of the HORSE system.

The data types collected and managed by SM are selected for their strategic value within the HORSE ecosystem:

- **PCAP data:** Packet capture (pcap) data is critical for any system that monitors and makes decisions associated with network traffic. SM is responsible for fetching, processing and of course storing pcap data in a readable format that can be easily consumed by most tools.
- **PCAP data anonymized:** To ensure a future-proof solution and address data privacy, SM integrates with the Privacy-Aware Gateway (PAG) within HORSE. This collaboration maintains an anonymized version of the data, with the objective of enabling its broader utilization.
- **IMEI data:** SM exposes and serves information about the ever-changing ecosystem, particularly concerning International Mobile Equipment Identity (IMEI) data. This allows HORSE components to take advantage of this information for more sophisticated decision-making based on the connected devices.
- **Analytics data:** Analytics play a vital role in HORSE. SM is utilized to store analytics data generated and used to train the artificial intelligence components within HORSE (specifically DEME and DT). This data is crucial for re-training purposes or for diagnosing issues that may have occurred during the training process.

The operational framework of Smart Monitoring is fundamentally built around ElasticSearch and its robust integration capabilities. ElasticSearch serves as the core data store and indexing engine, facilitating the efficient collection, processing, and storage of all aforementioned data types, ensuring their rapid accessibility for querying and analysis. SM incorporates various methods for capturing and processing data to ensure fast indexing, which is a major factor in the performance of live systems.

2.2 Security and Data Collection

Security is a fundamental design principle embedded within the SM component, handling every stage of the data lifecycle from collection to storage and sharing within HORSE. A primary objective is to minimize data exposure, ensuring that sensitive information is protected at all times.

The initial layer of security focuses on secure data ingress. Each endpoint feeding data into Smart Monitoring is either co-located on the same Virtual Machine or connected via HTTPS, ensuring secure data transfer between components [4]. Furthermore, security extends to the data itself through SM's crucial connection with the PAG, which provides anonymized data for use. For each data type, the objective is to ensure it is securely processed, stored, and shared within HORSE.

A detailed analysis of how each individual data category is monitored, collected, processed, and stored within SM is provided below:

- **pcap_data:** Pcap data collection begins with capturing PCAP files using tools like TCPDump, which log events occurring on HORSE. The established procedure dictates that components and devices produces PCAP file with a one-minute lifespan for processing and saving, before the next file takes its place. PCAP files provide vital information for identifying potential attacks. These data are then transformed by SM into a digestible format, which is subsequently saved into ElasticSearch, making the data indexable and searchable by authorized users. Providing an indexable form is crucial as it enables the execution of complex queries, delivering vital information to the rest of the system.
- **pcap_data_anonymization:** The importance of data anonymization has been previously mentioned. It is essential to note that anonymization requires the data to have been transformed into the digestible Elastic format (JSON in this case) to facilitate a faster and easier anonymization process.
- **imei_data:** Each HORSE release will involve different devices connected to the network, and HORSE must be able to identify these devices for critical decision-making. SM tracks these devices and provides a mechanism for HORSE components to access this data. This information is provided by the host of the HORSE system, as Smart Monitoring is agnostic to the specific devices connected and relies on this external provision.
- **analytics_data:** This category comprises accumulated data generated by the Pre-processing module based on the PCAP data provided. Generally, these data are related to the training of the machine learning components of HORSE (DEME and DT).

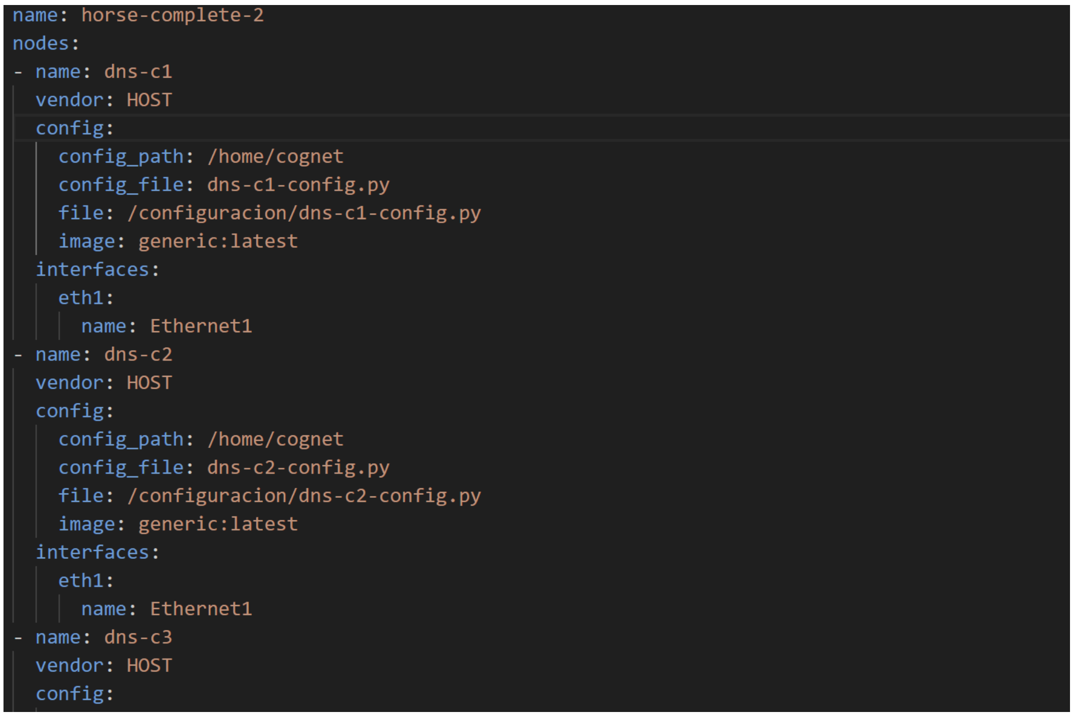
2.3 Integration and Interfaces

Smart Monitoring's design positions it primarily as an "observer" rather than an "actor" within the HORSE ecosystem, which simplifies its integration into the broader platform. While SM plays a vital role in HORSE, it is designed to adapt to the system's existing architecture rather than requiring the system to adapt to it. As other components and parts of HORSE extract information from Smart Monitoring, it offers dedicated interfaces for direct communication.

2.3.1 APIs and Format Exposed Through Interfaces

Smart Monitoring provides a suite of APIs tailored to the specific data consumption needs of the HORSE platform. These interfaces offer three distinct mechanisms for interaction:

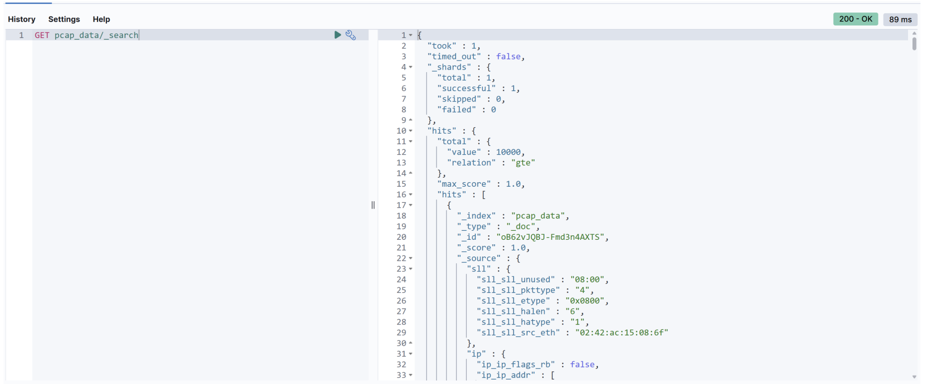
- **RESTful API:** SM exposes a dedicated RESTful API for specific data queries. An example includes functionality to "fetch connected devices location within the network" where Smart Monitoring exposes the location of UEs within the network. Access to this sensitive information is strictly controlled and limited only to whitelisted IP addresses.



```
name: horse-complete-2
nodes:
- name: dns-c1
  vendor: HOST
  config:
    config_path: /home/cognet
    config_file: dns-c1-config.py
    file: /configuracion/dns-c1-config.py
    image: generic:latest
  interfaces:
    eth1:
      name: Ethernet1
- name: dns-c2
  vendor: HOST
  config:
    config_path: /home/cognet
    config_file: dns-c2-config.py
    file: /configuracion/dns-c2-config.py
    image: generic:latest
  interfaces:
    eth1:
      name: Ethernet1
- name: dns-c3
  vendor: HOST
  config:
```

Figure 2.2 Example API result for UEs location

- **Kibana for data visualization and management:** Recognizing the critical importance of rapid data visualization and statistical analysis for informed decision-making in a dynamic system, SM incorporates Kibana. This integration grants users with appropriate permissions the ability to observe system operations at a higher level, facilitating quick insights into operational metrics, security events, and data trends.
  - Kibana provides an intuitive way to query and view specific data saved in the ElasticSearch environment



```
GET pcap_data/_search
```

