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

D5.2 First HORSE Release: HORSE IT-1 version

Revision: v.1.0

Work package	WP 5
Task	Task 5.2
Due date	30/06/2024
Submission date	30/06/2024
Deliverable lead	TUBS
Version	1.0
Authors	Iulisloi Zacarias (TUBS); Admela Jukan (TUBS); Chukwuemeka Muonagor (TUBS); Paulo Paixão (EFACEC); Pedro Elisio (EFACEC); Eva Rodriguez Luna (UPC); Fabrizio Granelli (CNIT); Alice Piemontti (MAR); Jose Manuel Manjón (TID); Orazio Toscano (ETI); Eduardo Canovas Martinez (UMU); Stefanos Venios (S5); Manuel Angel Jiménez Quesada (ATOS); Josep Martrat (ATOS); Rodrigo Diaz Rodriguez (ATOS); Alexandros Dimos (8BELLS); Sofia Giannakidou (STS); Panagiotis Trakadas (NKUA); Nikolaos Nomikos (NKUA); Panagiotis Gkonis (NKUA)
Reviewers	Fabrizio Granelli (CNIT), Charalampos Skianis (8BELLS)

Abstract	This document presents the initial release of the HORSE cybersecurity platform, designed to enhance the security of 5G/6G networks. It provides a detailed account of the development, integration, and testing of the modules composing the first release of the HORSE platform. The document outlines the methodologies employed in system integration tests, showcases practical applications through demonstrators, and presents the components and interfaces that the demonstrator validates. Additionally, each demonstrator description includes a matrix assessing the interactions and status of the integration tests. The demonstrators were carefully designed to validate the designed components via functional tests and ensure the components' integration was handled correctly. Additionally, a glance at Key Performance Indicators (KPIs) that will be used in the validation phase of the next HORSE iteration (IT-2) is presented.
Keywords	Functional Validation; Integration; Cybersecurity; Mobile Networks

DOCUMENT REVISION HISTORY

Version	Date	Description of change	List of contributor(s)
V0.1	20/09/2022	1st version of the template for comments	Miguel Alarcón (Martel)
V0.1.1	06/05/2024	Preliminary version of the Table of Contents	Iulisloi Zacarias (TUBS)
V0.1.2	09/05/2024	Updates in the Table of Contents	Paulo Paixão (EFACEC)
V0.2	16/05/2024	Updates in the Table of Contents and new sections for version control of the modules	Iulisloi Zacarias (TUBS)
V0.3.1	21/05/2024	Discussions about the structure of the document.	ALL
V0.3.2	24/05/2024	Updates in the Table of Contents and reorganization of contents following the discussions in the General Assembly on 21/05/2024.	Iulisloi Zacarias (TUBS)
V0.4.0	03/06/2024	Adding Integration Matrix and components integration	Eva Rodriguez Luna (UPC)
V0.4.1	12/06/2024	Adding Prevention DT description	Fabrizio Granelli (CNIT)
V0.4.2	14/06/2024	Contribution to Section 2	ALL
V0.4.3	14/06/2024	Contributions to Section 4	ETI, TID, TUBS, CNIT, ATOS, NKUA
V0.4.4	18/06/2024	Update Annex A with KPI definitions collected from partners in particular from EFACEC and Ericsson	Paulo Paixão, Pedro Elísio; Orazio Toscano
V0.4.5	19/06/2024	Contribution to Section 3	Eduardo Canovas Martinez (UMU)
V0.4.6	19/06/2024	Updates in the validation matrix and system integration test. Compliance matrix.	Eva Rodriguez Luna (UPC) and Josep Martrat (ATOS)
V0.5	20/06/2024	Initial formatting and harmonization of the document	Iulisloi Zacarias (TUBS)
V0.6	23/06/2024	Version ready for internal review	Iulisloi Zacarias (TUBS)

V0.7	24/06/2024	Internal review process	Charalampos Skianis (8BELLS), Fabrizio Granelli (CNIT)
V0.8	28/06/2024	Version ready for quality assessment	Iulisloi Zacarias (TUBS)
V1.0	28/06/2024	Quality assessment and final version to be submitted.	Fabrizio Granelli (CNIT)

Disclaimer

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the other granting authorities. Neither the European Union nor the granting authority can be held responsible for them.

Copyright notice

Project co-funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	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

This document presents a comprehensive account of the initial release of the HORSE cybersecurity platform, a solution designed to enhance the security of 5G/6G networks. It is supposed to accompany the software release of components in the HORSE project and guide the reader through the code available on the GitHub platform. Additionally, it details the development, integration, and testing of the modules composing the HORSE platform's first release. This document outlines the methodologies employed in system integration tests, showcases practical applications through demonstrators, and provides a validation matrix to assess the interactions and performance of the integrated components. The integration matrix shows the current state of the integration. Additionally, it demonstrates the ability of HORSE partners to link the developed components in line with modular, yet highly integrated architecture foreseen in the beyond 5G landscape. The document follows by proposing five demonstrators that aim to validate the components against the proposed workflow for preventing, detecting, and mitigating network threats. For each demonstrator, a set of integration tests is described together with their expected input/output to ensure straightforward end-to-end integrations of modules. Finally, Annex A provides a first thought on Key Performance Indicators (KPIs) that will be used by the HORSE project to evaluate the platform's performance in future iterations.

Table of contents

List of figures

List of tables

Abbreviations

5G	Fifth Generation of Mobile Networks
6G	Sixth Generation of Mobile Networks
AI	Artificial Intelligence
API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
cKB	Common Knowledge Base
CN	Core Network
DDoS	Distributed Denial of Service
DEME	Detector and Mitigation Engine
DN	Data Network
DNS	Domain Name System
DRL	Deep Reinforcement Learning
DT	Digital Twin
DTE	Distributed Trustable AI Engine (DTE).
EM	Early Modeling
ePEM	End-to-End Secure Connectivity Manager
FAR	Forwarding Action Rule
GHCR	GitHub Container Registry
gNB	Next Generation Node B
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IA-DT	Impact Analysis Digital Twin
IBI	Intent-based Interface
IP	Internet Protocol
IT-X	Project Iteration X
JSON	JavaScript Object Notation
ML	Machine Learning
NDT	Network Digital Twin
NF	Network Function
NTP	Network Time Protocol

P&P	Prediction and Prevention
PAG	Policies and Data Governance
PDU	Protocol Data Unit
PFCP	Packet Forwarding Control Protocol
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
RTR	Reliability, Trust and Resilience
SEID	Session Endpoint Identifier
SM	Smart Monitoring
SMF	Session Management Function
TEID	Tunnel Endpoint Identifier
TTP	Tactics, Techniques, and Procedures
TX.Y	Task X.Y
UE	User Equipment
UI	User Interface
UPF	User Plane Function
URL	Uniform Resource Locator
VM	Virtual Machine
WebUI	Web-based User Interface
WPX	Work Package X
XML	Extensible Markup Language

1 Introduction

The HORSE project represents a concerted effort to advance the cybersecurity framework within the evolving landscape of 5G/6G networks. As the first major release, this document encapsulates the extensive research, collaborative development, and meticulous integration efforts that the consortium has undertaken. The HORSE platform is designed to offer a comprehensive, scalable, and robust solution to address the multifaceted challenges of cybersecurity in next-generation networks. This document not only details the initial release but also sets a foundation for ongoing and future enhancements.

All the software and tools are available in a GitHub repository created specifically for HORSE and located at https://github.com/HORSE-EU-Project.

1.1 Purpose of the document

The primary aim of this deliverable is to be an accompanying document that will guide the readers through the components composing the first release for the HORSE platform. It is designed to guide stakeholders through the intricacies of the HORSE modules, offering detailed descriptions of their functionalities, installation processes, and operational parameters. The document presents the resulting efforts of partners in transitioning from conceptual development to practical implementation of the platform. It is worth mentioning that during the implementation phase, some additional requirements arose, culminating in small changes in the initial architecture, such as the addition of the Common Knowledge Base.

Furthermore, it acts as a reference point for the project's progress, showcasing the consortium's achievements regarding the integration of the developed components. It sheds light on the integration test procedures designed and implemented following the guidelines presented in this deliverable to guarantee a smooth end-to-end integration of the HORSE platform. The document demonstrates the functional validation of HORSE components by proposing a set of demonstrators and the integration endpoints between components with relevant tests that will guarantee a smooth end-to-end integration of the developed software.

The document describes the five demonstrators proposed to showcase the HORSE platform in action. Each demonstrator covers distinct parts of the HORSE workflow, aiming to prevent, detect, and mitigate mobile network threats and attacks. Although a unique demonstrator does not cover the end-to-end workflow, the complete set of demonstrators proposed in this document can ultimately validate the entire HORSE workflow.

Finally, Annex A provides a glimpse into the Key Performance Indicators (KPIs) that will be used to evaluate the platform's performance in future iterations.

1.2 Relation to other HORSE deliverables and project work

This document reflects the efforts employed in designing and implementing the components in all Work Packages (WP) of the HORSE Project. Specifically, it presents the software developed in WP3, WP4, and WP5. In the scope of WP5, it presents the development of the Intent-based Interface described in T5.1. Task T5.2 deals with the integration of components developed in WP3 and WP4. It also presents the outcomes of tasks T5.3 and T5.4 regarding demonstration strategy and continuous validation and scalability tests, respectively. It builds on the foundational research already presented in D2.1 HORSE Landscape: Technologies, State of the Art, AI Policies and Requirements [1], and it follows the architecture proposed in D2.2 HORSE Architectural Design (IT-1) [2].

1.3 Structure of the document

The document is structured as follows:

- Section 2: presents all the components developed so far in the HORSE Project's scope, consolidating the work carried out in WP3, WP4, and WP5. It contains a short description of the components' functionalities, a link to the source code available on GitHub, dependencies, and instructions for installing and running the modules.
- Section 3 describes the continuous integration methodology adopted in HORSE, the integration workflow, and the required artifacts for module integration tests. It also presents an example of an integration test to be adopted by all components.
- Section 4 presents the validation strategy adopted in the HORSE intermediate release (IT-1), the interactions between components, and the demonstrators proposed for the functional validation of the HORSE platform's components.
- Section 5 concludes the document and draws the next steps in the HORSE development. It highlights the complexity and advantages of the adopted integration strategy.
- Annex A presents the first thoughts and briefly describes the KPIs that will be used to analyze and map HORSE platform features on the scenarios that will be used for validation and demonstration.
- Annex B presents a detailed guide on how to setup secrets and specific configurations on GitHub to enable the building and testing of the components.

2 List of Modules Included in HORSE Release 1

This section provides an overview of the modules included in the first release of the HORSE platform, describing each module, its implemented functionalities, dependencies, and installation instructions. It also points the reader to the software's GitHub repository. The implementation of the modules follows the architecture proposed in the work of WP2 and reported in D2.2 HORSE Architectural Design (IT-1) [2]. The overview of the main modules and the interfaces and connections between modules is presented in Figure 1 to help the reader navigate through the list of modules provided in this section.