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 10000,
      "relation" : "gte"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "pcap_data",
        "_type" : "_doc",
        "_id" : "oB62vJQBJ-Fmd3n4AXTS",
        "_score" : 1.0,
        "_source" : {
          "sll" : {
            "sll_sll_unused" : "08:00",
            "sll_sll_pkttype" : "4",
            "sll_sll_etype" : "0x0800",
            "sll_sll_halen" : "6",
            "sll_sll_hatype" : "1",
            "sll_sll_src_eth" : "02:42:ac:15:08:6f"
          },
          "ip" : {
            "ip_ip_flags_rb" : false,
            "ip_ip_addr" : [
```

Figure 2.3 Example Kibana dev querying

- As well as intuitive dashboards for the overall overview of the SM.

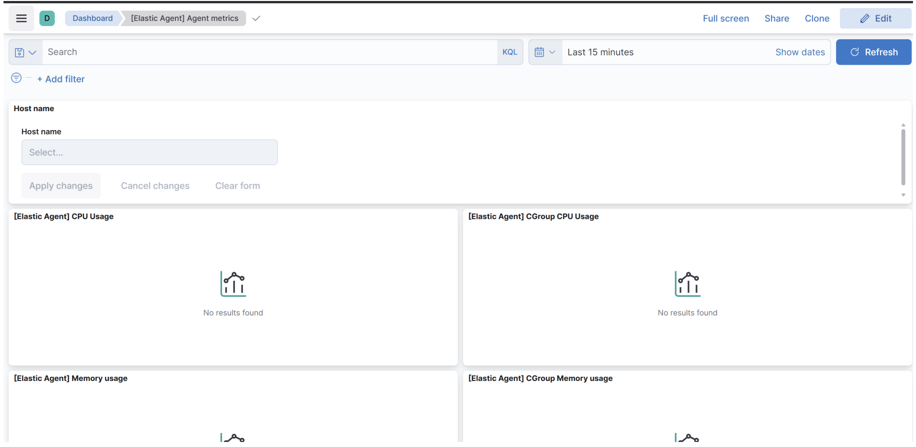


Figure 2.4 Example Kibana dashboards for visualisation

- **ElasticSearch API for direct read/write access:** SM internally reads and writes data to ElasticSearch for fast querying and indexing. Additionally, SM provides a direct interface to these ElasticSearch indexes (individual data partitions). This allows specific users or components, granted precise permissions based on their role within HORSE, to directly access these data segments. By providing direct access to ElasticSearch, components are empowered to perform new queries without the need for an intermediary.Access Control and Permissions [5].

Given that Smart Monitoring serves as the repository for critical and potentially sensitive operational and security data, the implementation of a strict access control protocol is important. This protocol ensures data integrity, confidentiality and availability, ensuring that only authorized components and users can access specific data sets.

The access control mechanisms are multi-layered:

- **API Whitelisting:** For the API, access requires whitelisting to retrieve UE locations. This enables robust data handling and ensures that only whitelisted components can access sensitive information, significantly enhancing data handling security.
- **Principle of Least Privilege for Data Access:** Adhering to the principle of least privilege, data collected and stored within Smart Monitoring is not universally accessible. Instead, each user is granted access only to the data necessary for their specific process.
- **Role-Based Access Control (RBAC) for Indexes:** Direct access to ElasticSearch indexes is governed by a sophisticated RBAC system. This ensures that permissions are assigned based on the defined roles of partners and components within the HORSE consortium, allowing for precise management of read and write privileges across different data categories [6].

Table 2.1 provides a comprehensive overview of the current data access control matrix, detailing the permissions granted to each partner for various data indexes within the Smart Monitoring component.

Table 2.1 SM's data access control matrix

Component/Data	pcap_data	anonym_index	imei_index	analytics_index

SM	Read, Write	-	-	-
PAG	Read	Read, Write	Read	-
Pre-processing	Read	-	-	Read, Write
DEME	-	-	-	Read
TESTBED	Read, Write	-	Read, Write	-
TESTBED	Read, Write	-	Read, Write	-
TESTBED	Read, Write	-	Read, Write	-

2.4 Changes and Debugging from IT-1 to IT-2

The way Smart Monitoring developed from IT-1 to IT-2 shows a dynamic and adaptive evolution, driven by constantly looking at system needs and what we learned from debugging. Since HORSE started, Smart Monitoring's purpose and how it approaches the solution has changed quite a bit. This change, it has really matured SM, making it a much more central and pivotal component within the HORSE architecture.

Key evolutionary changes and specific additions from IT-1 to IT-2 include:

- **Centralized Architectural Role:** Smart Monitoring became a more centralized point of the architecture, underscoring its increased importance as a data hub for the ecosystem and consolidating monitoring capabilities.
- **Enhanced Data Accessibility:**
- **UE Location Information:** A specific and valuable addition was the implementation of a mechanism for components to retrieve the location of User UEs within the network. This capability, exposed via a dedicated RESTful API, provides critical context for security analysis and resource management.
- **Expansion of Indexed Data:** The scope of data managed by Smart Monitoring significantly expanded with the introduction of additional indexes for collected, processed, and served data. These new indexes accommodate a broader range of data types, reflecting the evolving information requirements of HORSE's various functionalities.
- **Granular Access Control Implementation:** A critical security enhancement was the robust implementation of access control for each individual user. This ensures that data access is strictly managed based on roles and necessity, significantly bolstering the security posture of the sensitive data residing within SM.

The evolution of SM from IT-1 to IT-2, particularly its centralization, directly addresses the challenges of handling massive, real-time, and diverse data streams characteristic of 6G environment. The initial architectural choices in IT-1 presented bottlenecks in managing the volume, velocity, and variety of data. The shift to a centralized aggregation point simplifies data ingestion, while the Pre-processing bridge offloads complex transformations, improving overall system throughput and scalability. The expansion of indexed data accommodates the increasing diversity of information, and the robust access control ensures secure governance of this growing dataset. These developments, informed by iterative debugging and integration insights, position SM as a resilient and high-performance data backbone, fully prepared to meet the rigorous demands of the HORSE platform.

3 Pre-Processing Module

3.1 Overview and Final Development Details

The Pre-processing Module in the HORSE architecture (Figure 3.1) serves as a critical intermediary, harmonizing and standardizing raw data from the Smart Monitoring (SM) component for subsequent analytical processes. This module is responsible for extracting data from the Elasticsearch database of the SM component and then pre-processing it to clean and restructure the information according to the needs of the consuming software components. This rigorous approach to data preparation is essential for efficient and reliable knowledge discovery from complex datasets, aligning with the principles discussed in general reviews on data preprocessing techniques [7]. After the pre-processing procedure is complete, the data is stored back into the Elasticsearch DB, within the "analytics_index" (displayed in Table 2.1) for easy future access, and directly sent to the consumer, which is the next component in the HORSE workflow. In the IT-2 iteration, the module underwent significant enhancements to improve performance, reliability, and integration capabilities. Key developments include:

- **Error Correction Functionality**: Robust error-handling mechanisms were implemented to address inconsistencies in data payloads. The module now validates incoming data against predefined JSON schemas, flagging and correcting malformed or incomplete datasets to ensure data integrity before processing.
- **Code Optimization**: The codebase was optimized for performance, reducing processing latency and improving throughput. This involved streamlining data parsing algorithms and optimizing interactions with the Data Fusion Framework (DFF) platform, from which the Pre-processing Module is derived. The Pre-processing Module is a performance-optimized subset of DFF, tailored specifically for the HORSE platform's real-time and security requirements, resulting in faster data harmonization and scalability for high-volume data streams.
- **Dual Data Output**: The module now supports immediate data sharing with the Detector and Mitigation Engine (DEME) component for real-time processing, while simultaneously storing processed results in a specialized index within the SM component's Elasticsearch instance. This dual-output approach enhances data availability for both real-time and historical analysis.
- **Containerized Execution**: The module is now deployed in a containerized environment (e.g., Docker), ensuring isolation and controlled communication with external components. This enhances security and simplifies deployment across diverse infrastructures.

These advancements ensure that the Pre-processing Module delivers reliable, efficient, and scalable data preprocessing pipelines, fully aligned with the HORSE platform's objectives.

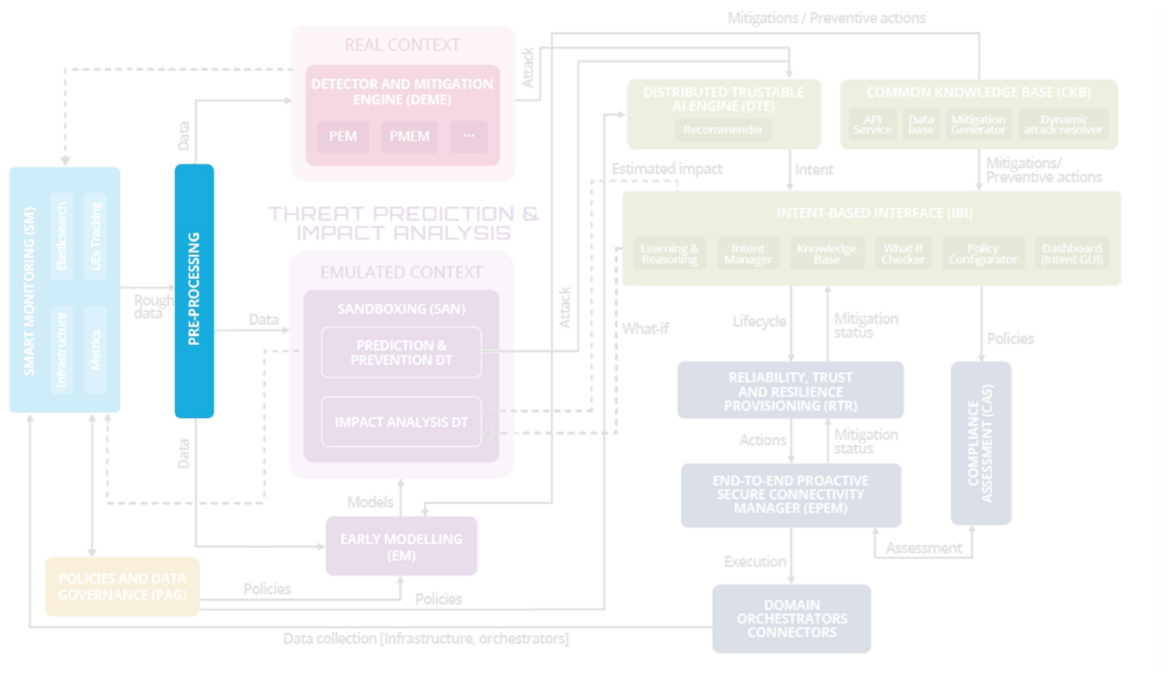


Figure 3.1 The Pre-Processing component within the HORSE architecture

3.2 Security and Data Collection

Security remains a cornerstone of the Pre-processing Module, with IT-2 introducing enhanced measures to protect data during collection, transmission, and processing. The module continues to integrate with the SM component via Elastic Beats, leveraging HTTPS for secure data transmission to maintain confidentiality and integrity. Additional security enhancements include:

- **Containerized Environment**: The module operates within a containerized environment, communicating only through monitored ports and endpoints. This minimizes the attack surface by restricting interactions to predefined, secure channels.
- **Data Validation and Sanitization**: Enhanced schema verification ensures that only valid JSON data is processed. Input sanitization prevents injection attacks, and any anomalous data triggers alerts for further inspection.
- **DFF Platform Security**: The Pre-processing Module inherits core security mechanisms from the DFF platform, which has undergone multiple rounds of security auditing. While it does not expose a graphical user interface (GUI) or allow dynamic pipeline creation by end users (as in the full DFF), it leverages the same secure data handling libraries and OAuth2.0-based authentication model. In the HORSE context, pipeline configurations are managed centrally and deployed securely as part of the containerized module. This provides the privilege of easily creating and managing data pipelines for the HORSE platform by authorized users.
- **Compliance with Regulations**: The module adheres to GDPR and other data protection standards, applying anonymization and data minimization principles to eliminate risks of privacy breaches.

These measures ensure that the Pre-processing Module maintains robust cybersecurity, safeguarding data throughout the preprocessing pipeline.

3.3 Integration and Interfaces

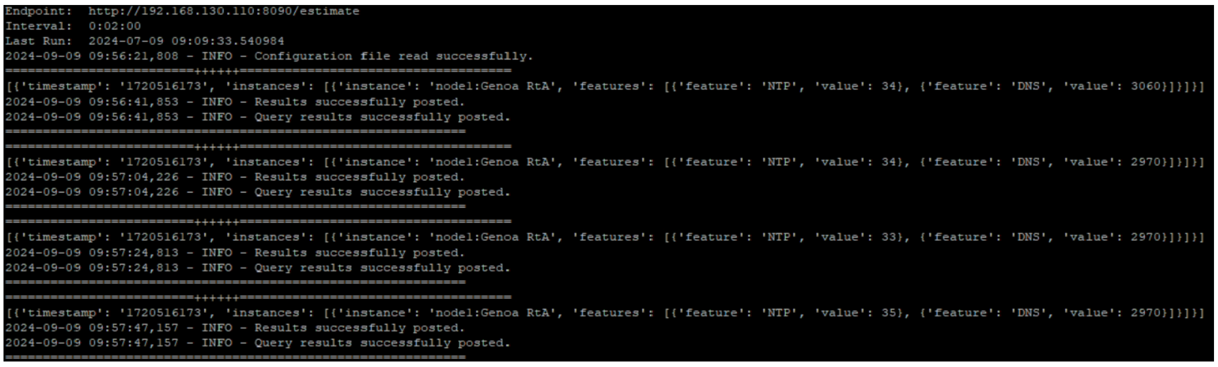
The Pre-processing Module is a performance-optimized subset of the DFF platform. While it does not expose a GUI within the containerized deployment used in HORSE, authorized users can still use the secure GUI of the full DFF platform to configure data pipelines. These configurations are exported and deployed to the Pre-processing Module, ensuring alignment between user-defined flows and the module's hardened backend execution engine.

The Data Fusion Framework (DFF) is a web-based platform that enables authenticated users to design and configure custom data pipelines via a GUI. These pipelines define the data flow from various source databases (e.g., relational, NoSQL, Elasticsearch) to target endpoints (e.g., APIs). Once configured, pipelines are executed by the DFF backend, which handles runtime tasks such as data querying, transformation, and delivery to the defined endpoints. This dual architecture—interactive GUI configuration combined with robust backend execution—ensures both flexibility and scalability in production environments.

The Pre-processing Module is a streamlined version of the DFF platform, optimized to meet the performance, security, and responsiveness requirements of the HORSE platform. Although the secure DFF GUI remains available for authorized users to define and configure pipelines, the backend engine of the Pre-processing Module has been trimmed down and fine-tuned for minimal latency, efficient resource use, and seamless integration in containerized environments. It retains DFF's core capabilities—such as data querying, transformation, and forwarding—while eliminating unnecessary overhead to better serve HORSE's real-time and security-focused architecture.

IT-2 enhancements improve integration reliability and responsiveness:

- **Data Ingestion**: The module uses HTTP requests to ingest JSON-formatted data from the SM component. Enhanced error handling ensures that mismatched payloads are flagged and corrected, with resilience strategies such as retry mechanisms for unresponsive endpoints.
- **Data Retrieval and Storage API**: The module retrieves data from **the SM component** using HTTP GET requests and stores processed results in a specialized Elasticsearch index within the SM component. This index supports historical queries and analytics.
- **Immediate Data Sharing with DEME**: Processed data is transmitted directly to the DEME component via a dedicated API, enabling real-time decision-making. The API uses JSON for interoperability and supports high-frequency data streams.
- **Resilience Strategies**: To address endpoint unresponsiveness, the module implements retry logic (e.g., exponential backoff) and message reconfiguration (e.g., adjusting payload formats) to ensure reliable communication. These strategies were developed based on debugging insights from IT-1.
- **Standardized Data Formats**: Input data from the SM component adheres to a standardized format via Elastic Beats, while output data (to DEME and SM index) uses JSON for consistency and ease of parsing.