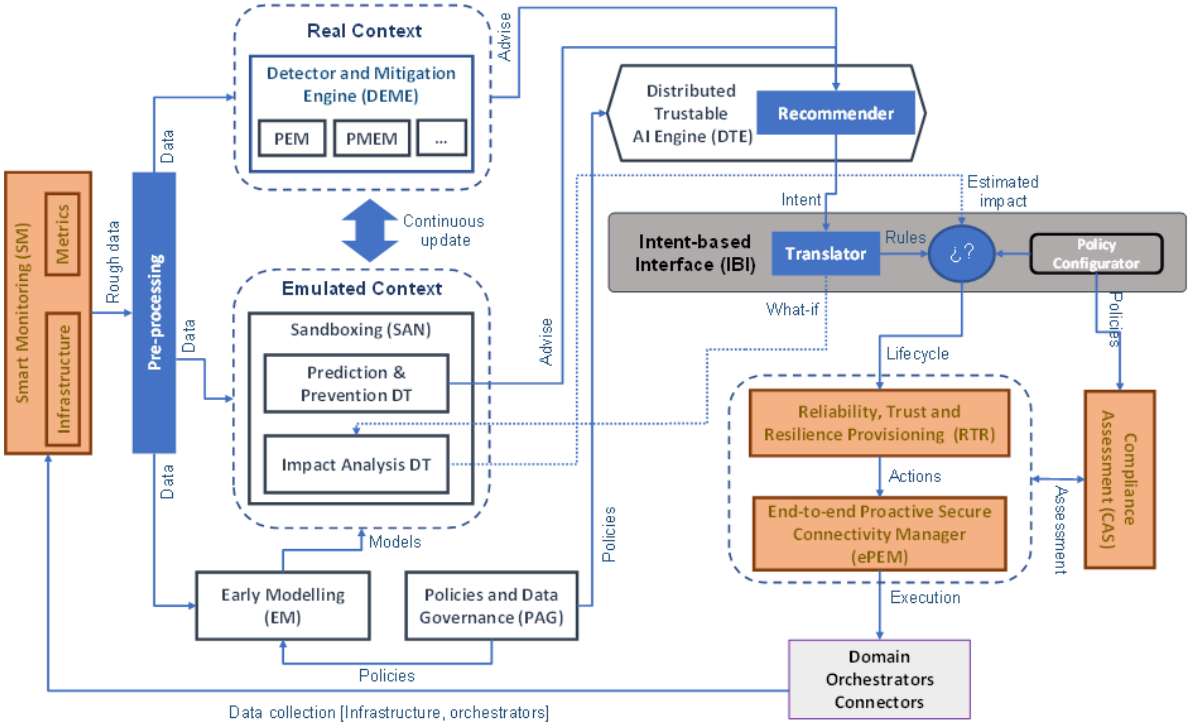


Figure 1: The HORSE architecture

2.1 Smart Monitoring

This module provides the following functionalities:

- Data collection from Elastic Beats (e.g. network traffic, system metrics) and persistence in Elasticsearch
- Data querying via Elasticsearch
- Data visualization via Kibana

Version: v 1.0

Link to repository:

https://github.com/HORSE-EU-Project/Sphynx-Data-Collection

Installation Instructions:

https://github.com/HORSE-EU-Project/Sphynx-Data-Collection/blob/main/README.md

List of dependencies and third-party software Included in the current release

- elasticsearch:7.17.13

- kibana:7.17.18
- Docker, version 24.0

2.2 Pre-Processing

The pre-processing module acts as a middleware between the Smart Monitoring (SM) and several HORSE components, which require continuous flow of network traffic to operate. These are Detector and Mitigation Engine (DEME), Network Digital Twin (NDT) and Early Modeling (EM). The main task of the Pre-Processing is to gather collected network traffic from the SM's Elasticsearch instance, process the data and filter the useful information and finally forward these pieces of information to the correct HORSE component.

Version: v0.1

Link to repository:

https://github.com/HORSE-EU-Project/Pre-Processing

Installation Instructions:

https://github.com/HORSE-EU-Project/Pre-Processing/blob/main/README.md

List of dependencies and third-party software Included in the current release

- Ubuntu 22.04.3
- Docker, version 26.1.1
- Docker Compose version v2.27.0
- Git version 2.34.1

2.3 Prediction and Prevention Digital Twin

The Prediction and Prevention (P&P) Digital Twin (DT) is one of the components of the Sandboxing module. The module is aimed at building an emulated replica of the actual 5G/6G infrastructure in order to perform predictions about potential events that might happen in the future and enable prevention strategies. This module receives information from the Early Modelling (EM) about the models of the attacks and the mitigations of each of them, and it produces a list of potential threats and mitigation actions to submit to the Distributed Trustable AI Engine (DTE).

The module is provided in the form of a Virtual Machine integrating the network emulator environment and associated software to build and run the Digital Twin. This version includes the input and output REST interfaces and implements simple algorithms to perform prediction and detection of attacks. Next releases will focus on improving the performance of such algorithm in order to provide reliable input to the DTE. For the purpose of testing, a web interface is provided in order to check the received input files and produced outputs, and to test the actions and predictions of the P&P DT.

Version: v0.1

Link to repository:

https://github.com/HORSE-EU-Project/PredictionDigitalTwin

Installation Instructions:

https://github.com/HORSE-EU-Project/PredictionDigitalTwin/blob/main/README.md

List of dependencies and third-party software Included in the current release

- Virtualbox 7.0 (VM hosting environment)
- Vagrant v2.4.1 (VM provisioning)
- Ubuntu 20.04
- Comnetsemu: v0.3.0
- UERANSIM: v3.2.6
- Open5gs: v2.4.2
- Docker v20.10 or newer
- Python 3.10 or newer
- FASTApi

2.4 Impact Analysis Digital Twin

The Impact Analysis Digital Twin is one of the components of the Sandboxing module. This module is in charge of building an emulated environment of the physical network and will help the Intent-Based Interface (IBI) to take the decision about the proper mitigation for certain attack. For this, this DT will work with what-if actions, where the IBI will send requests about some countermeasures for the attacks in certain points of the network and the Impact Analysis will respond with the concrete impact that they have in the emulated environment (e.g., bandwidth, latency, etc.). Also, this module receives information from the EM about the models of the attacks and the mitigations of each of them.

Version: v0.1

Link to repository:

https://github.com/HORSE-EU-Project/Impact-Analysis-Digital-Twin

Installation Instructions:

https://github.com/HORSE-EU-Project/Impact-Analysis-Digital-Twin/blob/main/README.md

List of dependencies and third-party software Included in the current release

- KIND (Kubernetes cluster)
- Golang, version 1.21 or newer
- Kubectl, version 1.27.3 or newer
- Make
- Docker, version 20.10.16 or newer
- Kubernetes Network Emulator (KNE)
- Open5GS, version 2.7.0
- UERANSIM, version 3.2.6

2.5 Early Modelling

Version 0.1 of the Early Modeling (EM) software module contains a taxonomy responsible for characterizing threats. The EM module analyzes the information on threats by characterizing assets, threat actors, Tactics, Techniques, and Procedures (TTP), cyber security

vulnerabilities, threats, and impact assessment of the attack and mitigation strategy. Version 0.1 is demonstrated with the help of key inputs and output. The input encompasses the threat information, the threat actor, assets, TTP, and security vulnerabilities associated with the respective attack. The mitigation actions are provided by the knowledge base and the outcome is produced in the form of an XML model. The sandboxing module will use the XML model produced by the EM software module for the prediction of threats and the preliminary assessment of the attack impact.

Version: v0.1

Link to repository:

https://github.com/HORSE-EU-Project/earlymodeling_prototype

Installation Instructions:

https://github.com/HORSE-EU-Project/earlymodeling_prototype/blob/main/README.md

List of dependencies and third-party software Included in the current release

- Docker version 24.0.7

2.6 Policies and Data Governance

Version 0.1 of the Policies and Data Governance (PAG) component includes: (a) the access policy User Interface (UI) and the pertinent mechanism to enforce access policies per user and per component, (b) the data retention policy UI and the pertinent mechanism to enforce data retention policies per dataset, (c) the anonymization UI without the mechanism to implement anonymization on datasets, (d) storage for the metadata of datasets. Version 0.1 of the PAG, for demonstration reasons, also includes a mock sign-up and login (without email authentication) and a minIO instance as the demo data store (populated with 2 datasets). The PAG has been built in a way to allow to connect to more than one, new data stores in the future.

Version: v0.1

Link to repository:

https://github.com/HORSE-EU-Project/PAG

Installation Instructions:

https://github.com/HORSE-EU-Project/PAG/blob/main/README.md

List of dependencies and third-party software Included in the current release:

- Docker, version 24.0.7
- Postgres, version 15
- MinIO, version : RELEASE.2024-05-10T01-41-38Z
- OpenFGA, version v1.5.3

2.7 Detector and Mitigation Engine

The Detector and Mitigation Engine (DEME) works in the “real context” providing threat detection in the real infrastructure. Having historical network data, network topology configuration and the name of the network features monitored for each network instance of the topology, then

- DEME can receive, via Application Programming Interface (API), from Pre-processing, in real-time a snapshot of the network features monitored for each network instance of the topology and it provides in output, for its second Machine Learning (ML) step, the real-time comparisons of the incoming network data (that are periodically collected) with the predicted baselines.
- Using a (Supervised) ML tool trained on the variations of the N variables provided by the previous steps, DEME promptly detects any related attack with a 360 degrees perspective (no siloization).
- DEME returns the accuracy of an attack for each network instance via API to DTE

Version: v 1.0

Link to repository:

https://github.com/HORSE-EU-Project/DEME-apis

List of dependencies and third-party software Included in the current release

- Docker, version 20.10.16 or newer

2.8 Common Knowledge Database

The Common Knowledge Base (cKB) is a component of the HORSE project, designed as a centralized database to store and provide essential information on attack mitigations. The uniqueness of this database lies in its enhancement with AI during the data population phase.

The cKB is composed by:

- Database: stores detailed information on various attacks and their corresponding mitigation actions.
- Web Server: exposes APIs to retrieve essential information from the database without direct access.

Here are listed the current functionalities of the cKB:

- The web server's API allows retrieval of a list of mitigations for a specified attack.
- Mitigations are provided in JSON format as an ordered list. Each mitigation includes an execution priority to effectively counteract the specified attack, along with a description of its impact and the rationale for its assigned priority.
- Artificial Intelligence is utilized to assign priorities to each mitigation related to specific attacks. Generative AI will also be used to create detailed descriptions for each mitigation, explaining why each priority was assigned.

Version: v 1.0

Link to repository:

https://github.com/HORSE-EU-Project/KnowledgeBase-proto

Installation Instructions:

https://github.com/HORSE-EU-Project/KnowledgeBase-proto/blob/main/README.md

List of dependencies and third-party software Included in the current release

- Docker, version 26.0.7

2.9 Distributed Trustable AI Engine

The Distributed and Trustable Engine (DTE) receives inputs from the DEME in the form of advises, for different types of attacks in the network. DTE maps the outcome of ML detection and classification to appropriate recommendations that are expressed through intents. These intents are later forwarded to the IBI and can be expressed in two forms: a) mitigative, which describes the objectives the system should accomplish in the short term (e.g., within a few seconds) so that the impact of ongoing security incidents is reduced and b) preventive, which describes the objectives the system should accomplish in the longer term so that is better protected against future potential threats.

Version: v 1.0

Link to repository:

https://github.com/HORSE-EU-Project/NKUA-DTE

Installation Instructions:

https://github.com/HORSE-EU-Project/NKUA-DTE/blob/main/README.md

List of dependencies and third-party software Included in the current release

- Docker version 25.0.3
- FastAPI version

2.10 Intent-based Interface

The Intent-based Interface (IBI) is a module that can receive intents regarding the desired state of the network and translate the intents to policies that can be applied in the infrastructure. In the HORSE implementation, the IBI can receive security intents from the DTE or from the network administrators through the Intent Graphical User Interface (GUI). The security intents could be mitigation or prevention intents, which are for threat mitigation and prevention, respectively. Within the IBI, the intents are processed and matched with the policies that are applied in the network to fulfill the active intents. The policies are extracted from the cKB of the HORSE project. A network administrator can also input Quality of Service (QoS) intents through the Intent GUI, which the IBI is expected to comply with when deciding which policies to select for specific security intents. Internally to the IBI, the Policy Configurator resolves conflicts between two policies and configures its conflict-free policies into workflows that are sent to the Reliability, Trust and Resilience (RTR) component. These workflows contain the desired mitigation or prevention actions to be implemented in the network to neutralize or prevent attacks. The management of active intents is possible through the Intent GUI or RESTful APIs.

Version: v 0.5a

Link to repository:

https://github.com/HORSE-EU-Project/IBI

Installation Instructions:

https://github.com/HORSE-EU-Project/IBI/blob/main/README.md

List of dependencies and third-party software Included in the current release

- Elastic Search, version 8.13.2
- Docker, version 26.1.2

- Docker Compose, version 2.27.0
- Python, version 3.12.3
- Flask, version 3.0.1
- FastAPI, version 0.111.0
- Pandas, version 2.2.2

2.11 Reliability, Trust and Resilience

The Reliability, Trust and Resilience (RTR) is part of the mitigation enforcement workflow. It is located between the IBI and the ePEM and is essentially responsible for interpreting a high-level sentence, describing the mitigation actions that need to be applied to certain areas of the topology. Based on this sentence, the RTR can create applicable rules in the form of Ansible playbooks. These playbooks will be forwarded to the End-to-End Secure Connectivity Manager (ePEM) and will be enforced on the requested part of the topology.

Version: v 0.1

Link to repository:

https://github.com/HORSE-EU-Project/RTR

Installation Instructions:

https://github.com/HORSE-EU-Project/RTR/blob/main/README.md

List of dependencies and third-party software Included in the current release

- Ubuntu 22.04.3
- Docker, version 26.1.1
- Docker Compose version v2.27.0
- git version 2.34.1

2.12 End-to-End Secure Connectivity Manager

The End-to-End Secure Connectivity Manager (ePEM) is a network-oriented meta-orchestrator, specifically designed for zeroOps and continuous automation. It can create, deploy, and manage the lifecycle of different network ecosystems by consistently coordinating multiple artifacts at any programmability level (from physical devices to cloud-native microservices).

The ePEM plays a pivotal role in the HORSE security infrastructure. HORSE represents a cutting-edge security infrastructure designed to safeguard complex, distributed, and heterogeneous systems. In this intricate environment, the ePEM serves as a central architectural element, orchestrating actions and providing observability over the various components that constitute the end-to-end services secured within the HORSE security perimeter.

Key Features

- Deploys blueprints that build 5G, kubernetes (k8s), and VyOS2 services
- Manages the lifecycle of blueprints
- Manages k8s cluster and Machines/VMs

Version: v1.0.1

Link to repository:

https://github.com/HORSE-EU-Project/ePEM

Installation Instructions:

https://github.com/HORSE-EU-Project/ePEM/blob/HORSE/README.md

https://nfvcl-ng.readthedocs.io/en/latest/index.html#getting-started

Prerequisites:

- An OpenStack instance (you can use all-in-one installation here).
- An Ubuntu (22.04 LTS) instance where the NFVCL will be installed and run.
- Having OSM 14 running on a reachable machine, in the following installation procedure, all the services will be installed on the same Ubuntu instance.
- Python 3.11 (Installation performed in setup.sh).
- If the NFVCL and OSM are running on the same machine:
  - **RECOMMENDED**: 4 CPUs, 16 GB RAM, 80GB disk and a single interface with Internet access.
  - **MINIMUM**: 2 CPUs, 8 GB RAM, 50GB disk and a single interface with Internet access.

2.13 Domain Orchestrator Connectors

The Domain Orchestrator Connectors (DOC) is the last component in HORSE Context and its main purpose is to manage multicluster environments. To achieve its goal, in its version 0.1, the DOC adapts different mitigation action in the HORSE context to a Liqo orchestrator. Each action is mapped into a particular way to enforce it in the infrastructure. In that way DOC achieves that Topology/infrastructure could be deployed in different geographical places.

Version: v0.1

Link to repository:

https://github.com/HORSE-EU-Project/Domain-Orchestrator-Connectors

Installation Instructions:

https://github.com/HORSE-EU-Project/Domain-Orchestrator-Connectors/blob/main/README.md

List of dependencies and third-party software Included in the current release

- Docker, version 26.1.2
- Golang 1.22.2
  - github.com/gin-gonic/gin v1.10.0
  - github.com/spf13/viper v1.19.0
  - github.com/stretchr/testify v1.9.0

3 System Integration Tests

This section focuses on the system integration tests conducted for the HORSE platform's first release. These tests are essential to ensure the various modules work seamlessly together and perform as expected in a real-world environment. This section outlines the methodologies employed and the specific tests conducted. It presents an example of integration tests that are performed between two HORSE components.

3.1 Continuous Integration and Methodology

The continuous integration (CI) processes within the HORSE project are enhanced through a robust and systematic approach. By leveraging industry best practices, the methodology centers around using GitHub Actions to automate integration and testing workflows, ensuring seamless validation and integration of all project components.

3.1.1 Integration testing

This type of testing evaluates how various software application modules interact and operate cohesively. The system is divided into components known as modules or units, each responsible for a specific task. The real challenge comes when combining these components to develop the entire software system.

At this stage, the connections between each module are carefully examined to uncover any potential issues arising from a single unit. Once the testing is completed, end-to-end tests are conducted to evaluate the application's functionality from start to finish.

3.1.2 Unit testing vs Integration testing

Unit tests and integration tests have distinct purposes and are conducted at different stages of the development lifecycle. Table 1 compares the differences between unit tests and integration tests:

Aspect	Unit Testing	Integration Testing
Scope	Focuses on individual units or specific modules.	Evaluates the interaction of integrated modules.
Objective	Verify that each element works independently.	Verify that all connected pieces work correctly together.
Interoperability	External dependencies are kept separate from the tests.	Requires integration with real-world modules.
Speed	Since it focuses on small units, execution is faster.	Execution time is slightly longer due to the testing of various modules.

Coverage	Provides extensive code coverage by thoroughly inspecting modules.	Ensures that the modules work correctly as part of the overall system.
Testing Workflow	Developers typically perform unit tests before integration tests.	Usually conducted during the software integration phase.

Table 1: Integration approaches

3.1.3 Integration Tests Approaches

- **Top-Down Approach (incremental)**: The top-down approach begins integration from the top-level modules and progresses downward. Stubs are used to simulate lower-level modules during early testing. While this approach validates high-level functionality early and can identify major design flaws, it delays testing of lower-level modules and requires the time-consuming development of stubs. Consequently, integration issues in lower-level modules might only surface late in the process.
- **Bottom-Up Approach (incremental)**: Conversely, the bottom-up approach starts from the lowest-level modules and moves upward, using drivers to simulate higher-level modules. This method allows for early validation of lower-level functionality and makes it easier to identify and fix defects in these modules. However, high-level functionality is tested late, and the development of drivers can also be time-consuming, leading to potential delays in discovering integration issues in higher-level modules.
- **Big-Bang Approach (non-incremental)**: The Big-Bang approach involves integrating all modules simultaneously and testing the complete system. This approach offers comprehensive testing of the entire system, faster integration, and immediate identification of interaction issues. It simplifies management by eliminating the need for stubs and drivers. However, debugging can be challenging due to the simultaneous integration of many components, and identifying the source of defects can be complex. This method requires all modules to be ready for integration at the same time.

3.1.4 Selected Integration Test Approach for HORSE Project

The Big-Bang approach for integration testing is employed to achieve efficient and reliable integration. This method integrates all components simultaneously and then tests them as a complete system. This approach is chosen over the bottom-up or top-down methodologies due to its distinct advantages:

Why **Big-Bang** is Better for the HORSE Project

1. **Holistic Validation**: The HORSE project involves integrating multiple complex components, such as ePEM and RTR, that must work seamlessly together. The Big-Bang approach ensures a holistic validation of the entire system, confirming that all components interact correctly from the outset.

2. **Faster Integration**: Given the project's scope and the need for rapid development cycles, the Big-Bang approach allows for quicker integration compared to phased methods like top-down and bottom-up. This is crucial for meeting tight project deadlines and delivering incremental releases efficiently.

3. **Immediate Feedback**: By integrating all components simultaneously, the Big-Bang approach provides immediate feedback on the overall system's functionality. This allows for quick identification and resolution of integration issues that could affect multiple components.

4. **Resource Efficiency**: The Big-Bang approach reduces the overhead associated with creating and maintaining stubs and drivers required in other methods. This frees up resources for other critical development activities, enhancing overall efficiency.

5. **Simplified Management**: Managing the integration process becomes simpler with the Big-Bang approach, as there is a single, comprehensive integration phase. This reduces the complexity of coordinating multiple integration phases and ensures a streamlined workflow.

6. **Suitable for Tested Modules**: The Big-Bang approach is particularly effective when individual modules are relatively mature and well-tested in isolation. The HORSE project's components, such as ePEM and RTR, have undergone significant unit testing, making them suitable for this approach.

3.2 Horse Integration Workflow

The diagram in Figure 2 illustrates an integration workflow for the HORSE Project, using technologies such as GitHub Actions to automate and streamline the continuous integration process. The workflow showcases how modules can be integrated using the aforementioned Big-Bang approach, where all modules are integrated simultaneously and tested as a complete system. Although this example focuses on two modules (ePEM and RTR), the methodology can scale to accommodate more modules as required.

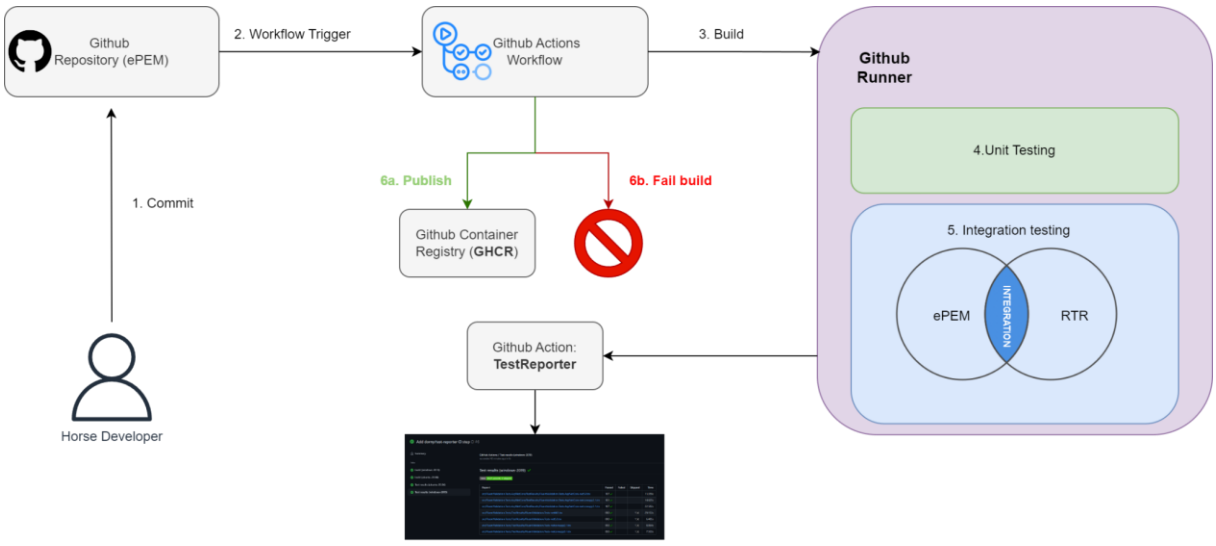


Figure 2: Integration workflow proposed for the HORSE Project

1. **Commit (Step 1)**: The process begins when a HORSE developer makes a commit to the GitHub repository. This commit serves as the initial trigger for the CI/CD pipeline.
2. **Workflow Trigger (Step 2)**: Upon this commit, a GitHub Actions workflow is automatically triggered. This workflow initiates the series of automated tasks in the pipeline.
3. **Build (Step 3)**: The GitHub Actions workflow starts the build process on a GitHub Runner. This runner provides a consistent environment for executing the build tasks.
4. **Unit Testing (Step 4)**: The first task within the build process is to perform unit testing. These tests verify the functionality of individual components (e.g., ePEM and RTR) in isolation, ensuring that each module operates correctly on its own.

5. **Integration Testing (Step 5)**: After the unit tests pass, integration testing is performed. This phase tests the interactions between the ePEM and RTR components, verifying that they work together seamlessly as an integrated system.
6. **Conditional Steps - Publish or Fail (Step 6a & 6b)**:
    a. **Publish (Step 6a)**: If all tests (unit and integration) pass successfully, the workflow proceeds to publish the build. This involves pushing the Docker image to the GitHub Container Registry (GHCR).
    b. **Fail Build (Step 6b)**: If any tests fail, the workflow stops, and the build process is marked as failed. The Docker image is not published, indicating that there are issues to be resolved before the next integration attempt.
7. **GitHub Container Registry (GHCR):** When the build is successful and the Docker image is published, it is stored in the GHCR. This makes the image available for deployment and further testing.
8. **Test Reporting:** The workflow includes a step using the `TestReporter` GitHub Action to generate detailed reports of the test results. This report helps developers understand the outcomes of their build and tests, providing insights into any issues that need addressing.

3.3 Requirements For Integration

To ensure a seamless and effective integration process for the HORSE project, the following needs must be addressed for each repository and project:

1. **Unit Testing**:
   Comprehensive unit tests should be implemented for each existing repository/project. This ensures that individual components function correctly in isolation.

2. **Load Testing and Performance Benchmarking**:
   Scripts and guidelines for load testing and performance benchmarking should be developed. These tests are crucial for identifying bottlenecks and ensuring the system can handle expected loads.

3. **Compilation of Component Dependencies and Interactions**:
   A detailed compilation of all component dependencies for each project in HORSE is necessary. This includes listing all libraries, frameworks, and tools that each component relies on.

4. **Integration Tests**:
   Integration tests should be designed to verify interactions between components. This includes calls to dependencies, API calls, and understanding who calls whom within the system.