```
Endpoint:  http://192.168.130.110:8090/estimate
Interval:  0:02:00
Last Run:  2024-07-09 09:09:33.540984
2024-09-09 09:56:21,808 - INFO - Configuration file read successfully.
=========================++++++====================================
[{'timestamp': '1720516173', 'instances': [{'instance': 'nodel:Genoa RtA', 'features': [{'feature': 'NTP', 'value': 34}, {'feature': 'DNS', 'value': 3060}]}]}]
2024-09-09 09:56:41,853 - INFO - Results successfully posted.
2024-09-09 09:56:41,853 - INFO - Query results successfully posted.
=============================================================
=========================++++++====================================
[{'timestamp': '1720516173', 'instances': [{'instance': 'nodel:Genoa RtA', 'features': [{'feature': 'NTP', 'value': 34}, {'feature': 'DNS', 'value': 2970}]}]}]
2024-09-09 09:57:04,226 - INFO - Results successfully posted.
2024-09-09 09:57:04,226 - INFO - Query results successfully posted.
=============================================================
=========================++++++====================================
[{'timestamp': '1720516173', 'instances': [{'instance': 'nodel:Genoa RtA', 'features': [{'feature': 'NTP', 'value': 33}, {'feature': 'DNS', 'value': 2970}]}]}]
2024-09-09 09:57:24,813 - INFO - Results successfully posted.
2024-09-09 09:57:24,813 - INFO - Query results successfully posted.
=============================================================
=========================++++++====================================
[{'timestamp': '1720516173', 'instances': [{'instance': 'nodel:Genoa RtA', 'features': [{'feature': 'NTP', 'value': 35}, {'feature': 'DNS', 'value': 2970}]}]}]
2024-09-09 09:57:47,157 - INFO - Results successfully posted.
2024-09-09 09:57:47,157 - INFO - Query results successfully posted.
=============================================================
```

Figure 3.2 Screenshot of Execution of the Pre-processing Module

3.4 Access Control and Permissions

Access to the Pre-processing Module is tightly controlled to ensure data security and integrity, with IT-2 refinements enhancing granularity and robustness:

- **DFF API Security**: API endpoints require token-based authentication, with each request validated to prevent unauthorized access. RBAC assigns permissions based on user roles, restricting sensitive operations (e.g., data uploads) to authorized entities.
- **GUI Access Controls**: Access to the DFF GUI is restricted to authenticated users through the HORSE IAM system, which ensures that only authorized individuals can define or modify data pipelines. The GUI allows users to visually construct data processing flows by selecting data sources, configuring preprocessing logic (e.g., filtering, transformation), and specifying output targets. Once submitted, these configurations are stored and picked up by the DFF backend engine for execution. Figure 3.3 illustrates the DFF pipeline configuration interface, showcasing source selection, transformation steps, and destination settings.
- **Containerized Communication**: The containerized environment restricts communication to specific, monitored ports, ensuring that only authorized components (e.g., SM, DEME, DFF) can interact with the module.
- **Data Access for Storage**: Write access to the SM component's Elasticsearch index is restricted to the Pre-processing Module, with read access granted to authorized analytics components via RBAC policies.

These controls ensure that only authenticated and authorized entities interact with the module, maintaining data security and operational integrity.

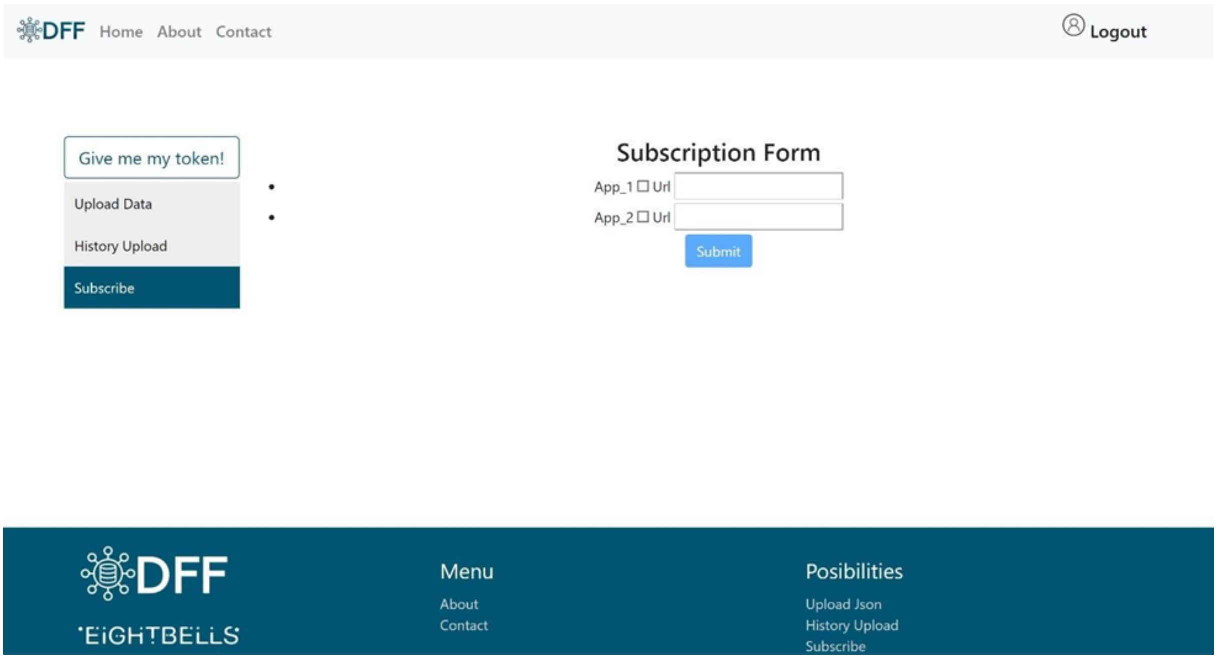


Figure 3.3 Screenshot of DFF UI, displaying page where the user can manage his subscriptions (adding, deleting, and editing their parameters).

3.5 Changes and Debugging from IT-1 to IT-2

The transition from IT-1 to IT-2 involved extensive debugging and targeted enhancements based on issues identified during the initial integration phase conducted in WP5.

- **Payload Mismatch Handling**: Debugging activities revealed frequent payload mismatches originating from the SM component. In IT-2, enhanced schema validation and automatic error correction mechanisms were introduced to resolve format inconsistencies, reducing processing errors.
- **Pipeline Responsiveness**: In IT-1, certain execution flows experienced delays due to the rigid handling of pipeline inputs and data inconsistencies. IT-2 improved internal flow control and implemented retry mechanisms and payload adjustment strategies within the Pre-processing Module, ensuring smoother execution even under fluctuating input conditions.
- **Performance Optimization**: Performance bottlenecks observed in IT-1 were mitigated through code refactoring, optimizing both data parsing routines and inter-component communication. These improvements reduced processing latency by approximately 20%.
- **Security Enhancements**: Migrating to a containerized environment resolved IT-1 security vulnerabilities, particularly those related to open ports. The introduction of monitored and isolated execution environments further minimized the risk of unauthorized access.
- **Dual Output Implementation**: Feedback from IT-1 highlighted the need to support both real-time and historical data access. IT-2 introduced simultaneous data forwarding to DEME and persistent storage in the SM index, thereby enhancing overall system flexibility.

These modifications, validated through WP5 integration tests, ensure that the Pre-processing Module is robust, efficient, and fully prepared for final deployment in the HORSE platform.

4 Common Knowledge Base Module

Initially, the Common Knowledge Base (CKB) was not part of the HORSE architecture. However, during the development process, we identified a gap in the architecture, as documented in Deliverable D2.4, *Landscape and Architectural Design*. To address this gap, we decided to implement and integrate a dedicated component to centrally store and provide essential information on attacks and mitigations.

The rationale behind this addition lies in the need for multiple components, such as the IBI and EM modules, to access consistent and accurate data on attacks and countermeasures. To eliminate data redundancy and ensure a single source of truth, the CKB was established as a centralized repository. Thanks to its API Service, the CKB can now be securely accessed by any HORSE component via REST APIs.

4.1 Overview, Rationale and Development Details

The **CKB** is a pivotal component within the HORSE architecture, leveraging Generative AI (GenAI) to proactively address emerging threats. Its modular design, built on an extensible Python framework, ensures independent and scalable functionality, enabling efficient maintenance, testing, and flexible deployment of its subcomponents. The CKB seamlessly integrates with other HORSE components via **REST APIs**, supporting comprehensive threat intelligence queries. By harnessing **GenAI** capabilities, it synthesizes and correlates data from diverse external sources, autonomously updating knowledge base entries and generating tailored mitigation strategies based on specific attack patterns. Furthermore, the CKB incorporates advanced prioritization features for countermeasures. It utilizes **Large Language Models (LLMs)** to evaluate risks, analyze attack patterns, and rank mitigation actions according to severity and potential impact. This approach ensures that HORSE remains responsive and adaptive to evolving cybersecurity challenges.

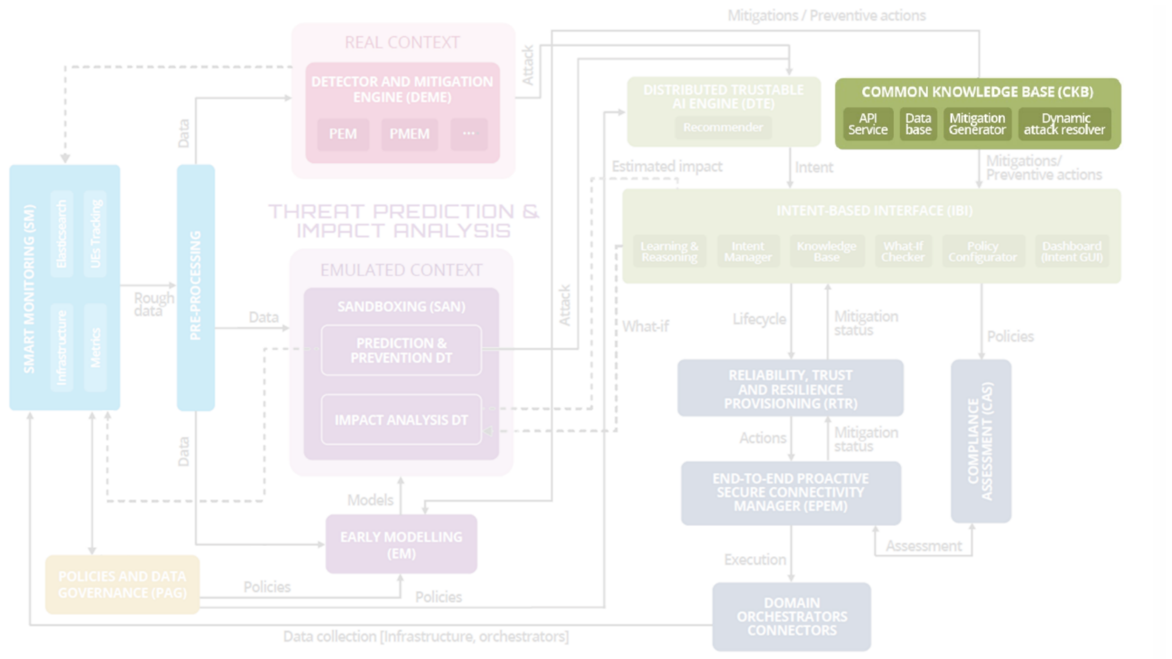

Mitigations / Preventive actions
REAL CONTEXT
DETECTOR AND MITIGATION ENGINE (DEME)
PEM
PMEM
...
Attack
DISTRIBUTED TRUSTABLE AI ENGINE (DTE)
Recommender
COMMON KNOWLEDGE BASE (CKB)
API Service
Data base
Mitigation Generator
Dynamic attack resolver
Estimated impact
Intent
Mitigations/ Preventive actions
INTENT-BASED INTERFACE (IBI)
Learning & Reasoning
Intent Manager
Knowledge Base
What-if Checker
Policy Configurator
Dashboard (Intent GUI)
THREAT PREDICTION & IMPACT ANALYSIS
EMULATED CONTEXT
SANDBOXING (SAN)
PREDICTION & PREVENTION DT
IMPACT ANALYSIS DT
SMART MONITORING (SM)
Elasticsearch
UEs Tracking
Infrastructure
Metrics
PRE-PROCESSING
Rough data
Data
What-if
Lifecycle
Mitigation status
Policies
RELIABILITY, TRUST AND RESILIENCE PROVISIONING (RTR)
Actions
END-TO-END PROACTIVE SECURE CONNECTIVITY MANAGER (EPEM)
COMPLIANCE ASSESSMENT (CAS)
Assessment
Execution
Models
EARLY MODELLING (EM)
POLICIES AND DATA GOVERNANCE (PAG)
DOMAIN ORCHESTRATORS CONNECTORS
Data collection (Infrastructure, orchestrators)

Figure 4.1 The Common Knowledge Base component within the HORSE architecture

The CKB's architecture consists of multiple components, each designed to optimize data management and integration within the HORSE framework.

- **The centralized database** serves as a structured storage system, ensuring secure and high-performance management of data related to threats and mitigation techniques. It is specifically optimized for rapid querying and scalability.
- **The Dynamic Attack Resolver** is composed of several Python modules designed to aggregate data from external sources to identify newly documented attack patterns and mitigation actions. It includes a data preprocessing pipeline to prepare the collected information as input for LLMs, ensuring that the data is consistently formatted and accurate.
- **The Mitigation Generator** is a Python module designed to leverage generative AI to enhance the processed data, resulting in a comprehensive list of attacks and mitigations, as well as a recommended order of execution for these mitigation strategies and a text description.
- **The API Service** provides a REST API-based interface that facilitates seamless interoperability with other HORSE components, allowing them to access the data stored in the CKB without compromising security.

The CKB is implemented using a modular and extensible Python framework aimed at improving maintainability, scalability, and flexibility in cybersecurity data processing within the HORSE architecture. Each core function is encapsulated within independent Python modules to support efficient development, testing, and integration.

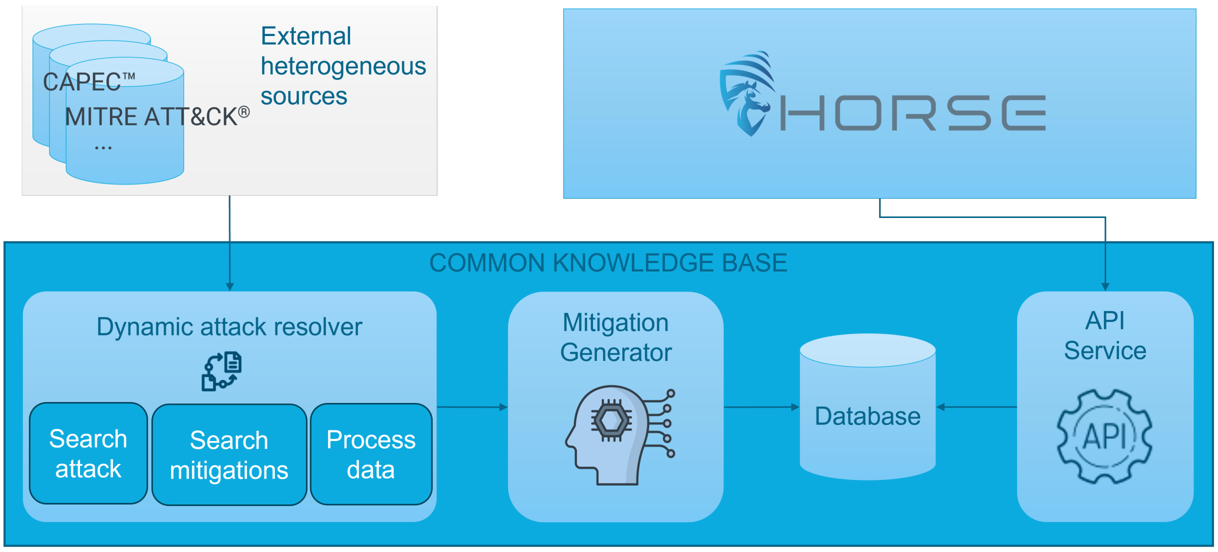


Figure 4.2 High level architecture of CKB component.

The high-level architecture of the CKB component, depicted in Figure 4.2, illustrates how its modular design supports robust data processing and integration. For instance, the Dynamic Attack Resolver plays a central role in gathering cybersecurity knowledge by connecting to trusted external sources. It identifies and extracts attack patterns and corresponding mitigation strategies, organizing them through a structured pipeline. This process involves dedicated modules for searching and structuring the data, which is then exported in a standardized format (CSV) to ensure compatibility with subsequent stages, such as AI-based processing and mitigation generation.

The Mitigation Generator uses these structured datasets to dynamically generate prompts for **generative AI** models, applying prompt engineering techniques to ensure standardized inputs and consistent responses. It interacts with **LLMs** such as Anthropic's Claude or Meta's Llama through a dedicated API handling module, either submitting the prompts to a locally running Ollama server or using external APIs. The module then manages the submission and retrieval of responses and systematically stores the results in the centralized database.

An **evaluation-specific module** works in parallel to assess the outputs generated by the LLMs. This module follows the innovative **LLM-as-a-Judge** approach: it creates evaluation prompts that are processed by more advanced LLMs, using the generated mitigations as inputs [8]. The evaluation process provides quantitative ratings along with detailed qualitative explanations, which facilitate **human-in-the-loop** validation by cybersecurity experts.

Once the database is populated, the API Service exposes **REST APIs**, allowing HORSE components to access the enriched data. This approach ensures the data remains up-to-date while maintaining secure and efficient access.

4.2 Security and Data Collection

This section outlines the approach adopted for data collection within the CKB module while ensuring compliance with security protocols, especially when handling potentially sensitive information.

First and foremost, the CKB does not process or store any personal or sensitive data, such as Personally Identifiable Information (PII) or financial records. Consequently, there is no need for data encryption within the database itself. All data interactions and queries are exclusively managed through a dedicated **API server**, strategically positioned between the database and external users (HORSE components).

This architecture mitigates the risk of data tampering by preventing direct database access. Even if an attacker impersonates a HORSE component, they cannot insert misleading information into the database, as all requests are processed through the secure API server. To further enhance security, the API server performs rigorous input validation and sanitization to prevent data injection attacks. This process is efficiently managed using **Pydantic models**, which enforce strict input/output schema validation, ensuring data integrity by design.

The connection between the API server and the database is secured via **authentication**, requiring a username and password. These credentials are managed securely using **.**env files, which facilitate the separation of sensitive configurations from the codebase, avoiding hardcoding secrets directly. Additionally, in CI/CD pipelines within GitHub Actions, secrets are managed using dedicated tools, ensuring that sensitive data is never exposed in public or private repositories. As a best practice, .env files are listed in the .gitignore file to exclude them from the shared codebase.

4.3 Integration and Interfaces

To ensure effective interoperability, security, and maintainability within the HORSE architecture, the CKB has been designed with a focus on clear interface exposure and robust access control. The system provides well-structured APIs for interacting with the database, while also implementing best practices. The following sections provide an overview of the technologies used to expose these interfaces and the mechanisms in place to safeguard data access.

4.3.1 APIs and Format Exposed Through Interfaces

The CKB exposes REST APIs via its API Service module to enable secure and efficient communication between HORSE components and the centralized database of attacks and mitigations. During the design phase, specific frameworks, libraries, and architectural choices were carefully selected to ensure seamless integration of the CKB into the existing HORSE infrastructure without disrupting its architecture.

Key Tools and Technologies Used:

- **FastAPI:**
  The CKB utilizes FastAPI, a modern web framework for building APIs with Python, leveraging standard Python type hints. FastAPI is ideal for defining clean, standard-based interfaces that promote good integration practices. Main APIs:
  - GET */allattacks*: Retrieves the list of all attacks in the database. This endpoint allows for users to access the entire catalog of attacks and helps them finding the correct nomenclature for a specific attack.
  - POST */mitigations*: Retrieves a list of mitigations, including an AI-generated priority list and a natural language description. The request body must include the name of the attack.
- **Pydantic:** To ensure data consistency and validation, the CKB employs Pydantic, a widely used data validation library in Python. It defines data schemas to validate the input and output of API requests. Pydantic models also generate JSON schemas, making them easily integrable with FastAPI.
- **Swagger:** The API Service leverages Swagger, an integrated tool suite within FastAPI, to automatically generate visualization and documentation of REST APIs. Through the OpenAPI specification, users can access an intuitive interface that describes the API structure and functionality. The Swagger UI also supports API testing, enabling simulation of requests and responses to facilitate seamless integration with other HORSE components.
- **JSON Format:** To maintain consistency and ensure easy parsing and integration with HORSE components, all API responses and data exchanges use the JSON format. This standardized serialization method is commonly employed in web applications for data transmission between servers and clients, offering both human- and machine-readable outputs.

4.3.2 Access Control and Permissions

To access the API Service that exposes REST APIs for querying the database, robust access mechanisms and control measures are implemented to ensure security and data integrity.

- **Segregation of Duties:** The API server and the actual database containing attack and mitigation information are developed as separate Python modules. This architectural separation minimizes potential vulnerabilities by limiting direct access to the database. The database itself is protected, and access is strictly managed through authentication mechanisms. By segregating responsibilities, the exposure of sensitive data is significantly reduced.
- **Secure Management of Secrets:** Sensitive information required to access the database is securely stored in .env files. To ensure security, database queries are exclusively performed through the API service, preventing direct interaction with the database.**Principle of Least Privilege (PoLP):** Only the minimum necessary permissions are granted to users and HORSE components when accessing the

database. This practice limits the potential impact of compromised credentials or malicious actions, as each user or component has access only to the data and functions they specifically require.

4.3.3 Changes and Debugging from IT-1 to IT-2

The CKB was not part of the initial implementation (IT-1) since its relevance was discussed and approved only during the first phase of the project. It was officially introduced during the update of the architecture presented in Deliverable D2.4. Since then, the development of the CKB has followed an incremental process, undergoing continuous changes and debugging to ensure smooth integration with existing HORSE components.

Version 1.0 of the CKB:

- **Initial Release:** The first version of the CKB included two core components: the database and the API Service. The primary goal was to rapidly deliver a functional module capable of retrieving mitigation lists for specific attacks.
- **Data Population:** A curated subset of 10 predefined attacks and their corresponding mitigations were manually inserted into the database to provide a basic but functional dataset.
- **Generative AI Testing:** Early experiments with Generative AI were conducted to generate mitigation priority rankings and natural language descriptions. This process was human-curated, both during data generation and the subsequent evaluation phase.

Version 2.0 of the CKB:

- **Enhanced Generative AI Integration:** The second version significantly improved the database content by leveraging Generative AI. The focus was on automating data collection and enrichment.
- **Dynamic Attack Resolver:** A major addition was the Dynamic Attack Resolver, developed in Python, which automatically integrates data from the MITRE ATT&CK framework. This component populates the database with over 100 attacks, significantly expanding the initial dataset. The resolver is designed to periodically check for updates in the MITRE ATT&CK framework and download new data, keeping the CKB consistently up to date.
- **Mitigation Generator Module:** A new module was introduced to process the expanded database, utilizing cutting-edge pre-trained LLMs to:
    - Expand mitigation lists
    - Generate priority rankings
    - Create detailed, context-rich descriptions
- **Evaluation and Data Curation:** To assess the accuracy of LLM-generated data, the LLM-as-a-Judge approach was implemented. This method allows for automated evaluation and selection of the most reliable outputs.
- **Data Insertion Module:** The best responses generated by the LLMs are transformed and systematically inserted into the database, making them accessible to all HORSE components.

Version 3.0 of the CKB:

- **Error Handling Improvements:** This version focused on addressing integration issues that arose during interaction with other HORSE components, especially related to changes in the output JSON format.

- **Enhanced LLM Utilization:** Additional state-of-the-art LLMs were integrated to improve the consistency and accuracy of generated data, leveraging their ability to better synthesize complex mitigation strategies.
- **Security Enhancements:** Updated security measures were introduced to strengthen access control and authentication mechanisms, safeguarding the CKB against unauthorized access.
- **Performance Optimization:** Code refactoring and API interaction improvements led to more efficient processing and reduced response times. Performance bottlenecks were addressed through code cleansing and optimization of API calls.

5 Reliability, Trust, and Resilience Provisioning Framework

5.1 Overview

The Reliability, Trust, and Resilience Provisioning Framework (RTR) is a pivotal component within the HORSE platform (Figure 5.1), specifically engineered to bolster the security, reliability, and resilience of AI-assisted human-centric orchestration. Its core function involves processing high-level security intents originating from the Intent-Based Interface (IBI) and translating these into actionable mitigation measures, which are subsequently forwarded to the End-to-End Proactive Secure Connectivity Manager (ePEM). This process directly aligns with the principles of Intent-Based Networking (IBN), which significantly impacts network configuration management and security by automating the translation of high-level objectives into actionable network policies [9]. The RTR module is designed to support diverse input formats and provides a comprehensive mechanism for tracking the enforcement status of mitigation actions. The module can process mitigation actions in two different formats: a well-structured and predetermined JSON schema, and a natural language format. The dual format input works by initially attempting to validate the inputted mitigation action via the predetermined JSON schema. If the validation succeeds, it is processed and sent to the next component of the HORSE workflow, the ePEM. If the mitigation action fails to validate via this JSON schema, the RTR attempts to translate the command via an advanced NLP mechanism. If the confidence in the translation of the command is over a certain threshold, the mitigation action is formatted according to the next component's expected format and sent to it (IBI); otherwise, a translation error is raised and sent back to the component that originated the mitigation action.

.

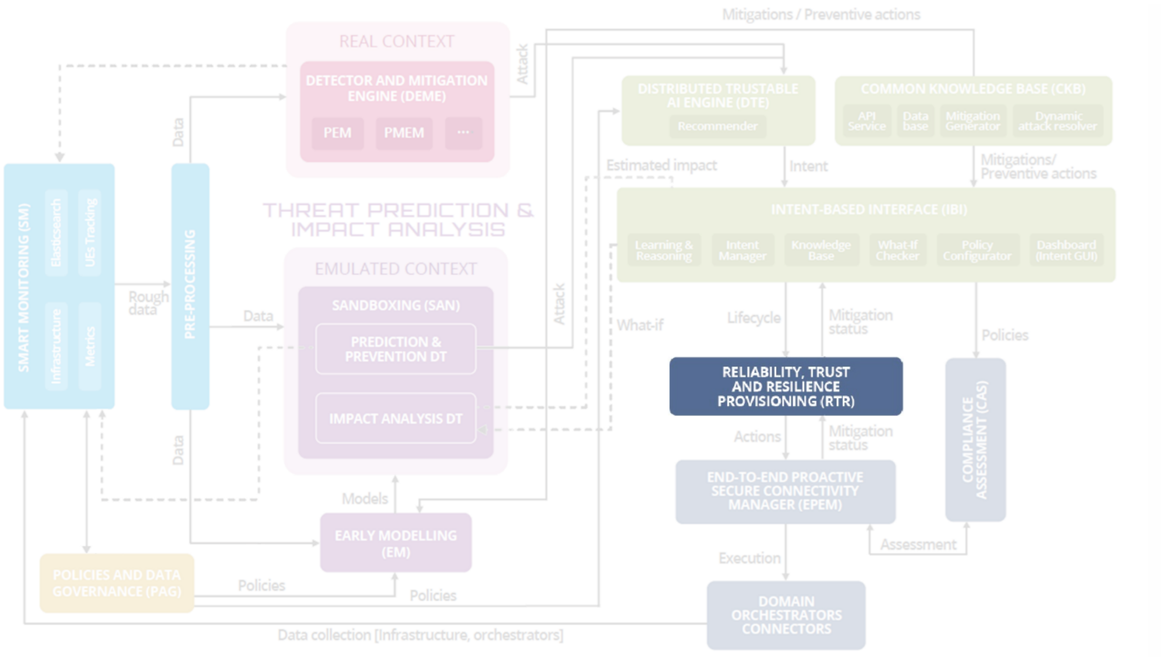


Figure 5.1 The RTR component within the HORSE architecture

5.2 Final Development Details

The IT-2 iteration of the RTR module builds on the IT-1 foundation, incorporating significant enhancements to improve functionality, performance, and reliability. Key developments include:

- **Error Correction and Handling**: Robust error-handling mechanisms were added to address inconsistencies in IBI inputs. The module validates incoming intents (JSON or natural language) against predefined schemas, correcting malformed data to ensure reliable processing.
- **Performance Optimization**: The codebase was optimized to reduce latency and improve throughput. Streamlined algorithms for intent parsing and mitigation action generation, coupled with efficient API interactions, enhance the module's scalability for high-volume workloads.
- **Advanced NLP Integration**: The module now supports advanced NLP techniques, allowing it to process natural language-based mitigation intents from the IBI (in the advanced version). These intents are translated into structured JSON mitigation actions and corresponding Ansible commands using a combination of large language models (LLMs) and predefined schema mappings.
- **Mitigation Action State Monitoring**: A new mechanism tracks the enforcement state of mitigation actions, with states including pending, enforced, and failed. The module logs enforcement results, including messages and timestamps, enabling real-time monitoring and post-incident analysis.
- **Containerized Execution**: The module is deployed in a Docker-based containerized environment, ensuring isolation and secure communication through monitored ports and endpoints.

These advancements ensure that the RTR module delivers a secure, efficient, and adaptable framework for threat mitigation within the HORSE platform

5.3 RTR Architecture and Core Workflow

The RTR module exposes well-defined RESTful APIs to facilitate seamless integration within the HORSE architecture.

- Northbound API (IBI Interaction)

This API serves as the primary interface for receiving mitigation intents from the IBI. It is designed to be flexible, accommodating both structured and natural language inputs.

- **Structured JSON:** Accepts mitigation intents conforming to a predefined schema, specifying parameters such as subnet address, mitigation action type, threat classification, and timeframes.
- **Natural Language:** Processes intents expressed in natural language, leveraging the internal NLP pipeline to extract and map key parameters to structured formats.

- Southbound API (ePEM Interaction)

This API manages the forwarding of translated mitigation actions to ePEM and provides the mechanism for status updates.

- **Action Forwarding:** Transmits the translated mitigation actions, which can be in the form of Ansible Security Playbooks (YAML-based) or structured JSON (aligned with

CACAO Security Playbook standards), to the ePEM module via secure HTTP POST requests.
- **Status Update Endpoint:** The RTR component provides an asynchronous status update endpoint that external orchestrators can call to notify about the execution state of a requested mitigation action. (e.g., `enforced`, `failed`, `pending`). This endpoint also allows for the inclusion of descriptive messages.

5.4 Security and Data Collection

Security is paramount in the RTR module, with IT-2 introducing enhanced measures to protect data and ensure operational integrity. The module continues to use TLS for secure data transmission between the IBI, ePEM, and itself, safeguarding against interception and tampering. Additional security enhancements include:

- **Containerized Environment**: The module operates in a containerized environment, communicating only through predefined, monitored ports. This minimizes the attack surface and ensures secure interactions with external components.
- **Authentication and Authorization**: Token-based authentication validates all API interactions, while access control lists enforce strict authorization policies. Only authorized entities (e.g., IBI, ePEM) can initiate or respond to requests.
- **Sensitive Data Handling**: The module minimizes personally identifiable information (PII) and applies anonymization techniques. Data retention policies align with GDPR and other regulatory standards.
- **Dynamic Threat Intelligence**: Integration with threat intelligence feeds, enriched by machine learning, enables proactive threat detection and adaptation to emerging risks.
- **Logging and Auditing**: Comprehensive logs capture all intents, mitigation actions, and enforcement states, including timestamps and results. Regular audits support forensic analysis and continuous improvement.
- **Data Validation**: Enhanced validation ensures that both JSON and natural language inputs adhere to expected schemas, preventing injection attacks or processing errors.

These measures ensure that the RTR module maintains a robust and trustworthy security posture.

5.5 Integration and Interfaces

The RTR module operates as a critical intermediary, ensuring that abstract security intents are transformed into concrete, executable mitigation actions. The workflow (displayed in Figure 5.2) is meticulously designed to handle various input formats, validate information, translate contexts, and maintain real-time status updates.

- Input Reception and Secure Communication

The RTR module's primary input originates from the Intent-Based Interface (IBI). Mitigation actions are received by the RTR API in one of two formats: a structured JSON format or a natural language format. Communication between the IBI and RTR is secured through a robust OAuth 2.0 workflow, ensuring authentication, authorization, and the integrity of transmitted data.

- Mitigation Action Registration

Upon receiving a mitigation action message, the RTR immediately registers this action into its internal registry. This registry is implemented using a MongoDB database, which serves as a persistent record of all incoming and processed mitigation actions, along with their evolving statuses.

- Format Validation and Translation

Following registration, the RTR proceeds to validate the format of the received mitigation action. This validation step is crucial for ensuring data integrity and correct processing.

- **Structured JSON Translation:** If the input format is a structured JSON representing a predefined mitigation action, the RTR translates the context of this action directly into an Ansible command. This translation is executed based on a set of pre-established rules that map specific JSON parameters to corresponding Ansible playbook directives.
- **Natural Language Processing (NLP) Translation:** If the input format is natural language, the translation process is executed through an advanced Natural Language Processing (NLP) pipeline. This NLP process is a sophisticated component, part of which has been detailed and published in academic papers, specifically referenced in [10]. The NLP pipeline extracts key entities, intents, and contexts from the natural language input to generate the appropriate Ansible command.

- Action Forwarding and Status Management

Once the translation is complete, the generated mitigation action (in the form of an Ansible command or structured JSON) is then forwarded to the ePEM module's API. Concurrently, the status of the mitigation action within the MongoDB registry is updated to reflect its current state (e.g., "forwarded" or "in progress").

- Status Update Mechanism for ePEM

The RTR API also exposes a dedicated function that allows the ePEM module to update the status of a specific mitigation action. This update mechanism is vital for maintaining real-time visibility into the enforcement lifecycle. ePEM can utilize this function to report when an action has been:

- **Enforced:** Successfully executed.
- **Failed to Execute:** Encountered an error during execution.
- **Changed Status:** Transitioned from "in progress" to any other state.

This API function includes a field for an accompanying message, which can convey an error message, a success message, or general informational updates regarding the action's enforcement.

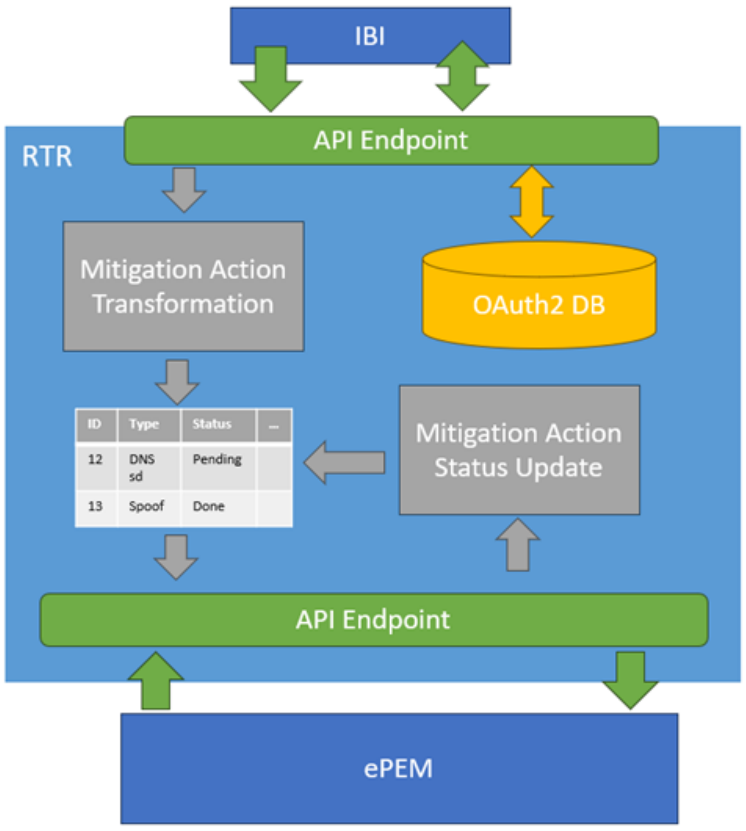


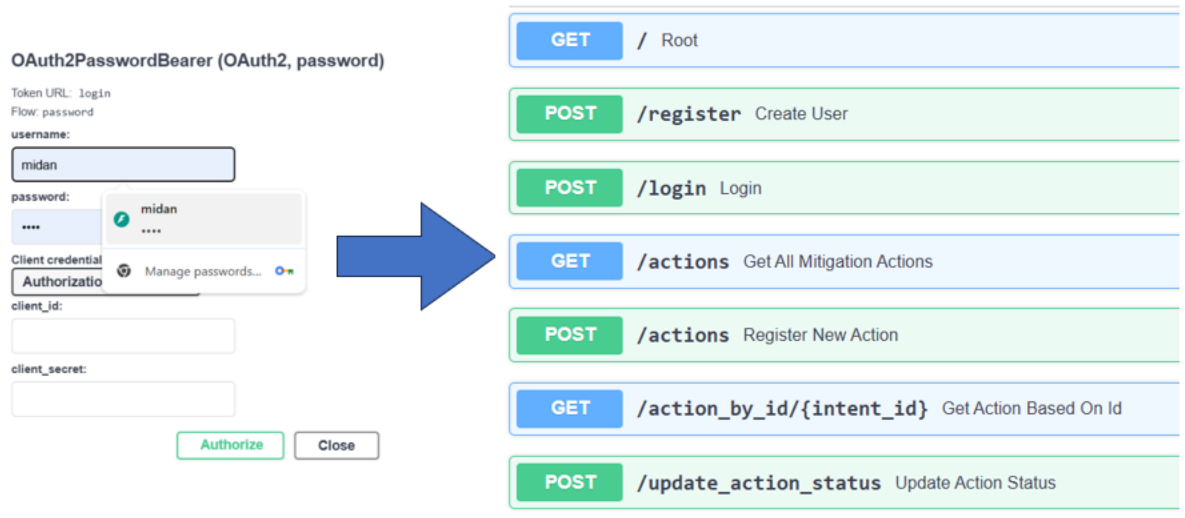
Figure 5.2 Overview of the RTR API Workflow

5.6 RTR API Endpoints

The RTR module exposes a set of RESTful API endpoints to manage mitigation actions and facilitate secure interactions with other components as displayed in Figure 5.2. These endpoints are built using the FastAPI framework, providing automatic interactive API documentation.

- **`/login`**: This endpoint facilitates user authentication. It typically accepts credentials (e.g., username and password) and, upon successful validation, returns an authentication token (e.g., a JWT) that subsequent API requests must include for authorization.
- **`/logout`**: This endpoint allows authenticated users to invalidate their current session or token, effectively logging them out of the system.
- **`/register` (POST)**: This function is responsible for receiving and registering new mitigation actions. It accepts structured JSON or natural language inputs from the IBI, initiates the registration process in the MongoDB database, and triggers format validation and translation.
- **`/action_by_id/{intent_id}` (GET)**: This endpoint allows for the retrieval of a specific registered mitigation action based on its unique `intent_id`. It returns the details of the action, including its current status and any associated messages.
- **`/actions` (GET)**: This endpoint provides a comprehensive list of all mitigation actions currently registered in the MongoDB database. It can be used for monitoring and auditing purposes, offering an overview of all processed and pending actions.
- **`/update_action_status` (POST)**: This critical endpoint is utilized by the ePEM module to update the status of a specific mitigation action. It accepts the `intent_id` and the new status (e.g., "enforced", "failed", "in_progress"), along with an optional

message (error, success, or info). This ensures real-time tracking of mitigation action enforcement.



OAuth2PasswordBearer (OAuth2, password)

Token URL: `login`

Flow: `password`

username: midan

password: ••••

Client credential Authorizatio

client_id:

client_secret:

GET `/` Root

POST `/register` Create User

POST `/login` Login

GET `/actions` Get All Mitigation Actions

POST `/actions` Register New Action

GET `/action_by_id/{intent_id}` Get Action Based On Id

POST `/update_action_status` Update Action Status

Figure 5.3 Screenshot of RTR API Documentation (FastAPI /docs). Automatically generated API documentation from FastAPI's /docs endpoint, illustrating the available API endpoints and their specifications.

5.7 Technical Implementation and Deployment

The RTR module is engineered for modern, scalable, and secure deployment, leveraging contemporary software development practices.

- Framework and Containerization

The application logic for the RTR module is developed using the FastAPI framework, known for its high performance and ease of use in building APIs with Python (a screenshot of this FastAPI deployment is shown in Figure 5.3). For deployment and operational consistency, the entire application is fully containerized using Docker. This ensures that the RTR module, along with all its dependencies, runs in an isolated and consistent environment. Docker Compose is utilized to orchestrate the deployment, managing the RTR service, its MongoDB database, and any other auxiliary services, facilitating easy setup and scaling.

- Security Enhancements

Security is a paramount concern for the RTR module, with multiple layers of protection integrated into its design and operation:

Secure Communication: All data transmission between the IBI, RTR, and ePEM is secured through a two-layered approach. Firstly, all communications occur within a secure internal network, minimizing external exposure. Secondly, an OAuth 2.0 workflow is consistently applied across all API interactions, providing robust authentication and authorization. This dual-layer security safeguards against interception, tampering, and unauthorized access.

Containerized Environment: Operating within a Docker container minimizes the attack surface. The module communicates only through predefined, monitored ports, restricting unauthorized access.

Authentication and Authorization: Token-based authentication validates all API interactions, ensuring that only authorized entities (e.g., IBI, ePEM) can initiate or respond to requests. Access control lists (ACLs) enforce strict authorization policies.

Logging and Auditing: Comprehensive logs capture all intents, mitigation actions, and enforcement states, including timestamps and results. Regular audits support forensic analysis and continuous improvement of the security posture.

5.8 Deployment, Operation and Maintenance Guidelines

The RTR module's deployment, operation, and maintenance are designed to ensure continuous functionality and adaptability.

1. **Deployment Guidelines**:
    a. **Environment Setup**: Verify dependencies (e.g., Python, Django, Docker) and configure environment-specific settings via configuration files.
    b. **Containerization**: Deploy the module in a Docker container, ensuring secure port configurations and resource allocation for scalability.
    c. **Scalability**: Design the architecture to support horizontal scaling, leveraging container orchestration (e.g., Kubernetes) for high workloads.
2. **Operation Guidelines**:
    a. **Real-Time Monitoring**: Monitor performance metrics (e.g., response times, resource utilization) and mitigation action states using integrated logging tools.
    b. **Incident Response**: Establish procedures for handling security incidents or operational disruptions, with clear roles for rapid resolution.
    c. **State Tracking**: Regularly review enforcement state logs to ensure mitigation actions are executed as intended.
3. **Maintenance Guidelines**:
    a. **Patch Management**: Apply regular security patches to the module and its dependencies to address vulnerabilities.
    b. **Backup and Recovery**: Maintain backups of configurations and logs, with tested restoration procedures for data recovery.
    c. **Version Control**: Use version control to track code changes, documenting enhancements and bug fixes for traceability.
    d. **Continuous Improvement**: Incorporate feedback from WP5 integration tests and update NLP models to improve intent processing accuracy.

5.9 Changes and Debugging from IT-1 to IT-2

The transition from IT-1 to IT-2 addressed issues identified during preliminary integration in WP5, resulting in a more robust and efficient RTR module:

- **Error Handling**: IT-1 revealed issues with malformed IBI inputs. IT-2 introduced enhanced validation and error correction for both JSON and natural language inputs, reducing processing errors.
- **Performance Optimization**: IT-1 performance bottlenecks were resolved through code refactoring and optimized API interactions, reducing latency.
- **NLP Integration**: The advanced NLP version was developed to handle natural language inputs, addressing IT-1 limitations where only structured JSON was supported. Debugging ensured accurate intent translation into JSON and Ansible commands.

- **State Monitoring**: IT-1 lacked enforcement state tracking. IT-2 introduced the state monitoring mechanism, validated through WP5 tests to ensure reliable state updates and logging.
- **Endpoint Resilience**: Unresponsive endpoints in IT-1 were mitigated with retry logic and payload reconfiguration, improving communication reliability.
- **Security Enhancements**: The containerized environment and monitored ports resolved IT-1 vulnerabilities, ensuring secure execution and communication.

These changes, validated through WP5 integration tests, ensure that the RTR module is fully prepared for final deployment in the HORSE platform.

6 End-to-End Proactive Secure Connectivity Manager

6.1 Overview

The End-to-End Proactive Secure Connectivity Manager (ePEM) plays a pivotal role in the HORSE security infrastructure, orchestrating actions and providing observability over various components that constitute the end-to-end services secured within the HORSE security perimeter [11]. To enhance its effectiveness, several improvements can be considered based on the evolving landscape of network security and operational efficiency.

One potential improvement is the integration of advanced machine learning algorithms for threat detection and response. By leveraging real-time data analytics, ePEM could enhance its ability to identify anomalies and respond to security incidents more swiftly, thereby reducing the response time to potential threats. This aligns with the current trend of utilizing AI-driven solutions to bolster cybersecurity measures, ensuring that the infrastructure remains resilient against emerging threats.

Additionally, enhancing the modular architecture of ePEM could facilitate better scalability and adaptability. By allowing for more flexible integration of new network functions and services, ePEM can more effectively manage the complexities of diverse network environments. This modular approach would enable quicker deployment of new features and improvements, ensuring that the system can evolve in tandem with technological advancements and user needs.

Furthermore, implementing a more robust feedback mechanism from users and stakeholders could provide valuable insights into the operational performance of ePEM. Regularly soliciting input on the system's functionality and user experience would inform future updates and enhancements, fostering a user-centric approach to development. This continuous improvement cycle would not only enhance user satisfaction but also ensure that ePEM remains aligned with the dynamic requirements of the HORSE project.

In summary, by integrating advanced threat detection technologies, enhancing modular architecture, and establishing robust feedback mechanisms, ePEM can significantly improve its operational effectiveness and security posture within the HORSE infrastructure.Final Development Details

The ePEM is a cornerstone of the HORSE security infrastructure, designed to manage secure connectivity across complex, distributed, and heterogeneous systems. As outlined in the document, ePEM serves as a central architectural element that orchestrates actions and provides observability over various components within the HORSE security perimeter. This section delves into the final development details of ePEM, highlighting its functionalities, integration capabilities, and the technologies employed to ensure its effectiveness.

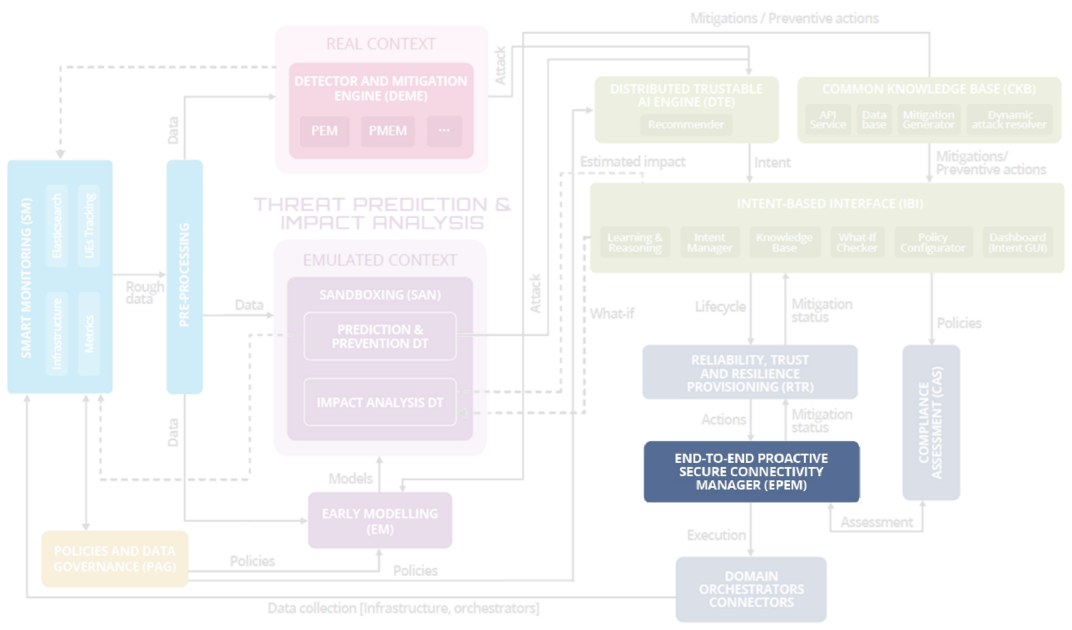


Figure 6.1 The ePEM module highlighted within the HORSE architecture

6.1.1 Central Coordination and Observability

At the heart of ePEM's functionality is its role as a central coordinator, ensuring that all elements and artifacts within the HORSE ecosystem operate securely and harmoniously. ePEM orchestrates actions and maintains observability across diverse components, which is crucial for managing the intricate interplay of resources and services. This orchestration is vital for ensuring the resilience and security of the infrastructure, particularly in environments characterized by rapid changes and evolving threats.

6.1.2 Topology Information Management

ePEM maintains a comprehensive database of the logical topology of the distributed infrastructure, which includes information at both the wide-area connectivity and Virtualized Infrastructure Manager (VIM) levels. This database records details about the orchestrators and controllers responsible for governing these components, with each topological entity annotated with resource constraints and access levels. Such detailed management of topology information is essential for efficient resource management and access control, enabling ePEM to respond dynamically to the needs of the network.

6.1.3 Management of NFV and Applicative Services

In addition to topology management, ePEM actively participates in the management of Network Function Virtualization (NFV) and applicative services. It keeps track of the localization and degrees of freedom granted by VIMs to Virtual Network Functions (VNFs) and application components within the HORSE security perimeter. This continuous awareness allows ePEM to update information related to NFV and applicative services based on exposure levels provided by domain orchestrators and controllers, ensuring that the system remains agile and responsive to operational demands.

6.1.4 Data Homogenization and Simplification

A significant aspect of ePEM's role is to homogenize and consolidate data from diverse sources after the pre-processing module. This simplification provides a unified, coherent view of the services managed by the HORSE platform, significantly aiding in decision-making and security management. By streamlining data from various components, ePEM enhances the overall efficiency of the orchestration process, allowing for quicker and more informed responses to security incidents.

6.1.5 Meta-Actions for Security

ePEM autonomously acquires and exposes a repertoire of action types that can be applied to each artifact or group of artifacts within the end-to-end services. These meta-actions assist in formulating contingency plans for security threats and vulnerabilities, derived from a collection of pre-designed Blueprint profiles that encapsulate the functional behaviour of diverse network elements. This proactive approach to security ensures that ePEM can respond effectively to potential threats, maintaining the integrity and availability of the network.

6.1.6 Blueprint Profiles

Blueprint profiles within the context of ePEM encompass a diverse range of complex network elements, including 5G/6G radio mobile networks, distributed firewalls, and monitoring overlay systems. These profiles serve as comprehensive templates that outline specific actions and primitives essential for orchestrating activities throughout different phases of network management. By providing a standardized framework, Blueprint profiles empower ePEM to coordinate and execute operations seamlessly, ensuring efficiency and coherence in the management of various network elements.

6.1.7 Modular Architecture

The modular and flexible architecture of ePEM allows for easy extension to support various Network Functions (xNF) and ecosystems. At the foundation of this architecture lies the metamodel, specifically designed to augment extensibility and flexibility while driving clear interaction patterns among the different internal modules during Lifecycle Management (LCM) operations. This modularity is crucial for adapting to the evolving needs of the network and integrating new functionalities as they become necessary.

6.1.8 Integration with Domain Orchestrators

ePEM collaborates with Domain Orchestrators and controllers to enhance the efficiency and intelligence of operations. These external entities bring different degrees of automation and intelligence to the management of resources and artifacts throughout their lifecycle. This collaboration is essential for achieving a cohesive orchestration environment, where various components can work together seamlessly to deliver secure and reliable services.Security and Data Collection