5. **Dockerfiles**:
   Dockerfiles need to be created for each component. These files define how the application is packaged into a Docker container, ensuring consistency across different environments.

6. **Docker-Compose File**:
   A Docker-Compose file could be provided to define how multiple Docker containers interact. This includes service dependencies, network configurations, and volumes.

7. **Environment Configuration**:
   Clear and detailed environment configuration is necessary. This includes environment variables, configuration files, and setup scripts for different environments (development, staging, production).

8. **API Documentation**:
   Comprehensive API documentation should be provided. This documentation should include details of all endpoints, request and response formats, and any authentication mechanisms.

3.4 Assets Provided by the Integration Plan

The integration plan provides the following resources and support:

1. **Comprehensive Guide on CI/CD Best Practices**:
   A detailed guide tailored specifically for the HORSE project, outlining best practices for CI/CD.

2. **Instructions and Assistance for GitHub Actions Workflow**:
   Clear instructions and hands-on assistance to help teams build their GitHub Actions workflow `.yml` files. This support ensures that automated testing and deployment processes are correctly implemented.

3. **Example Repositories**:
   Example repositories with pre-configured CI/CD pipelines will be provided. These serve as references to help teams set up their own workflows efficiently.

4. **Templates for Dockerfiles and Docker-Compose Files**:
   Standardized templates for Dockerfiles and Docker-Compose files will be provided to ensure consistency and ease of use across different projects.

5. **Standards and Guidelines for Security and Compliance**:
   Standards and guidelines to ensure that all code and dependencies meet security and compliance requirements. This includes automated checks integrated into the CI/CD pipeline.

6. **Integration of Code Review Tools**:
   Integration of code review tools and processes, such as SonarQube or similar, to ensure high-quality code. These tools help in identifying potential issues early and maintaining code quality throughout the development process.

3.5 Example Continuous Integration Testing (ePEM and RTR)

To test the integration between RTR and ePEM, first some modifications in the files of both projects must be carried out. These are listed below including the explanation for such modifications. Subsequently, taking these changes into account, we explain how to run the integration with GitHub Actions.

3.5.1 Changes in ePEM and RTR files

Use of a shared network

First, there is a need to create a docker external network called, for example, shared network, in order to communicate RTR and ePEM:

```
docker network create shared-network
```

Modification of ePEM `compose.yaml`

There are two modifications that must be done in ePEM `compose.yaml` file: first, to use the shared network in all services; second: to name the `nfvcl` service to be called from RTR:

```
services:
  nfvcl:
    [...]
    container_name: epem
    networks:
      - shared-network

[...]

networks:
  default:
    ipam:
      driver: default
      config:
        - subnet: "10.224.52.0/24"
  shared-network:
    external: true
```

Modification of ePEM Dockerfile

There is one addition that must be done in ePEM Dockerfile to set the `PYTHONPATH` environment variable to `/app/nfvcl-ng/src/nfvcl`, to ensure that Python includes this directory when it searches for modules to import:

```
ENV TZ=Europe/Rome
ENV PYTHONPATH=/app/nfvcl-ng/src/nfvcl
```

Modification of RTR `docker-compose.yaml`

Modify the RTR `docker-compose.yaml` file to use the shared-network in all services:

```
services:
  rtr-api:
    [...]
    networks:
      - shared-network

[...]

networks:
  shared-network:
    external: true
```

Modify the same file to use port `27018` as the host port for `mongodb`. This prevents collision with host port `27017` used by the `mongodb` instance in ePEM.

```
mongodb:
  [...]
  ports:
    - "27018:27017"
```

Modification of RTR `send_mitigation_rules.py` file

With this file there is enough code to do a prelimiary test of RTR - ePEM communication, but some modifications must be carried out to accommodate both sides. The definition of the `simple_uploader` method must be changed to include two parameters that must be included in ePEM `/v2/horse/rtr_request`: `target_ip` and `target_port`. The following code exempt show the updates.

```
def simple_uploader(target_ip, target_port, action_id,
```

```
                    action_definition, service, playbook_yaml)
```

The `receiver_url` must point to the ePEM entry point for horse messages:

```
receiver_url = "http://epem:5002/v2/horse/rtr_request"
```

The params sent in the request must accomodate ePEM API definition:

```
params = {
        "target_ip": target_ip,
        "target_port": target_port,
        "service": service,
        "actionType": action_definition,
        "actionID": action_id
    }
```

Modification of RTR `IBI-RTR_api.py` file

In order to complete the call to the previous `simple_uploader()` method, the code in the `register_new_action()` in `IBI-RTR_api.py` must include some mock data for the two new parameters:

```
target_ip = "192.168.1.12"
target_port = "8080"
action_id = "123"
action_definition = "Service Modification"
service = "DNS"
simple_uploader(target_ip, target_port, inserted_action_id,
    action_definition, service, complete_playbook)
```

3.5.2 Integration Tests

With the previous modifications, it is possible to start the services using `docker compose up` in each project (ePEM and RTR).

Once both services are running, it is possible to use the tests included in the RTR project to check the connectivity. More specifically, two files could be run to check that everything is working properly:

```
pytest RTR/tests/test_user_register.py
pytest RTR/tests/test_create_and_get_specific_action.py
```

The first script creates a user in RTR users collection using RTR REST API. This user is then used to create a mitigation action also using RTR REST API. This action is propagated to the ePEM calling to the ePEM HORSE REST endpoint.

3.6 GitHub Actions (Core framework for HORSE-CI/CD)

GitHub Actions is a CI/CD tool that allows you to automate development workflows directly from a GitHub repository. With GitHub Actions, you can define a series of steps that will automatically execute in response to specific events, such as making a push to the repository (i.e., update the source code). The key components and the necessary secrets required by GitHub actions are detailed below. Subsequently, it is explained how to use GitHub Actions to create ePEM and RTR images that will be stored in the GitHub Container Repository. Following, it is described how to trigger the testing with docker compose with the use of GitHub Actions. Lastly, a workflow chart is shown and summarized for better understanding.

3.6.1 Key Components of GitHub Actions Workflow

These are workflow definitions that describe what actions should be taken when a specific event occurs. Workflows are defined in YAML files stored in the `github/workflows` directory of the project repository.

- **Jobs**: These are units of work or tasks within a workflow. A workflow can have one or more jobs that run either in parallel or sequentially, depending on the established conditions.
- **Steps**: These are individual steps within a job. Each step can execute a bash command or a specific action.
- **Actions**: These are reusable commands that can be used in steps. GitHub provides a large number of predefined actions, and you can also create your own actions.

3.6.2 Secrets in GitHub

Secrets in GitHub are sensitive values, such as API keys or access tokens, that are securely stored and used in GitHub Actions workflows. Secrets are encrypted and only accessible by the repository's workflows.

Adding secrets to HORSE Repository

The steps to create secrets are as follows. A later section will provide a detailed explanation of how to do this and the suggested naming conventions:

1. In the Settings tab of the repository.
2. In the left menu, select Secrets and variables and then Actions.
3. Click on New repository secret.
4. Enter the name and value of the secret and click on Add secret.

Using Secrets in a Workflow

To use a secret in your workflow, you can reference it using the syntax `${{ secrets.NAME_OF_SECRET}}`.

Necessary Secrets for the Build Job of the Workflow

For the correct operation of the build job, it is essential to configure a series of secrets in the GitHub repository. The necessary secrets are detailed in Annex B.

3.7 Explanation of the `rtr.yml` Workflow for Building the RTR Image

The `rtr.yml` workflow is a configuration file used in GitHub Actions to automate the process of building and deploying a Docker image to the GHCR. This workflow runs whenever a push is made to the main branch of the repository. Each section of the workflow is detailed below:

Workflow Name

```
name: RTR CI
```

Workflow Triggers

The workflow is configured to automatically run whenever any changes are made and pushed to the main branch.

```
on:
  push:
    branches:
      - main
```

Jobs

The workflow defines a single job named build, which contains a series of steps to build and deploy the Docker image. The build job runs on a container with the latest version of Ubuntu.

```
jobs:
  build:
    runs-on: ubuntu-latest
```

Repository Checkout

This step uses the action `actions/checkout@v2` to check out the repository's source code into the workspace. This allows subsequent steps to access the source code needed to build the Docker image.

```
  - name: Checkout repository
  uses: actions/checkout@v2
```

Docker Buildx Setup

This step sets up Docker Buildx, a tool that enables the building of multi-platform Docker images. The action `docker/setup-buildx-action@v1` is used to prepare the build environment.

```
  - name: Set up Docker Buildx
  uses: docker/setup-buildx-action@v1
```

Login to GitHub Container Registry

In this step, the login to GitHub Container Registry is performed using credentials stored in repository secrets. The secrets `GHCR_TOKEN`, `GHCR_URL`, and `GHCR_USERNAME` provide the authentication token, registry URL, and username, respectively.

```
  - name: Log in to GitHub Container Registry
  run: echo "${{ secrets.GHCR_TOKEN }}" | docker login "${{ secrets.GHCR_URL
}}" -u "${{ secrets.GHCR_USERNAME }}" --password-stdin
```

Build and Deploy Docker Image

This step performs two main tasks:

1. Building the Docker image using the docker build command. The image is tagged with the value of the `GHCR_IMAGE` secret, which specifies the image name and tag.

2. Publishing the Docker image to the GitHub Container Registry using the docker push command.

```
  - name: Build and push Docker image
  run: |
```

```
    docker build -t ${{ secrets.GHCR_IMAGE }} .
    docker push ${{ secrets.GHCR_IMAGE }}
```

The steps to configure the ePEM workflow for building the ePEM image is similar to the already explained RTR workflow.

3.8 Explanation of the rtr.yml Workflow for Integrating and Testing RTR and ePEM

The `rtr.yml` workflow for integrating and testing RTR and ePEM is a configuration file used in GitHub Actions to automate the process of running integration tests for the RTR and ePEM services. This workflow runs after the build job completes successfully. Each section of the workflow is detailed below:

Workflow Name

```
name: RTR CI
```

3.8.1 Workflow Triggers

The workflow is configured to run the integration tests once the build job has successfully completed.

Jobs

The workflow defines a single job named test, which contains a series of steps to integrate and test the RTR and ePEM services. The test job runs on a container with the latest version of Ubuntu.

```
jobs:
  test:
    runs-on: ubuntu-latest
    needs: build
```

Repository Checkout

This step uses the action `actions/checkout@v2` to check out the repository's source code into the workspace. This allows subsequent steps to access the source code needed for integration testing.

```
- name: Checkout repository
  uses: actions/checkout@v2
```

Docker Buildx Setup

This step sets up Docker Buildx, a tool that enables the building of multi-platform Docker images. The action `docker/setup-buildx-action@v1` is used to prepare the build environment.

```
- name: Set up Docker Buildx
  uses: docker/setup-buildx-action@v1
```

Login to GitHub Container Registry

In this step, the login to GitHub Container Registry is performed using credentials stored in repository secrets. The secrets `GHCR_TOKEN`, `GHCR_URL`, and GitHub.Actor provide the authentication token, registry URL, and username, respectively.

```
-name: Log in to GitHub Container Registry
  run: echo "${{ secrets.GHCR_TOKEN }}" | docker login "${{ secrets.GHCR_URL
}}" -u "${{ github.actor }}" --password-stdin
```

Docker Compose Setup

This step installs Docker Compose, which is used to manage multi-container Docker applications.

```
- name: Set up Docker Compose
  run: |
    sudo apt-get update
    sudo apt-get install -y docker-compose
```

Fetch Docker Compose Files

These steps fetch the `compose.yaml` and `config_compose.yaml` files from the ePEM repository. These files are required to set up and run the ePEM services.

```
- name: Get compose.yaml from ePEM
  id: get-docker-compose
  run: |
    curl -o compose.yaml \
    https://raw.githubusercontent.com/HORSE-EU-
Project/ePEM/HORSE/compose.yaml

- name: Get config_compose.yaml from ePEM
  id: get-docker-compose-config
  run: |
    curl -o config_compose.yaml \
    https://raw.githubusercontent.com/HORSE-EU-
Project/ePEM/HORSE/config_compose.yaml
```

Create Shared Network

This step creates a shared Docker network to facilitate communication between RTR and ePEM services.

```
- name: Create shared-network
  run: docker network create shared-network
```

Run Docker Compose for RTR

This step launches the RTR services using Docker Compose.

```
- name: Run docker-compose for RTR
  run: docker-compose -f docker-compose.yml up -d
```