Security and data collection are foundational pillars in the design and operation of the End-to-End Proactive Secure Connectivity Manager (ePEM) within the HORSE architecture. The ePEM employs a comprehensive cybersecurity toolkit, including VyOS and Suricata, to ensure robust security measures tailored to the specific demands of the network

VyOS serves as a cornerstone of ePEM's security architecture, providing a versatile and scalable solution that integrates seamlessly into both standard hardware and virtualized environments. This adaptability allows ePEM to establish, manage, and safeguard secure communication paths across the network, enhancing its capability to enforce network segmentation and sophisticated firewall policies. By leveraging VyOS, ePEM can effectively protect against unauthorized access attempts, thereby fortifying the overall security infrastructure.

In conjunction with VyOS, Suricata functions as an Intrusion Detection and Prevention System (IDPS), continuously monitoring network traffic for potential security incidents or anomalies. This proactive surveillance is crucial for identifying threats before they escalate, contributing to a robust and responsive security framework within the ePEM . Suricata's ability to initiate preventive actions upon detecting a security threat adds an extra layer of defense, minimizing the potential impact of security incidents and ensuring the continuity of secure network operations.

The ePEM's approach to data collection is sophisticated, facilitated by its Topology Manager, which orchestrates a network of Prometheus servers for efficient data gathering. These Prometheus instances are strategically positioned to collect critical metrics from diverse Network Functions (xNFs) across the network. The integration of Prometheus servers allows ePEM to dynamically deploy metric exporters on xNFs, tailoring the data collection process to align with specific operational requirements and evolving network conditions.

Moreover, the data collection framework is designed to be adaptable and responsive, with the nature and volume of collected data closely linked to the configuration of the exporters. This modular approach ensures that ePEM can flexibly adjust its data collection strategies, accommodating the unique characteristics of different xNFs within the network. By establishing a comprehensive framework for data management, ePEM enhances its analytical capabilities, fostering an informed, secure, and agile environment within the HORSE architecture.

Security measures during data collection are paramount. The ePEM employs industry-standard encryption protocols, such as TLS, to ensure the confidentiality and integrity of data exchanged between interconnected components. This safeguarding of communication channels mitigates the risk of data interception and tampering, reinforcing the security posture of the entire system. Additionally, rigorous authentication and authorization mechanisms are integral to the RTR module's security architecture, ensuring that only authorized entities can initiate and respond to data requests.

In summary, the integration of VyOS and Suricata within the ePEM framework, combined with a sophisticated data collection methodology facilitated by Prometheus, underscores the commitment to maintaining a secure and resilient network infrastructure. The proactive measures taken to safeguard data during collection, transmission, and storage align with industry standards and best practices, ensuring that the HORSE architecture remains robust against emerging threats and vulnerabilities.

6.2 Integration and Interfaces

6.2.1 APIs and Format Exposed Through Interfaces

ePEM within the HORSE architecture employs a robust set of RESTful APIs that serve as a foundational element for seamless interaction within the system. These APIs are designed to be user-friendly and standardized, facilitating dynamic resource provisioning, proactive security policy enforcement, and data retrieval. The simplicity and scalability of these APIs empower external systems and services to engage with ePEM effortlessly, ensuring efficient management and utilization of network resources.

ePEM's southbound APIs establish essential connections with a diverse array of network elements and devices, including physical and virtual components such as switches, routers, and virtualized network functions (VNFs). This connectivity is crucial for orchestrating resources across the network, ensuring cohesive management and utilization. Conversely, the northbound interfaces enable collaboration with higher-level services and orchestrators, such as the Open-Source MANO orchestrator and cloud services. This dual approach allows ePEM to translate high-level service requests into actionable network directives, facilitating seamless integration within a broader ecosystem.

To ensure interoperability, ePEM supports various data formats, including JSON, XML, and YAML. This flexibility allows for structured and interoperable data exchange with external entities, which is particularly valuable in heterogeneous environments where diverse systems coexist. The use of standard communication protocols such as HTTP/HTTPS, MQTT, and SNMP further enhances ePEM's ability to communicate across different technology stacks, contributing to a cohesive and interconnected network environment.

The system implements robust authentication, authorization, and encryption mechanisms to safeguard the APIs. Access to sensitive functionalities and data is strictly controlled, preventing unauthorized access and ensuring that data transmitted through APIs is encrypted to protect against potential threats such as eavesdropping and tampering. This commitment to security ensures a trustworthy and resilient communication framework within the ePEM ecosystem.

6.2.1.1 Topology

Table 6.1 ePEM's API endpoints (Topology)

HTTP Method	Endpoint	Description
GET	/v1/topology	Get information regarding managed topology.
POST	/v1/topology	Create the topology.
DELETE	/v1/topology	Delete Topology.

6.2.1.2 Topology – VIM

Table 6.2 ePEM's API endpoints (Topology - VIM)

HTTP Method	Endpoint	Description
GET	/v1/topology/vim/{vim_id}	Get the VIM data from OSM.
POST	/v1/topology/vim	Create the VIM in OSM.
PUT	/v1/topology/vim	Update the VIM in OSM.
DELETE	/v1/topology/vim/{vim_id}	Delete the VIM from OSM.

6.2.1.3 Topology – Network

Table 6.3 ePEM's API endpoints (Topology - Network)

HTTP Method	Endpoint	Description
GET	/v1/topology/network/{network_id}	Get the network data.
POST	/v1/topology/network	Create the network.
PUT	/v1/topology/network	Update the network.
DELETE	/v1/topology/network/{network_id}	Delete the network.
POST	/v1/topology/network/{network_id}/add/pool	Add pool network.
DELETE	/v1/topology/network/{network_id}/del/pool	Delete the pool network.
POST	/v1/topology/network/{network_id}/k8s/reserve	Reserve range to k8s Cluster. Takes one or more allocation pools from the topology network and partially assigns them to the k8s cluster, depending on the length requested.
DELETE	/v1/topology/network/{network_id}/k8s/release	Release the reserved range from the k8s cluster and the topology network.

6.2.1.4 Topology – Router

Table 6.4 ePEM's API endpoints (Topology - Router)

HTTP Method	Endpoint	Description
GET	/v1/topology/router/{router_id}	Get the router data.
POST	/v1/topology/router	Create the router.
DELETE	/v1/topology/router/{router_id}	Delete the router.

6.2.1.5 Topology – Kubernetes

Table 6.5 ePEM's API endpoints (Topology - Kubernetes)

HTTP Method	Endpoint	Description
GET	/v1/topology/kubernetes	Get the k8s data.

GET	/v1/topology/kubernetes/{cluster_id}	Get the k8s data of the cluster.
POST	/v1/topology/kubernetes_external	Add a external k8s cluster to the topology.
PUT	/v1/topology/kubernetes/update	Update NFVCL k8s cluster.
DELETE	/v1/topology/kubernetes/{cluster_id}	Delete the k8s cluster.

6.2.1.6 Blueprint

Table 6.6 ePEM's API endpoints (Blueprint)

HTTP Method	Endpoint	Description
GET	/nfvcl/v2/api/blue	Get all the blueprint data.
GET	/nfvcl/v2/api/blue/{blueprint_id}	Get the data of the blueprint.
DELETE	/nfvcl/v2/api/blue/{blueprint_id}	Delete blueprint.
DELETE	/nfvcl/v2/api/blue/all/blue	Delete all blueprints.
PATCH	/nfvcl/v2/api/blue/protect/{blueprint_id}	Protect the blueprint.

6.2.2 Access Control and Permissions

Access control and permissions are critical components of the ePEM architecture, ensuring that only authorized users can perform specific actions and access sensitive data. ePEM implements a robust Role-Based Access Control (RBAC) system to manage permissions in an efficient way. User roles, such as administrators, operators, and tenants, are assigned granular access privileges, which help reduce the risk of unintended configuration modifications or unauthorized data retrieval.

The access control framework is policy-driven, with access rights defined according to user roles. These policies dictate what actions users can perform and what data they can access, allowing for customizable and adaptable security measures that align with specific operational needs. In a multi-tenant environment, ePEM provides the capability to manage access control and permissions for different tenant organizations, ensuring that each tenant has its own isolated space within the infrastructure. This isolation safeguards the privacy and security of their resources and data.

Audit trails and logging are integral to ePEM's access control framework. The system logs and audits access and permission changes, providing a record of accountability and traceability. In the event of security incidents or policy infringements, these audit logs offer valuable insights into who initiated system access, the specific actions performed, and when they occurred. This comprehensive logging mechanism not only aids in identifying potential security incidents but also contributes to continuous improvement initiatives within the ePEM architecture.

In summary, the integration of APIs and the implementation of access control and permissions within ePEM are essential for maintaining a secure and efficient orchestration platform. By leveraging robust APIs and a well-defined access control framework, ePEM ensures that the HORSE infrastructure remains resilient, adaptable, and secure against emerging threats and vulnerabilities.

6.3 Deployment, Operation and Maintenance Guidelines

The deployment, operation, and maintenance of the ePEM module are crucial for ensuring the continuous security and functionality of the HORSE platform. To facilitate seamless integration and sustained effectiveness, comprehensive guidelines are outlined.

Firstly, environment preparations are essential, ensuring that the target environment aligns with the system requirements specified for the ePEM module, including the availability of necessary dependencies like the Python runtime environment. Configuration management practices should be implemented to streamline deployment processes, allowing for easy adaptation to diverse scenarios.

During operation, continuous monitoring mechanisms must be established to track the ePEM module's performance, including key performance indicators such as response times and resource utilization. Incident response procedures should be documented to guide the team in the event of security incidents, clearly defining roles and responsibilities.

For maintenance, a robust patch management strategy is vital to keep the ePEM module updated, mitigating vulnerabilities. Additionally, establishing backup and recovery procedures ensures the safeguarding of critical data and configurations, while version control systems help track changes in the ePEM module's codebase. These guidelines collectively contribute to the effective management of the ePEM module within the HORSE architecture.

6.4 Changes and Debugging from IT-1 to IT-2

Topology Info Management

Removed support for Open-Source Mano. Now everything is handled directly from ePEM:

- **VIM** (*Virtualized Infrastructure Manager*) - added support for *Proxmox* in addition to OpenStack (prev. supported).
- **K8S** - now ePEM interacts directly with Kubernetes using K8S APIs.

Performance metrics

Now ePEM collects metrics when a Blueprint is created or a *day-2* is called and saves the time needed for each operation to the database.

Management of NFV/Application Services

The creation of NFV/App Services has been abstracted such that it is easier for the intermediate user to develop a custom Blueprint in the ePEM. Implemented a system that enables users to write Configurators to support custom Devices in addition to *Amarisoft* and *Liteon* gNBs.

Data Homogenization and Simplification

For each 5G blueprint type (Core, RAN, UPF), the request format is the same for the different implementations allowing to replicate the same environment, but for different implementations.

Meta-Actions for Security

Added beta system for user authentication when using ePEM APIs thought the use of tokens obtained on login. Planned a system, to be implemented in the future, to group users into groups with RBAC.

Enhancements to Network Function Virtualization Orchestrators (NFVO) and VIMs

Removed unstable and buggy OSM in favor of a custom ad-hod designed framework, internal to the ePEM, to interact with VIMs. Performance indicators, in terms of time, have increased drastically (deployment time has decreased).

Blueprint Profiles

5G- improved 5G blueprint system, now most of the code is abstracted in generic classes greatly reducing the work needed to develop a new blueprint, currently supported 5G blueprints:

Core

- SDCore
- OAI
- Free5GC
- HPE Athonet (only configuration of an already installed instance)

RAN

- UERANSIM
- OAI (only the simulated UE and USRP version not yet the ORAN one), support all the available split level

PDU configurator

- Amarisoft gNB
- Liteon AIO gNB

Kubernetes Blueprint

Other blueprints: generic Ubuntu VM.

Deployment

- Added Docker container for ePEM
- Added Docker compose
- Added Helm Chart

Unit test

Create some infrastructure to allow an easy creation of unit tests

6.4.1 Major Architectural Changes

Blueprint System Evolution

- **BlueprintNG Introduction**. Complete rewrite of the blueprint system with improved lifecycle management, parent-child relationships between blueprints, and better resource management.
- **Day 2 Operations**. Added comprehensive day 2 operations with call history tracking
- **Blueprint Protection**. Implemented blueprint protection mechanisms and concurrent operation prevention.
- **Performance metrics collection**. Metrics regarding the blueprints creation and day2s are recorded to the database.

5G Blueprints

- **Generic 5G Blueprint**. Introduced a generic 5G core blueprint supporting multiple 5G implementations.
- **Multiple 5G Core Support**. Added support for SD-Core, OpenAirInterface (OAI), Free5GC, and UERANSIM.
- **Kubernetes Integration**. Full K8S support for 5G network functions with Helm chart management.
- **RAN and UE Blueprints**. Added Radio Access Network and User Equipment blueprint implementations.

Infrastructure Provider Expansion

- **Proxmox Support**. Added Proxmox virtualization provider with VM configuration and SDN support.
- **Enhanced OpenStack**. Improved OpenStack provider with better project management, network handling, and authentication.
- **Kubernetes Provider**. Comprehensive K8S provider with node management, deployment scaling, and *Multus* networking.

Core Platform Improvements

Authentication & User Management

- **User Management System**. Complete user authentication system with JWT tokens.
- **Role-Based Access**. Implemented user roles and permissions.
- **Token Refresh**. Compliant token refresh mechanisms.

Networking Enhancements

- **Multus Support**. Advanced Kubernetes networking with *Multus* CNI.
- **IP Pool Management**. Sophisticated IP allocation and reservation system.
- **Network Topology**. Enhanced network topology management for K8S clusters.

Code Quality & Architecture

- **Core-REST Split**. Separated core components from REST API layer.

DevOps & Deployment

- **Configuration Management**. Enhanced configuration with environment variable support.

Performance & Reliability

Performance Monitoring

- **Metrics Collection**. Comprehensive performance metrics for blueprint operations.
- **REST API**. Added endpoints for retrieving performance data.
- **Error Tracking.** Enhanced error handling and logging systems.

7 Domain Orchestrator Connectors

7.1 Overview

The Domain Orchestrator Connectors (DOC) module is a pivotal component and an essential API within the HORSE platform. It primarily functions as an intermediary that receives security mitigation actions in a structured JSON format, typically originating from the Reliability, Trust, and Resilience Provisioning Framework (RTR) via the End-to-end Proactive Secure Connectivity Manager (ePEM). Responsible for orchestrating tasks across diverse network segments, including RAN, Core, Transport, Edge (near and far), and Cloud, as depicted in Figure 7.1, the DOC translates these received JSON actions into various specific formats suitable for the respective network managers within each segment.

Developed under the leadership of 8BELLS with contributions from CNIT, ETI, and ZORTE, the DOC is designed with high modularity, allowing new network managers and their distinct communication logics to be seamlessly integrated or removed. This provides a unified resource stratum that blurs orchestration boundaries between these segments. This multi-domain orchestration capability is crucial for effectively managing and controlling resources across disparate network infrastructures, aligning with advancements in multi-domain network slicing orchestration architectures and federated resource control[12].

Serving as a SouthBound Interface for the ePEM (which is based on the Open Source MANO orchestrator), the DOC ensures seamless management and orchestration of resources, supporting cross-domain trust mechanisms, and enabling the secure integration of multi-stakeholder infrastructures by pushing translated mitigation actions for enforcement.

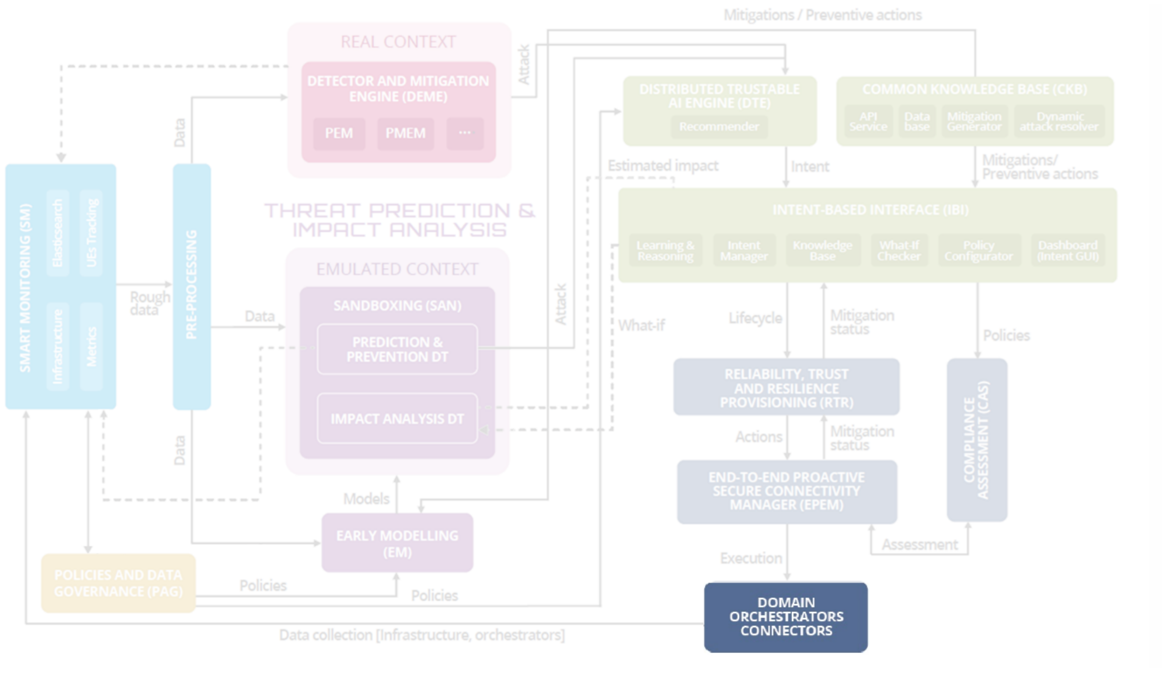


Figure 7.1 The DOC component within the HORSE architecture

7.2 Main Functionalities

The DOC module facilitates the enforcement of security mitigation actions across various network segments by interfacing with their respective network managers. Its key functionalities include:

- **Mitigation Action Processing and Translation:** The DOC functions as an API that receives mitigation actions in a structured JSON format, as created by the RTR component. It then translates these actions into various specific formats suitable for the diverse network managers with which the DOC communicates. This adaptable translation mechanism ensures that security directives can be effectively applied across heterogeneous network environments.

- **Flexible Network Management Integration:** The module's design emphasizes modularity, enabling the seamless addition or removal of new networks and their respective managers, along with their unique communication logics. This flexibility allows the DOC to easily integrate with and orchestrate mitigation actions across a dynamically evolving set of networks, ensuring security enforcement can scale and adapt to different infrastructure requirements.

- **Mitigation Action Enforcement Tracking:** For every mitigation action it receives, translates, and sends, the DOC maintains a detailed internal record within its dedicated database. This internal record tracks the status of enforcement for each action (e.g., pending, enforced, failed), providing comprehensive oversight of the security posture. It then pushes enforcement results, including relevant messages and timestamps, to the RTR Framework for state updates and holistic platform awareness.

7.3 Integration and Interfaces

The DOC module integrates with the HORSE architecture through well-defined interfaces, acting as a SouthBound Interface for the ePEM and communicating with network managers across various domains.

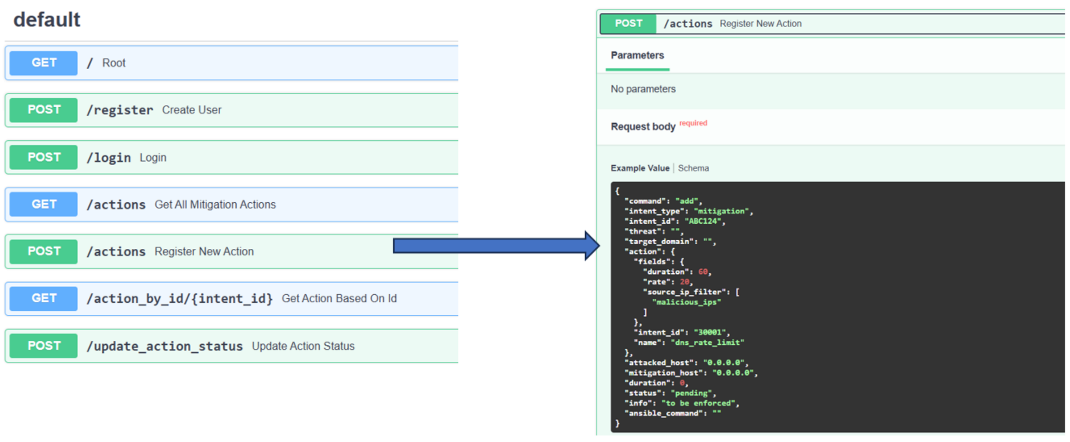
- **Northbound Interface (ePEM Integration)**: The DOC receives mitigation actions from the ePEM via RESTful APIs. Inputs are structured as JSON messages, which encapsulate details such as mitigation actions, threat information, and time frames.
- **SouthBound Interface (Network Managers)**: The DOC translates mitigation actions into domain-specific formats and sends them to RAN, Core, Transport, Edge, and Cloud network managers via HTTP POST requests. It supports diverse protocols and formats to ensure compatibility with each segment’s management system.
- **Enforcement Status Feedback**: The DOC implements a mechanism that tracks and stores the status of each mitigation action internally. It collects enforcement status updates (pending, enforced, failed) from network managers, including messages and timestamps. This tracking and storage mechanism is designed to be resilient, maintaining accurate records even in the presence of temporary network failures or communication disruptions. These results are pushed to the RTR component via a dedicated API endpoint, enabling the RTR to update mitigation action states accordingly.

7.4 API Functionality and Interfaces

The DOC module serve as a central hub, expertly managing how security actions get carried out across all sorts of different network parts. It's essentially our main API gateway, making sure everything runs smoothly between our high-level control systems and the actual network infrastructure down below.

- **Northbound API (Talking to ePEM):** Looking 'north,' the DOC talks to our End-to-end Proactive Secure Connectivity Manager (ePEM) through its Northbound API. This is where the DOC receives all the security actions it needs to carry out. These come in as neat, structured JSON messages via RESTful APIs, packed with all the important details like what the mitigation is, what threat it's addressing, and when it needs to happen, the API endpoint that can be utilized by the ePEM are displayed in Figure 7.2.

- **Southbound APIs (Talking to Network Managers):** Now, looking 'south,' the DOC connects with all the different network managers (like those for RAN, Core, Transport, Edge, and Cloud) using its flexible Southbound APIs. When the DOC gets a mitigation action, it doesn't just forward it as-is. Instead, it cleverly translates that generic JSON instruction into the exact format and protocol each specific network segment understands. These perfectly tailored actions are then sent off to their respective managers, usually through HTTP POST requests. What's really clever about this design is how modular it is: we can easily plug in or unplug new network managers and even customize how the DOC talks to each one. This flexibility means our system can adapt and scale according to changes in the network landscape.


default

GET / Root
POST /register Create User
POST /login Login
GET /actions Get All Mitigation Actions
POST /actions Register New Action
GET /action_by_id/{intent_id} Get Action Based On Id
POST /update_action_status Update Action Status

POST /actions Register New Action

Parameters

No parameters

Request body required

Example Value | Schema