Wait for RTR Services

This step ensures that the RTR services are ready before proceeding to launch the ePEM services.

```
- name: Wait for RTR services to be ready
  run: |
    while ! curl -s http://localhost:8000 > /dev/null; do
      echo "Waiting for rtr service..."
      sleep 5
    done
```

Run Docker Compose for ePEM

This step launches the ePEM services using Docker Compose.

```
- name: Run docker-compose for ePEM
  run: docker-compose -f compose.yaml up -d
```

Wait for ePEM Services

This step ensures that the ePEM services are ready before proceeding with the tests.

```
- name: Wait for ePEM services to be ready
  run: |
    while ! curl -s http://localhost:5002 > /dev/null; do
      echo "Waiting for epem service..."
      sleep 5
    done
```

Install Dependencies

This step installs the necessary Python dependencies for running the integration tests.

```
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install pytest
```

Run Integration Tests

These steps execute the integration tests using pytest [3]. Each test file is run separately to verify different functionalities.

```
- name: Run get homepage test
  run: pytest tests/test_homepage.py

- name: Run create user test
  run: pytest tests/test_user_register.py

- name: Run login test
  run: pytest tests/test_login.py

- name: Login and get all actions
  run: pytest tests/test_get_all_actions.py

- name: Create a new action
  run: pytest tests/test_create_and_get_specific_action.py
```

Show Logs

This step retrieves and displays the logs from both RTR and ePEM services to help with debugging and verification.

```
- name: Show logs RTR and ePEM
  run: |
    docker-compose -f docker-compose.yml logs rtr-api
    docker-compose -f compose.yaml logs nfvcl
```

Stop Docker Containers

This final step stops all the Docker containers, cleaning up the environment.

```
- name: Stop Docker containers
  run: |
    docker-compose -f docker-compose.yml down
```

3.9 GitHub Actions Workflow Chart

The GitHub Actions Workflow chart is depicted in Figure 3, and it is composed of build job and test job as described below.

Build Job:

1. Log in to GHCR: Authenticate to the GHCR.
2. Build from Dockerfile: Use the Dockerfile to build the service image.
3. Push Image to GHCR: Push the built image to the GHCR for storage.

Test Job:

Launch Services: Deploy the necessary services to run integration tests.

1. Integration Tests: Execute the integration tests to verify the functionality and integration of the services.
2. Get Logs: Retrieve and review logs from the integration tests to ensure everything ran as expected.

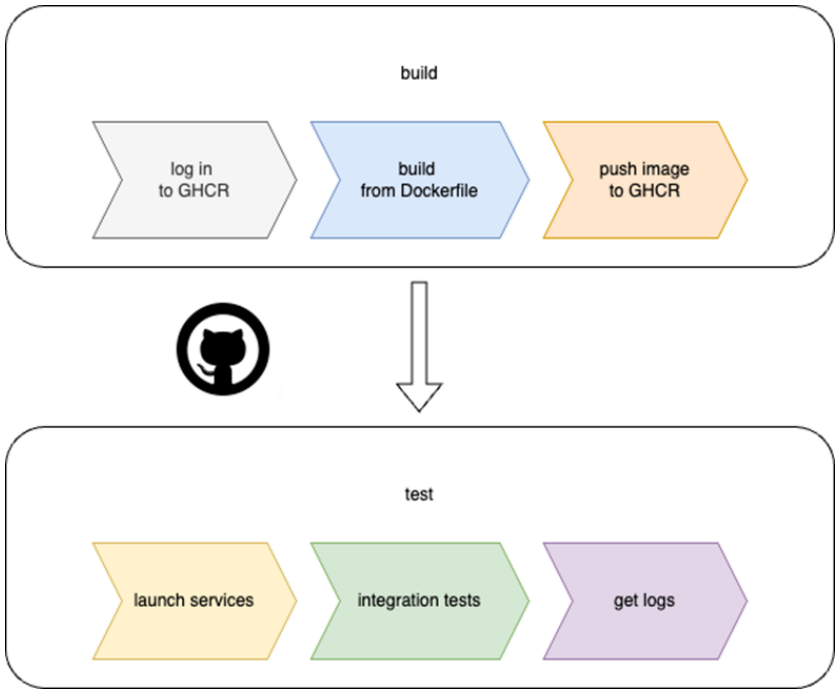


Figure 3: GitHub Actions workflow chart

3.10 Description of the Demonstrators and Functional Validation

As detailed the project settled the necessary integration environment to run automated system integration tests. The automation of the process according to CI/CD good software practices is essential to result in a stable framework and keep track of the integration (including possible

dependences). Table 2 gathers ten system integration tests defined in which input/output and data format are tested and the status (at the moment of editing this document) for HORSE framework version 1.0.

System Test	Integration test points – input/output	Status
1	DEME (mockup) - DTE	COMPLETED
2	DTE - IBI	COMPLETED
3	NDT – IBI (what-if interface, translator included)	PARTIALLY COMPLETED
4	KDB - IBI – RTR	PARTIALLY COMPLETED
5	NDT – EM – KDB (knowledge database)	COMPLETED
6	RTR – ePEM	COMPLETED
7	Preproc – DEME (mockup) [input (datasets)]	COMPLETED
8	ePEM - DOC and Testbeds	COMPLETED
9	Preproc - SM and Testbeds	PARTIALLY COMPLETED
10	SM - PAG	IN PROGRESS

Table 2: Status of integration tests for HORSE framework version 1.0

4 HORSE Framework Validation Matrix

This section includes the validation done for the HORSE intermediate release (IT-1). The HORSE validation process has been done by means of five demonstrators, later described in this section, in the three testbeds of the project. This validation process is facilitated through its proper documentation and monitoring. Towards this direction, we have developed an integration matrix, see Table 3, which represents the interactions between the HORSE components, based on a Design Structure Matrix (DSM) [4]. In this matrix, each row and column represent a HORSE component or the infrastructure (emulated through the three testbeds of the project). It should be noted that a unique code has been used for each interaction which is in the form of [component A id].[component B id]. This code indicates a link from component A to component B. The id of each component is unique, and it is used in all the validation tables.

	1. SM	2. Pre-processing	3. DEME	4. DTE	5. Prediction & Prevention DT	6. Impact Analysis DT	7. IA-DT Model Translator	8. EM	9. PAG	10. IBI	11. RTR	12. ePEM	13. DOC	14. CAS	15. Common Knowledge Database	16. Infrastructure
1. SM																
2. Pre-processing			2.3													
3. DEME				3.4												
4. DTE										4.10						
5. Prediction & Prevention DT				5.4											5.15	
6. Impact Analysis DT							6.7									
7. IA-DT Model Translator						7.6				7.10						
8. EM					8.5											
9. PAG																
10. IBI							10.7				10.11				10.15	
11. RTR												11.12				
12. ePEM													12.13			
13. DOC																13.16
14. CAS																
15. Common Knowledge Database										15.10						
16. Infrastructure																

Table 3: HORSE Framework intermediate release (IT-1) validation matrix

Based on the HORSE framework IT-1 validation matrix (Table 3), and the workflows defined in D2.2 HORSE Architectural Design (IT-1) [2], the HORSE components' interfaces have been developed, which are presented in Table 4, Table 6, Table 8, Table 10 and Table 12. The information they encapsulate includes:

- ID: Unique identifier as set in the section x HORSE integration matrix
- CompA and CompB: Names of the HORSE components (output from component A, input to component B)

- Responsible: Partner(s) responsible for the implementation and documentation of the respective integration endpoint
- Data Type and Protocol used for the information exchange between component A and component B

Furthermore, validation tests have been developed to verify the interactions between the HORSE components. A test case was defined for each interaction pair together with the expected results of the interaction. The information for the different test cases include:

- Unique identifier as set in the HORSE integration matrix;
- CompA and CompB: Names of the HORSE components;
- Test description;
- Test Result;
- Current implementation status for the test.

The technical validation tests are summarized in Table 5, Table 7, Table 9, Table 11, and Table 13, which outline the actual tests performed for each interaction as well as the result and the status of the testing process. Next, the five proposed demonstrators used in the test process are described together with its corresponding interfaces and validation tests.

4.1 Detection and mitigation of DDoS attack over NTP or DNS services

The demonstration will showcase the detection of an attack, using a DDoS DNS (and/or NTP) amplification attack as an example, leveraging the DEME threat detector based on an innovative architecture capable of deploying multiple hierarchical levels of ML. As shown in Figure 4, the attack, following known dynamics, is initiated by a hacker who controls a set of devices that, using the spoofed IP address of the victim, send false requests to the DNS server (and/or NTP server). These requests result in enormous responses that quickly exhaust connectivity resources. The scenario is demonstrated on the CNIT test bed illustrated in Figure 4 and follows the path from the hacker to the victim (Gateway), indicated with arrows of different colors (black for requests to the servers and red for amplified responses).

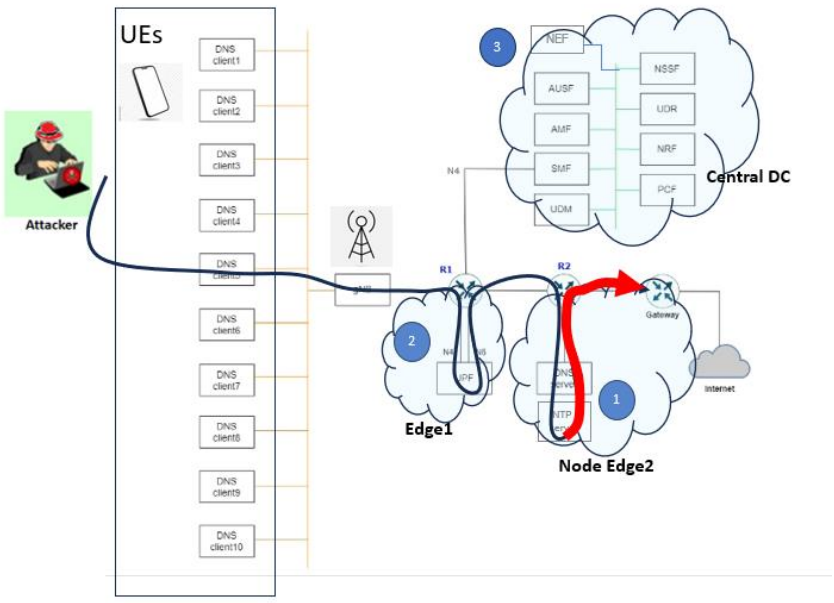


Figure 4: DNS/NTP Attack on the CNIT testbed

The data generated on the test bed through appropriate traffic generators follow theoretically studied trends, verified by dedicated field campaigns, and are appropriately extracted and processed by the upstream components of the DEME threat detector (SM and Pre Processing). These data are supplied to the threat detector in the form of time series that are used to complete both the historical logs, on which the ML model bases its training, and the runtime samples that are processed for detection. The outputs of the detector, including indications of the types of attacks detected and their relative confidence, are then passed to downstream components for the determination and implementation of the best possible mitigation actions based on the specific scenario encountered.

The list of HORSE components involves therefore:

- SM – Smart Monitoring
- Pre-Processing
- DEME – Detector and Mitigation Engine
- DTE – Distributed Trustable AI Engine

The demonstration mainly focuses on the features of the threat detector, DEME, which showcases several important points of innovation. The workflow of the demonstrator is depicted in Figure 5.

- Firstly, from a general perspective, the MML-based Threat Detector was conceived within the HORSE framework to form an innovative and extremely powerful synergy with the Digital Twin. These combined characteristics are deployed synergistically within HORSE to provide a robust defense against cybersecurity attacks. Not only do the different perspectives and predictive capabilities complement each other, but the Digital Twin also provides the necessary assessments for the dynamic and non-disruptive application of mitigation actions triggered by the detection of attacks.
- Secondly, the DEME threat detection block, with its innovative architecture, proves capable of achieving higher learning levels to effectively and rapidly tackle not only 1-day attacks but also 0-day attacks, which, for example, may exploit multiple vulnerabilities in combination.
- Thirdly, the multiple levels of ML will allow, in the context of greater learning, the ability to transcend pre-configured detection thresholds. This enhances the level of automation and dynamic adaptation to network variations.
- Finally, within the combination of tested ML algorithms, an innovative algorithm patented by Ericsson to improve the detection speed will be evaluated.

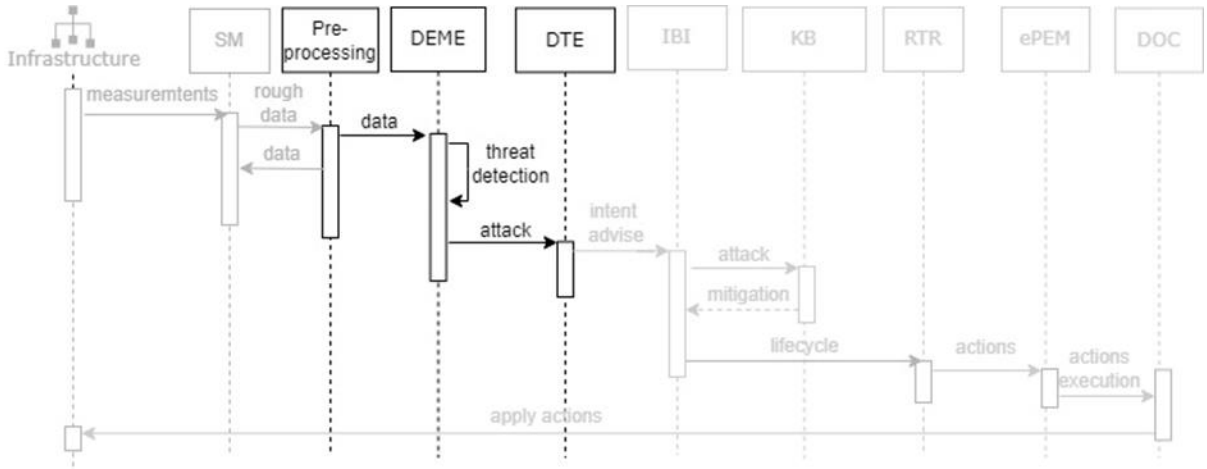


Figure 5: Demonstrator 1 Workflow

Integration Endpoints			Responsible For		Data Type & Protocol
ID	CompA	CompB	CompA	CompB	
2.3	Pre-processing	DEME	8BELLS	ETI	JSON over HTTP/REST API
3.4	DEME	DTE	ETI	NKUA	JSON over HTTP/REST API

Table 4: HORSE components integration endpoints – Demonstrator 1

Interactions			Test Description	Result	Status
ID	CompA	CompB			
2.3	Pre-processing	DEME	Check that the collected data (PCAP files and logs) is processed and transferred to the DEME module.	The collected and processed data (PCAP files and logs) is available at the DEME module.	In progress
3.4	DEME	DTE	Detection of attacks stimulated on the test bed (DNS and/or NTP) for their duration.	Indication of which attacks are occurring, along with their detected type and detection confidence.	In progress

Table 5: HORSE framework interactions testing - Demonstrator 1

4.2 Use of Network Digital Twin to assess the impact of DDoS mitigation actions

The main goal of this demonstration is to present the capacity of doing continuous analysis in a Digital Twin. The modules involved here are the Impact Analysis Digital Twin (IA-DT), including the Model Translator as a part of this DT and the Intent-Based Interface.

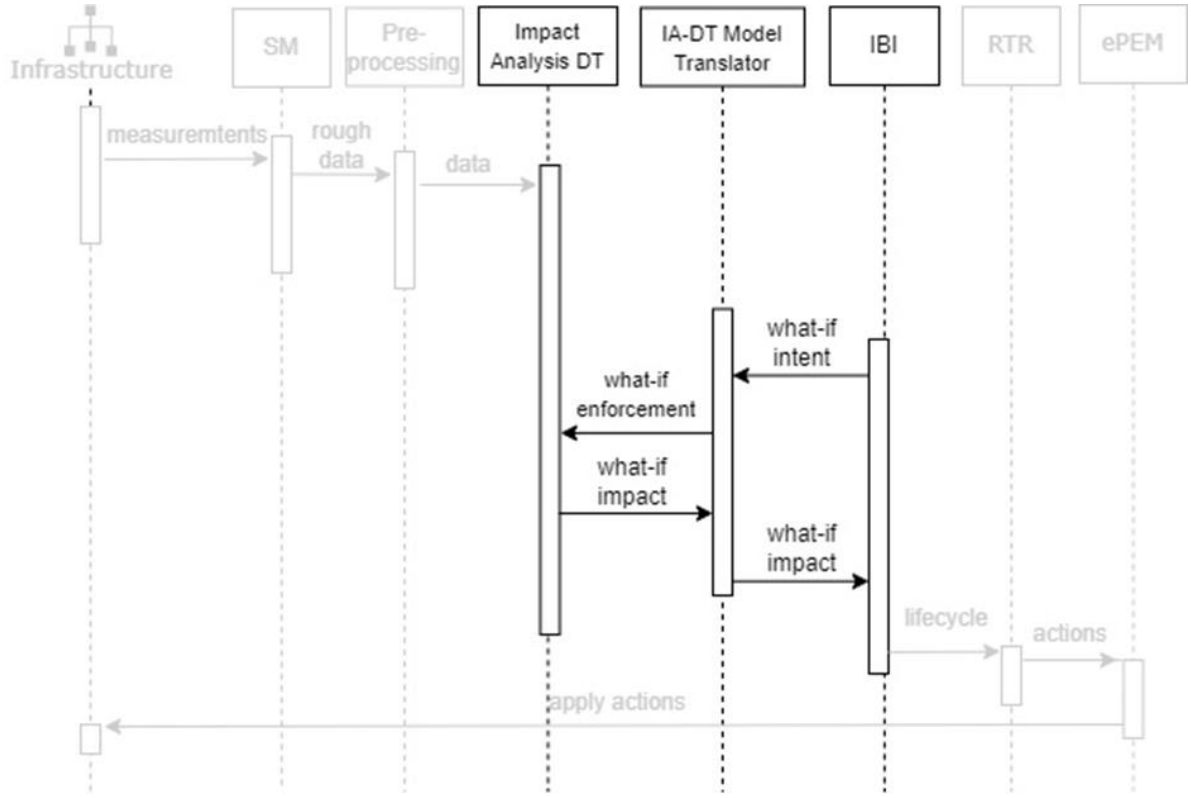


Figure 6: Demonstrator 2 Workflow

As indicated in the workflow depicted in Figure 6, the modules will follow the what-if loop. This means that the IBI will request to the Impact Analysis DT to test some possible mitigations to certain attacks that can appear in the network; in this case, we are working with a DDoS DNS attack. The NDT will replicate the network of the UMU testbed, which is the one acting as the real environment.

The IBI will communicate to the Impact Analysis DT through the Model Translator, which will translate the intents coming and then will enforce the necessary action into the DT environment. The DT will run the simulation and the results will be given back to the IBI also going through the Model Translator, which will transform the result into a suitable format for the IBI. Following this workflow, the IBI will be able to estimate the impact that the mitigation will have and later it can decide whether the action will be applied on the real network.

Table 6 shows the components and interfaces involved in this demo. it is a bidirectional interface between the Model Translator and the IBI: first one (10.7) to receive the intents from the IBI and the second one (7.10) to send back the results from the Digital Twin.

Integration Endpoints			Responsible For		Data Type & Protocol
ID	CompA	CompB	CompA	CompB	
10.7	IBI	IA-DT - Model Translator	TUBS	TID	JSON over HTTP/REST API

7.6	IA-DT Model Translator	Impact Analysis DT	UMU	TID	-
6.7	Impact Analysis DT	IA-DT Model Translator	TID	UMU	-
7.10	IA-DT Model Translator	IBI	TID	TUBS	JSON over HTTP/REST API

Table 6: HORSE components integration endpoints - Demonstrator 2

Associated to this demonstration we also have an integration test. This integration test, described in Table 7, includes the interface between the IBI and the IA-DT Model Translator to check that the intents correctly arrive to the Model Translator and following the execution it also checks if the response from the DT comes back to the IBI.

Interactions			Test Description	Result	Status
ID	CompA	CompB			
10.7	IBI	Impact Analysis DT	Check that IBI what-if intents arrive to the Model Translator of the DT	Data is in the Model Translator	In progress
7.10	Impact Analysis DT	IBI	Check that the response of the DT arrives to the IBI.	Data is in the IBI	In progress

Table 7: HORSE framework interactions testing - Demonstrator 2

The main innovation behind this demonstration is the what-if loop between the IBI and the IA-DT. It allows us to test and estimate the impact in advance of mitigation actions and (re)configuration of network elements. By this loop, the real network will not suffer any inconvenience as all the modifications can be done previously in the IA-DE and we can estimate the impact that the action will have in the real environment.

4.3 Enforcement of Mitigation Actions over a Multi-domain Infrastructure

The Demo showcases the workflow of mitigating an attack after the HORSE Context detects an anomaly in the infrastructure. To achieve that goal, the IBI component will receive the notification of the threat detection, and a high-level intent to mitigate the detected threat. The IBI in its turn, will process the intent and convert it to policies that should be enforced in the infrastructure to mitigate the threat or the attack. The process involves some steps, aiming to show:

- IBI: The IBI receives high-level intents from the detection mechanisms and converts them into policies that will be applied to the infrastructure. The translation of the mitigation

intents into policies also considers the network operator's intents about services in the network, therefore proposing policies that try to fulfill both intents. After the translation process, the policies are sent to the RTR module.

- RTR: Receives the policies from the IBI, interprets it, and produces an Ansible playbook. This playbook contains the tasks needed to configure HORSE's topology and is then forwarded to the ePEM.
- ePEM: Decide If the ansible playbook is applied directly on the infrastructure or forward this action to DOC module in order to apply in a multicluster environment.
- DOC: Covert the JSON received via Northbound interface in a particular data model provided by owner of each multicluster testbed. The DOC module is an abstract layer between Horse Context and the different types of multicluster manager and orchestrators. The DOC responds with the status of each action to allow hierarchically higher horse components to know the status of each mitigation.

This whole interactions between different components in this particular demonstrator are shown in Figure 7 below.

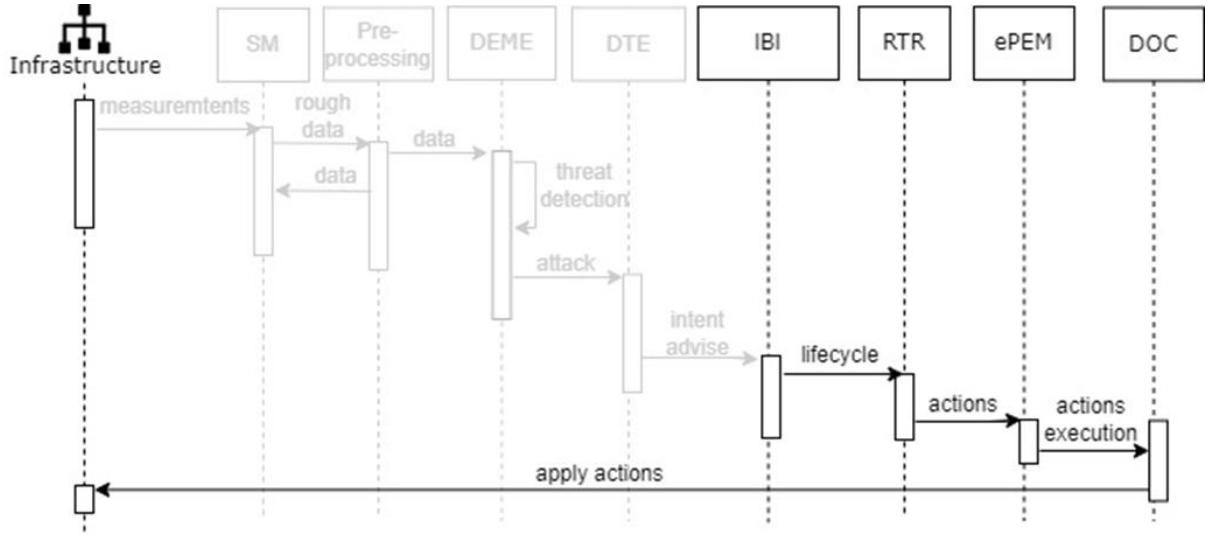


Figure 7: Demonstrator 3 Workflow

The interactions involved in the workflow of converting a high-level intent into a particular command or policy and the corresponded interfaces are shown in Table 8.