```
{
  "command": "add",
  "intent_type": "mitigation",
  "intent_id": "ABC124",
  "threat": "",
  "target_domain": "",
  "action": {
    "fields": {
      "duration": 60,
      "rate": 20,
      "source_ip_filter": [
        "malicious_ips"
      ]
    },
    "intent_id": "30001",
    "name": "dns_rate_limit"
  },
  "attacked_host": "0.0.0.0",
  "mitigation_host": "0.0.0.0",
  "duration": 0,
  "status": "pending",
  "info": "to be enforced",
  "ansible_command": ""
}
```

Figure 7.2 DOC API endpoints

7.5 Changes and Debugging from IT-1 to IT-2

The transition from IT-1 to IT-2 involved significant debugging and enhancements to address issues identified during preliminary integration in WP5:

- **Error Handling**: IT-1 revealed challenges with mismatched mitigation action formats for different network segments. IT-2 introduced robust validation and translation mechanisms to ensure compatibility, reducing errors.
- **Performance Optimization**: IT-1 performance bottlenecks in multicluster environments were resolved through code refactoring and optimized API interactions, improving orchestration latency.
- **Enforcement Status Monitoring**: IT-1 lacked a mechanism to track mitigation action enforcement. IT-2 introduced status monitoring (pending, enforced, failed) and feedback loops with the RTR, validated through WP5 tests to ensure accurate state updates.

- **Endpoint Resilience**: Unresponsive network managers in IT-1 were addressed with retry logic (e.g., exponential backoff) and payload reconfiguration, improving communication reliability across segments.

These changes, validated through WP5 integration tests, ensure that the DOC module is fully prepared for final deployment, providing a secure, scalable, and efficient orchestration solution for the HORSE platform.

8 Compliance Assessment Procedures

The Compliance Assessment System (CAS) primarily aims to ensure all HORSE actions comply with predefined security and privacy policies. It also verifies that decisions from the Trustable AI engine align with regulations. This systematic validation aligns with comprehensive frameworks for counteracting cyber threats and building future cybersecurity, which emphasize robust policy enforcement and resilient systems [13]. CAS acts as a critical checkpoint, ensuring AI decisions are not automatically executed without validation against these established policies, thereby maintaining control over all actions within HORSE. Internally, CAS utilizes Open Policy Agent, a policy engine that streamlines policy definition and authoring for regulators.

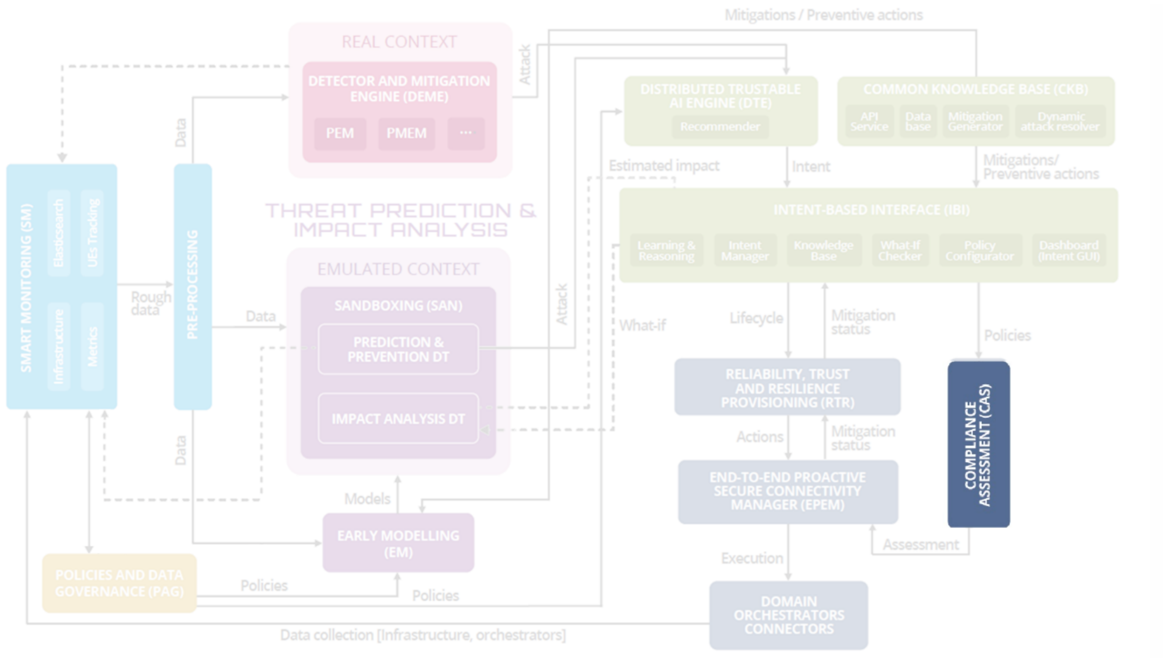
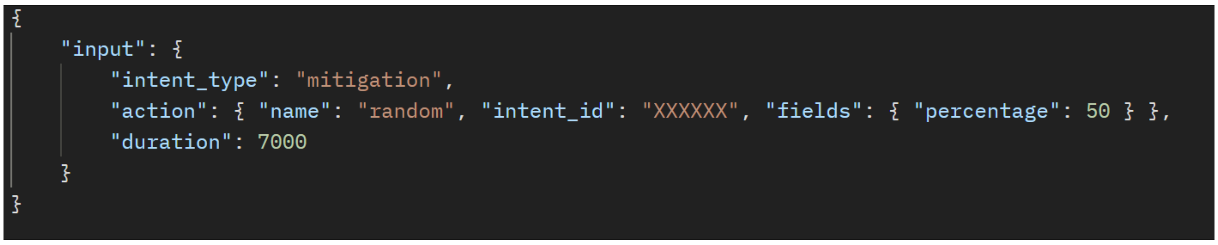

Mitigations / Preventive actions
REAL CONTEXT
DETECTOR AND MITIGATION ENGINE (DEME)
PEM
PMEM
...
Attack
DISTRIBUTED TRUSTABLE AI ENGINE (DTE)
Recommender
COMMON KNOWLEDGE BASE (CKB)
API Service
Data base
Mitigation Generator
Dynamic attack resolver
Estimated impact
Intent
Mitigations/ Preventive actions
INTENT-BASED INTERFACE (IBI)
Learning & Reasoning
Intent Manager
Knowledge Base
What-if Checker
Policy Configurator
Dashboard (Intent GUI)
THREAT PREDICTION & IMPACT ANALYSIS
EMULATED CONTEXT
SANDBOXING (SAN)
PREDICTION & PREVENTION DT
IMPACT ANALYSIS DT
SMART MONITORING (SM)
Elasticsearch
UEs Tracking
Infrastructure
Metrics
PRE-PROCESSING
Rough data
Data
What-if
Lifecycle
Mitigation status
Policies
RELIABILITY, TRUST AND RESILIENCE PROVISIONING (RTR)
COMPLIANCE ASSESSMENT (CAS)
Actions
END-TO-END PROACTIVE SECURE CONNECTIVITY MANAGER (EPEM)
Assessment
Execution
Models
EARLY MODELLING (EM)
POLICIES AND DATA GOVERNANCE (PAG)
DOMAIN ORCHESTRATORS CONNECTORS
Data collection (Infrastructure, orchestrators)

Figure 8.1 The CAS component within the HORSE architecture

8.1 Overview and Final Development Details

CAS is a "blackbox" component focused on validating all actions within HORSE. CAS is connected to HORSE through the IBI component in order to validate every action that HORSE is going to take. Within the HORSE workflow, CAS stands between the detection and/or prediction and the action that will take place since it's the component that will either give the go ahead or stop a possible action that will take place within HORSE.

By receiving the minimum information required for a specific mitigation action in a standardized format (JSON) that indicates the intended mitigation action and the fields as shown in Figure 8.2

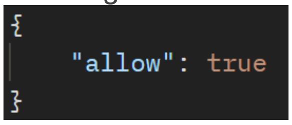