Integration Endpoints			Responsible For		Data Type & Protocol
ID	CompA	CompB	CompA	CompB	
10.11	IBI	RTR	TUBS	8BELLS	JSON over HTTP/REST API
11.12	RTR	ePEM	8BELLS	CNIT	YAML over HTTP/ REST API
12.13	ePEM	DOC	CNIT	ATOS	JSON over HTTP/ REST API

13.16	DOC	Infrastructure	ATOS	UMU	XML over HTTP/ REST API

Table 8: HORSE components integration endpoints - Demonstrator 3

Finally, associated with this Demonstration we developed some integration tests to verify the successful behavior between each pair of components in an isolated way. The tests are listed in Table 9.

Interactions			Test Description	Result	Status
ID	CompA	CompB			
10.11	IBI	RTR	Check if the RTR is able acknowledge the receiving of the policy sent over the REST API	An HTTP code response equals to 200.	In Progress
11.12	RTR	ePEM	Check if the ePEM is able to receive and enforce the Ansible playbook formed based on IBI's policy.	An HTTP code response equals to 202	In Progress
12.13	ePEM	DOC	Check if DOC is able to map mitigation action into a certain way to enforce actions and response status of it.	DOC enforces mitigation action	In Progress
13.16	DOC	Infrastructure	Check if DOC and infrastructure are able to enforce mitigation action.	Infrastructure is deployed in different locations and mitigations are applied	In Progress

Table 9: HORSE framework interactions testing - Demonstrator 3

Specifically, this demonstrator will apply actions to mitigate DDoS DNS attacks in the network. The attacks in this demonstrator are simulated since the main goal is to analyze how HORSE architecture react after the detection of an attack.

This demonstrator is executed in the UMU Testbed whose infrastructure provide a multi-cluster environment managed by *Liqo* with a 5G core network based on *Open5GS* and DNS servers running with ISC BIND9 DNS server [5]. Each cluster is deployed in a different geographical position as shown in Figure 8.

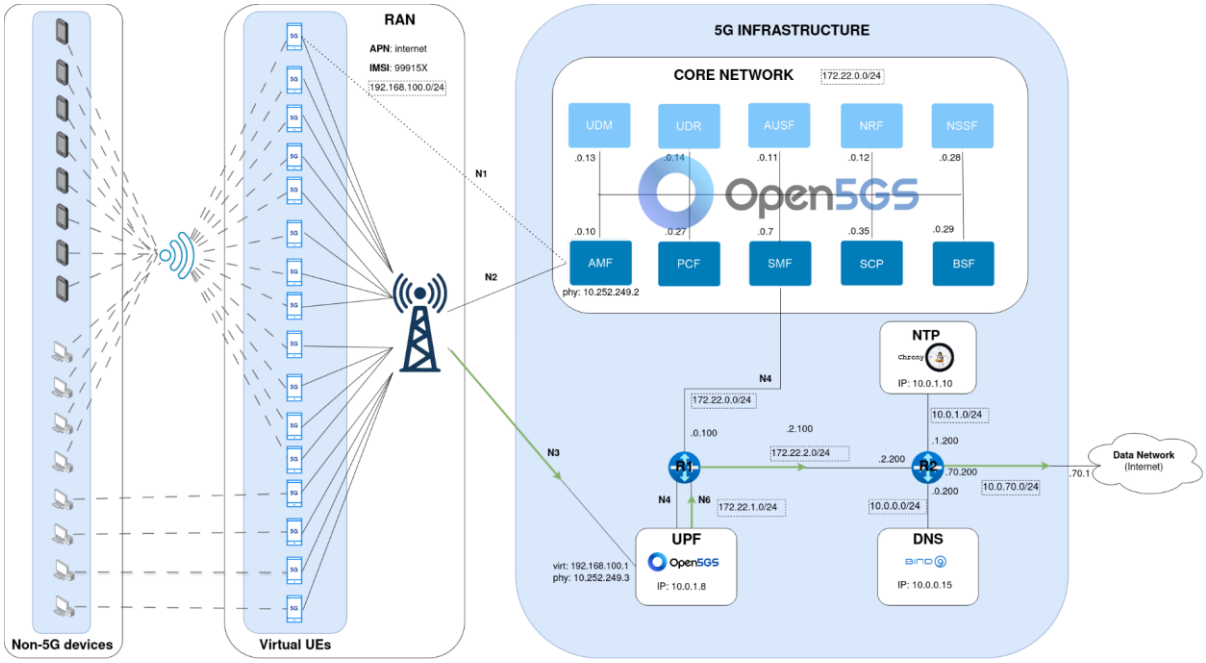


Figure 8: Deployment of the demonstrator in a mutli-cluster environment on the UMU Testbed

This demo closes the loop in the HORSE context with the enforcement of a mitigation action in different geographical position and testbed technologies in a dynamically way, combining high-level intents with each particular way to apply actions in each technology.

4.4 Prediction of Attacks using a Network Digital Twin

This demo showcases the capabilities of prediction of anomalies and security attacks of the HORSE platform. The prediction is based on the usage of the P&P NDT.

In this demo, the P&P NDT will be triggered by the EM module, requesting to analyze the current system status and to perform predictions on specific types of attacks, provided in the input by the EM component and describing which modules and scenario parameters to predict. The P&P NDT will acquire the current snapshot of the state and traffic flows in the physical twin of the 6G Network via the Common Knowledge Base (cKB), perform the requested analysis and provide an input to the DTE to inform whether a threat is predicted and its related probability of occurrence.

The simulated attack will be the DDoS, and the reference testbed environment in the first phase will be CNIT testbed.

Involved components include:

- cKB - Common Knowledge Base module will provide context, setup and real time traffic on the HORSE testbed (representing the Network Physical Twin).
- EM - Early Modeling module will trigger the P&P NDT module by providing specifications of the attacks to detect, the potential recipients, and the related setup parameters to monitor (e.g. unusual computational load, sharp input traffic increase, anomalies in the control plane signaling).
- P&P NDT - Prediction & Prevention Network Digital Twin will replicate the status of the physical network infrastructure and running services, and it will provide predictions about the near future potential anomalies and attacks. The module will be capable of accurately replicating traffic flows, actual services behavior and related traffic, and to generate traces for further processing – if required. In this first validation phase, synchronization between physical and digital twin will be performed during the triggering phase, while in the final

validation it will be capable of periodically re-synchronize its status if most recent data will be available.

- DTE - Distributed Trustable AI Engine is not directly affected, but the P&P NDT will generate an output compatible with its input interface, ready for the full integration of the HORSE architecture.

The innovations presented in the demo are the following:

- First “real time” implementation of a 5G/6G NDT
- Traffic mirroring between Physical and Digital Twin of the network
- Threat modelling and prediction in a DT environment
- Network state estimation and real-time synchronization

The workflow of the demonstrator is described in Figure 9, while the interfaces between components of this demonstrator are listed in Table 10, and its associated tests in Table 11.

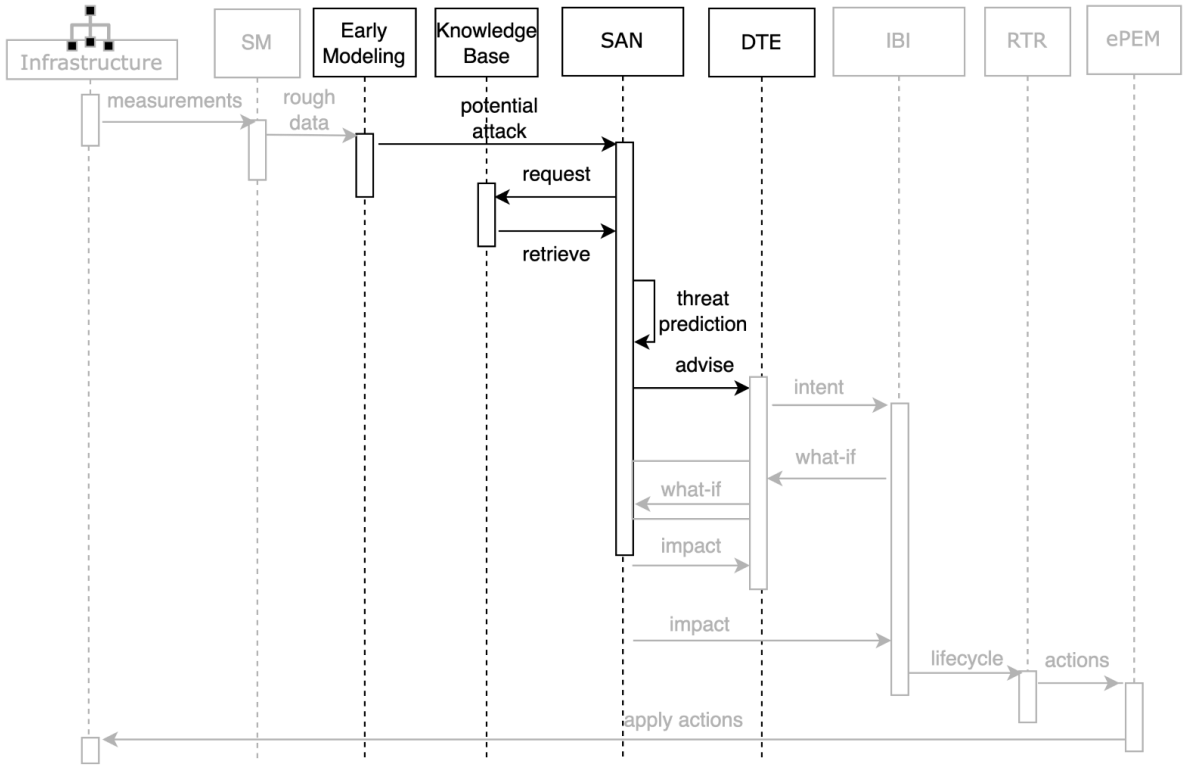


Figure 9: Demonstrator 4 Workflow

Integration Endpoints			Responsible For		Data Type & Protocol
ID	CompA	CompB	CompA	CompB	
8.5	EM	Prediction & Prevention DT	UPC	CNIT	XML over HTTP

5.15	Prediction & Prevention DT	cKB	CNIT	8BELLS	Different files: topology xml file, PCAP network traffic
5.4	Prediction & Prevention DT	DTE	CNIT	NKUA	JSON over HTTP

Table 10: HORSE components integration endpoints – Demonstrator 4

Interactions			Test Description	Result	Status
ID	CompA	CompB			
8.5	EM	Prediction & Prevention DT	Check that the required threat model instance is transferred to the Prediction & Prevention DT.	The threat model instance is available at the Prediction & Prevention DT.	Completed
5.15	Prediction & Prevention DT	cKB	The P&P NDT acquires the network topology, service topology, 5G setup and most recent traffic flows	The P&P NDT self-configures itself and it ready to perform predictions	Completed
5.4	Prediction & Prevention DT	DTE	The P&P NDT sends to the DTE an estimate of the potential security attack	An HTTP message containing the XML specifications of the predicted attack are generated consistently with the interface between DEME and DTE	Completed

Table 11: HORSE framework interactions testing - Demonstrator 4

4.5 Using the Distributed Trustable Engine to detect PFCP attacks

This demo showcases the detection and mitigation of attacks on the N4 interface of the 5G core network using ML and the DTE processing for creating different types of intents that are sent to the IBI, in order to implement appropriate mitigation and prevention measures. This process involves various steps, aiming to show: 1) the DEME-DTE API where ML attack detection and classification results are given as input to the DTE, 2) the DTE processing of this information and creation of intents, and 3) the DTE-IBI API where these intents are forwarded to the IBI. The demo includes three steps, which are highlighted in Figure 10.