```
{
    "input": {
        "intent_type": "mitigation",
        "action": { "name": "random", "intent_id": "XXXXXX", "fields": { "percentage": 50 } },
        "duration": 7000
    }
}
```

Figure 8.2 Example input for CAS to validate showcasing the action and fields

CAS offers a RestAPI that takes a mitigation action in the aforementioned format and returns one of four possible outcomes:

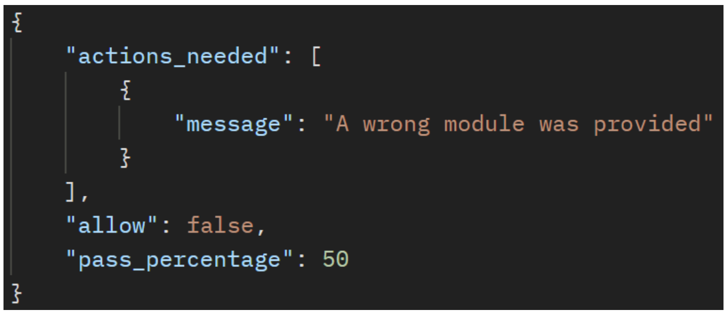
- **Full compliance:** The mitigation action fully meets security and privacy policies, returning a confirmation, as shown in Figure 8.3.



```
{
    "allow": true
}
```

Figure 8.3 Example result for full compliance

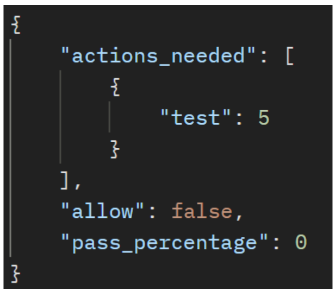
- **Partial compliance:** The action doesn't fully comply. CAS flags this, advises against proceeding as is, provides a compliance percentage, and suggests changes for improvement. This helps HORSE adapt to policy updates, as shown in Figure 8.4.



```
{
    "actions_needed": [
        {
            "message": "A wrong module was provided"
        }
    ],
    "allow": false,
    "pass_percentage": 50
}
```

Figure 8.4 Example result for partial compliance

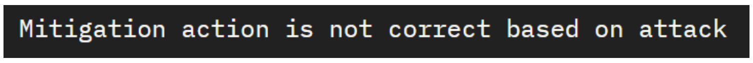
- **Zero Compliance:** The compliance level is too low to proceed, or CAS can't make a decision with the given info, as shown in Figure 8.5.



```
{
    "actions_needed": [
        {
            "test": 5
        }
    ],
    "allow": false,
    "pass_percentage": 0
}
```

Figure 8.5 Example result for zero compliance

- **Unable to make a decision:** CAS detects an issue but withholds details, for example, if a valid action seems suspicious given the current threat with the goal not to expose information on an attack.



```
Mitigation action is not correct based on attack
```

Figure 8.6 Example result for suspicious mitigation action

Internally, CAS validates each request through **a pipeline**.

1. First, it checks the request's context against HORSE's current state. If it seems illogical, it's rejected without extra details to avoid aiding potential attacks.
2. The action is checked against relevant policies. Policies are created and linked to actions.
3. CAS evaluates each policy for a given request and can provide helpful suggestions for achieving compliance, like removing an IP from a rate-limiting list if it's deemed that this violates a policy.
4. Finally, CAS logs data for auditing by authorized users, removes sensitive details, cleans the results, and responds quickly to avoid delays.

Integration with HORSE

CAS integrates with HORSE via components like IBI, which decides on the mitigation actions. IBI uses CAS's RestAPI to validate these actions and their parameters.

The external API for HORSE is simple, with a single validation endpoint for broader applicability to future mitigation actions:

- **[POST] /external-data:** Submits a query for validation and returns the result after pipeline processing.

A request specifies the mitigation to validate and the relevant fields. Internally, CAS has more detailed endpoints for better understanding its operations, but these aren't exposed to other components for security reasons.Internal API endpoints include:

- **[PUT] /policies:** Used to populate evaluation policies (internal use only as policies are predefined). It is usually used if we want to dynamically adjust (add/remove/modify) our policies while the CAS is running.
- **[GET] /possible-actions:** Returns a list of all the possible mitigation actions that CAS can validate (internal only to prevent revealing capabilities if compromised).
- **[GET] /current-attack:** CAS communicates with HORSE to know the current attack type that was detected or predicted.

System-wide awareness

CAS needs system-wide awareness to make informed decisions, knowing the system's status at any point. For HORSE, CAS should assess if a mitigation action is relevant based on detections from DEME or predictions from DTE.

“Blackbox” nature

CAS's "blackbox" nature helps integration, as users like IBI only interact via the API without needing details about the underlying policies. This improves scalability, allowing policy updates without changes elsewhere in HORSE.

Open Policy Agent

OPA is a general-purpose policy engine that unifies policy enforcement across the stack. OPA provides a high-level declarative language that lets you specify policy for a wide range of use cases.

Internally, for the assessment of policies CAS uses Open Policy Agent. This means that policies are being written in Rego language and in combination with the provided data OPA can make a binary decision on a query.

8.2 Changes and Debugging from IT-1 to IT-2

The transition from IT-1 to IT-2 involved several changes:

Based on the changes in the architecture of HORSE, it was decided that CAS needs to gets vital information about HORSE's state in order to prevent possible attacks like a Man in the Middle where a component of HORSE is compromised and the mitigation action doesn't correlate with the predicted / detected attack.

Also, CAS was adjusted from a typical allow or not allow binary system to a more dynamic and insightful compliant system of (compliant, partial compliant, not compliant, no-decision). To both provide better information for someone that utilizes CAS and support for policies that can be later introduced.

Introduction of OPA as an agent to validate the policies authored to validate that every possible action that HORSE will take is passing the regulations and policies that are being set for that specific action.

9 Discussion

9.1 Achievements of WP4

WP4 has successfully delivered a fully functional and secure orchestration layer for the HORSE platform, achieving the IT-2 maturity level across all its core components. Major accomplishments include:

- **Completion and Maturation of Core Components**: All WP4 components—Smart Monitoring (SM), Pre-processing Module, Common Knowledge Base (CKB), Reliability, Trust, and Resilience Framework (RTR), End-to-End Proactive Secure Connectivity Manager (ePEM), Domain Orchestrator Connectors (DOC), and Compliance Assessment Procedures (CAS)—have been developed, refined, and validated. Each component has evolved from initial prototypes to robust, secure, containerized services integrated with the platform.
- **Security-by-Design and Compliance**: WP4 prioritized privacy and security, with every module incorporating TLS-encrypted communication, access control (RBAC), anonymization (e.g., PCAP data in SM), OAuth2.0-based authentication, and GDPR-compliant data handling. These measures ensure that the orchestration platform operates securely in high-threat 5G/6G environments.
- **AI-Augmented Automation**: Several WP4 components now leverage AI, notably the CKB, which uses LLMs for attack-mitigation mapping, and the RTR, which supports natural language interpretation of mitigation intents. These advances improve decision automation, scalability, and usability of the platform.
- **Scalability and Interoperability**: Thanks to the modular and containerized nature of the components and adherence to RESTful APIs and standard data formats (e.g., JSON, YAML), we ensure interoperability with external systems and enables scalable deployment across diverse infrastructures.
- **Performance and Debugging Enhancements**: Each module underwent extensive debugging and optimization following IT-1 and WP5 preliminary integration feedback. Latency was reduced, retry mechanisms were added, and data mismatches were resolved—ensuring that the orchestration layer can support real-time threat mitigation scenarios.

Collectively, WP4 has laid the technical foundation for a secure, AI-assisted orchestration system that is human-centric, modular, and prepared for deployment in operational environments.

9.2 Integration with WP5

Throughout the IT-2 development cycle, WP4 maintained strong collaborative ties with WP5 to ensure smooth integration into the broader HORSE platform. The following integration outcomes were achieved:

- **Feedback-Driven Refinement**: WP4 components were iteratively improved based on WP5's preliminary integration and testing feedback. This included enhancements in interface robustness, error handling, and standardization of data exchange formats.
- **Interface Compatibility**: All WP4 components exposed RESTful APIs in accordance with WP5 integration guidelines, enabling seamless connectivity with upstream (e.g., IBI, DEME) and downstream (e.g., ePEM, testbed) modules.

- **Data Pipeline Coherence**: The Pre-processing Module and Smart Monitoring were aligned to ensure that data collected in WP5 scenarios could be harmonized and consumed by analytics components without delays or inconsistencies. Dual data output support enabled both real-time and historical data access for validation tasks.
- **Mitigation Workflow Synchronization**: The RTR and ePEM modules were successfully aligned with WP5's orchestration and enforcement flow. RTR now supports asynchronous status tracking via callbacks from ePEM, which proved essential during WP5 tests of end-to-end threat mitigation cycles.
- **Validation Readiness**: All WP4 modules were containerized, thoroughly documented, and delivered with deployment and maintenance guidelines, easing their integration in WP5's full platform validation tasks (D5.2). Their current readiness positions WP4 components to be tested under realistic use-case scenarios in the HORSE testbed environments.

In summary, WP4 achieved full technical readiness for final integration, providing WP5 with stable, mature components and ensuring the orchestration platform meets the functional and non-functional requirements of the HORSE project.

References

[1] E. Rodriguez *et al.*, 'A Security Services Management Architecture Toward Resilient 6G Wireless and Computing Ecosystems', *IEEE Access*, vol. 12, pp. 98046–98058, 2024, doi: 10.1109/ACCESS.2024.3427661.

[2] X. Zhang *et al.*, 'SRv6-INT Enabled Network Monitoring and Measurement: Toward High-Yield Network Observability for Digital Twin', in *Proceedings of the 3rd International Conference on Machine Learning, Cloud Computing and Intelligent Mining (MLCCIM2024)*, vol. 1328, F. Sun, H. Wang, H. Long, Y. Wei, and H. Yu, Eds., in Lecture Notes in Electrical Engineering, vol. 1328. , Singapore: Springer Nature Singapore, 2025, pp. 113–126. doi: 10.1007/978-981-96-1698-5_12.

[3] J. M. Parra-Ullauri, X. Zhang, A. Bravalheri, Y. Wu, R. Nejabati, and D. Simeonidou, 'Federated Analytics for 6G Networks: Applications, Challenges, and Opportunities', Jan. 09, 2024, *arXiv*: arXiv:2401.03878. doi: 10.48550/arXiv.2401.03878.

[4] X. Chen, W. Feng, N. Ge, and Y. Zhang, 'Zero Trust Architecture for 6G Security', Mar. 16, 2022, *arXiv*: arXiv:2203.07716. doi: 10.48550/arXiv.2203.07716.

[5] *Bulk index or delete documents*. [Online]. Available: https://www.elastic.co/docs/api/doc/elasticsearch/group/endpoint-document

[6] M. Penelova, 'Access Control Models', *Cybernetics and Information Technologies*, vol. 21, no. 4, pp. 77–104, Dec. 2021, doi: 10.2478/cait-2021-0044.

[7] C. Fan, M. Chen, X. Wang, J. Wang, and B. Huang, 'A Review on Data Preprocessing Techniques Toward Efficient and Reliable Knowledge Discovery From Building Operational Data', *Front. Energy Res.*, vol. 9, p. 652801, Mar. 2021, doi: 10.3389/fenrg.2021.652801.

[8] L. Zheng *et al.*, 'Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena', Dec. 24, 2023, *arXiv*: arXiv:2306.05685. doi: 10.48550/arXiv.2306.05685.

[9] Adeola Adewa, Vincent Anyah, Omoniyi David Olufemi, Adedeji Ojo Oladejo, and Toluwanimi Olaifa, 'The impact of intent-based networking on network configuration management and security', *Global J. Eng. Technol. Adv.*, vol. 22, no. 1, pp. 063–068, Jan. 2025, doi: 10.30574/gjeta.2025.22.1.0012.

[10] M. Danousis, K. Kaltakis, A. Dimos, C. Skianis, E. Kafetzakis, and I. Giannoulakis, 'Optimizing Network Cybersecurity: AI-Powered NLP for Natural Language Command Interpretation', in *2024 IEEE 29th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Athens, Greece: IEEE, Oct. 2024, pp. 1–7. doi: 10.1109/CAMAD62243.2024.10942816.

[11] A. Carrega and R. Rabbani, 'ePEM: An End-to-End Proactive Secure Connectivity Manager for 6G Orchestrator Solutions', in *2024 IEEE 29th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Athens, Greece: IEEE, Oct. 2024, pp. 1–7. doi: 10.1109/CAMAD62243.2024.10942805.

[12] G. Gatti, J. M. Jorquera Valero, M. Gil Pérez, and C. Basile, 'Holistic Cyber Risk Assessment in the Cloud Continuum: A Multi-Layer, Multi-Domain Approach', 2025. doi: 10.2139/ssrn.5223333.

[13] M. F. Safitra, M. Lubis, and H. Fakhrurroja, 'Counterattacking Cyber Threats: A Framework for the Future of Cybersecurity', *Sustainability*, vol. 15, no. 18, p. 13369, Sep. 2023, doi: 10.3390/su151813369.