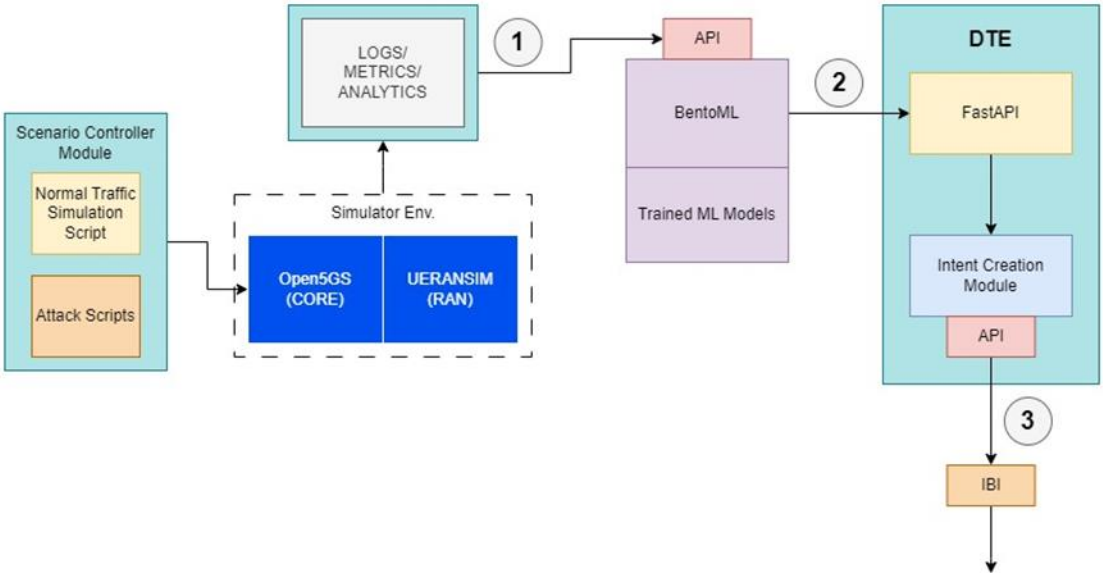


Figure 10: The three-step process of the PFCP attack demonstrator

More specifically, these steps are given in detail below:

- Step 1 - Execution of PFCP attacks: To create a realistic threat environment, this demo employs Scapy [6], a Python-based packet manipulation tool, to simulate various types of attacks on the N4 interface. A script is developed to prompt a malicious Session Management Function (SMF) to randomly execute an attack every X minutes, where X is a random number between 1 and 30 minutes. In greater detail, the following types of attacks are simulated:
  - Unauthorized PFCP Session Modification Request: For this scenario, the adversary aims to manipulate the UPF to discard packet handling settings. The malicious user achieves this by sending a PFCP Session Modification Request that includes a DROP flag in the Apply Action field of the FAR rules. This action results in the deletion of the Tunnel Endpoint Identifier (TEID) and the IP address of the gNB from the UPF. Consequently, while the connection between the UE and the gNB remains active, the client is unable to access the data network (DN).
  - Unauthorized PFCP Session Deletion Request: This attack sends unauthorized PFCP session deletion requests from the SMF to the UPF, aiming to disconnect a specific UE from the DN without disrupting its connection to the 5G Radio Access Network (RAN) or Core Network (CN). It targets PDU sessions on the N4 interface, affecting connectivity observed on the N3 interface. The only remedy for the affected UE is to restart its session or connect to another gNB, which generates a new SEID, halting the attack's effect.
  - Unauthorized PFCP Session Establishment: This attack is instantiated from the SMF of the 5G CN. The target of this attack is the UPF, which handles processes and forwards user data to the DN. The goal of this attack is the exhaustion of the UPF's resources to handle legitimate Session Establishment Requests and Heartbeat Requests. This will potentially hinder the capability of the 5G CN to successfully formulate new PDU sessions between clients and DN.

- Step 2 - Data collection and monitoring: In order to monitor and analyze the network operation and data traffic, two tools are mainly used:
  - Wireshark, a network protocol analyzer, capturing network traffic and generating datasets with the proper features that are exploited to update the ML model stored in BentoML.
  - Prometheus, an open-source monitoring and alerting toolkit, acquiring logs from various NFs, such as the AMF, SMF, and UPF.
- Step 3 - ML Models performance improvement: The next step focuses on using the captured datasets to improve the performance of ML models. The datasets generated by Wireshark are used to enhance the detection and classification of the ML models deployed via BentoML. These models are continuously updated with new data collected from the test-bed, ensuring their adaptation to evolving threat patterns and achieving high detection rates.

4.5.1 Testbed and Context Environment

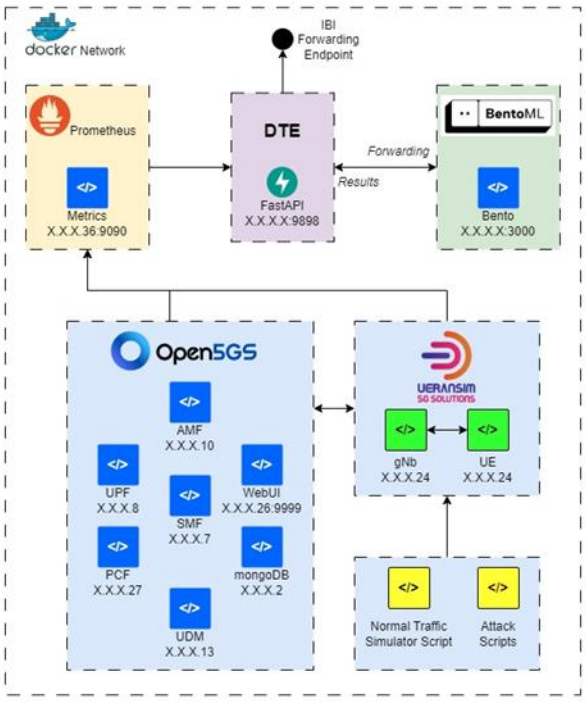


Figure 11: Test-bed architecture for the PFCP attacks demonstrator

This demo is executed in a fully operational 5G CN and RAN environment, based on Open5GS and UERANSIM. The system is entirely containerized using Docker, thus ensuring scalability, modularity, and easy deployment. The overall architecture is shown in Figure 11. In addition, this setup includes a user-friendly WebUI for UE registration and configuration, along with advanced network monitoring and ML tools, i.e. Prometheus, Wireshark, and BentoML. In this way, the developed demo environment allows testing, development, and validation of the DEME-DTE and DTE-IBI APIs and interactions, mainly focusing on the use of ML capabilities and the creation of appropriate mitigation and prevention intents. More specifically, below the different tools and frameworks used in the demonstrator and their corresponding role is discussed:

- Open5GS: An open-source project providing a complete 5G core network, including the AMF, SMF, and UPF. Open5GS forms the backbone of our 5G core network, enabling comprehensive network functionalities and services. Open5GS also provides a useful WebUI to register/deregister network users.

- UERANSIM: A robust UE and RAN simulator that interacts with the 5G CN. UERANSIM provides a realistic simulation environment for testing and development, allowing the evaluation of the network's performance and different security measures under varying numbers of UEs.
- Docker: The entire 5G CN and RAN are containerized using Docker, ensuring a modular, scalable, and easily deployable infrastructure. This approach simplifies the management and orchestration of NFs, facilitating updates and maintenance to accommodate more complex attack scenarios and corresponding intents.
- BentoML: This platform is utilized for storing, deploying, and managing ML models. In this demonstrator, BentoML plays a crucial role in the deployment of models designed for attack detection and classification, enhancing the security and reliability of the network. BentoML Utilizes an easy-to-use API in order to serve and output the inference results.
- Prometheus: An open-source monitoring and alerting toolkit, enabling the collection and monitoring of various metrics from all NFs within the 5G core network. Prometheus enables real-time visibility into the network's performance and health, facilitating proactive management and troubleshooting.
- Wireshark: A powerful network protocol analyzer employed for monitoring network traffic, creating datasets, and updating ML models. Wireshark's capabilities enable detailed inspection and analysis of the data traffic in the network, providing valuable insights for improving network performance and security.

4.5.2 Innovations Presented in the Demo

This demo provides an efficient way to simulate PFCP attacks on the 5G core networks and showcases how ML-based attack detection and classification is leveraged by the DTE to create proper intents. At this stage, the following innovations are provided:

- Integration of ML capabilities, deployed through BentoML to detect and classify PFCP attacks in the 5G core network in real-time, enhancing the network's responsiveness to threats. At the initial implementation of DTE for IT-1, classification models have been used to identify network anomalies. However, deep reinforcement learning (DRL) approaches will also be integrated in DTE in the next iteration, to leverage more dynamic responses and real time updates on the adopted mitigation strategies.
- Adoption of intent-based networking principles for dynamic and adaptive network configurations to mitigate attacks. In this respect, the DTE creates two types of intents:
  - Mitigative intents, describing objectives that should be accomplished in the short term (e.g., within a few seconds) to reduce the impact of ongoing security incidents. For instance, an intent can recommend to “Mitigate PFCP Session Deletion attack against UPF X” with the following requirements: “SMF downtime less than 1 sec, rate of PFCP Session Deletion requests reaching UPF X equal to 0”. In this case, the mitigation could be supported by configuring a firewall to drop all PFCP Session Deletion requests.
  - Preventive Intents, describing objectives that should be achieved in the longer term to ensure protection against future potential threats. For example, an intent can advise to “Prevent all PFCP-related attacks against UPF with the following requirement: “SMF downtime less than 150 sec” could be supported by checking the integrity of the SMF and, if found compromised, redeploying the SMF, creating a new slice, or deploying a new core network.

The complete workflow of the demonstrator is depicted in Figure 12 below while the interfaces and associated tests are listed in Table 12 and Table 13 respectively.

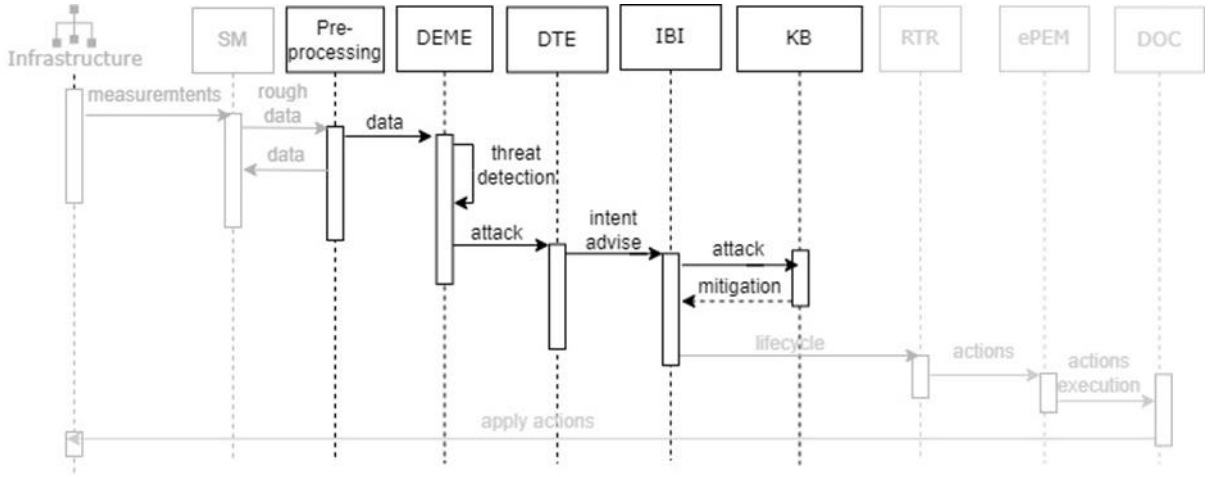


Figure 12: Demonstrator 5 Workflow

Integration Endpoints			Responsible For		Data Type & Protocol
ID	**CompA**	**CompB**	**CompA**	**CompB**	
2.3	Pre-processing	DEME	8BELLS	NKUA	JSON over HTTP/REST API
3.4	DEME	DTE	NKUA	NKUA	PCAP, JSON over HTTP/REST API
4.10	DTE	IBI	NKUA	TUBS	JSON over HTTP/REST API
10.15	IBI	KB	TUBS	MAR	JSON over HTTP/REST API

Table 12: HORSE components integration endpoints - Demonstrator 5

Interactions			Test Description	Result	Status
ID	**CompA**	**CompB**			
2.3	Pre-processing	DEME	Check that the collected data (PCAP files and logs) are processed and transferred to the DEME module.	The collected and processed data (PCAP files and logs)	In progress.

				is available at the DEME module.	
3.4	DEME	DTE	Check if the DTE correctly receives the DEME's detection data, being sent over the REST API	PCAP, JSON over HTTP/REST API	In progress.
4.15	DTE	IBI	Check if the IBI can correctly receive security intents regarding the detected attack from DTE	An HTTP response with code 200 to acknowledge the reception of the intent.	In progress
15.4	IBI	KB	Check if the IBI can correctly receive mitigation lists given the detected attack	JSON over HTTP/REST API	In progress

Table 13: HORSE framework interactions testing - Demonstrator 5

4.6 Compliance Matrix for HORSE Iteration IT-1

In this section we offer the compliance matrix (see Table 14) that relates the requirements listed in D2.1 HORSE Landscape: Technologies, State of the Art, AI Policies and Requirements [1] versus the validation tests and demonstrators detailed in previous sections and executed in the first phase.

Requirement	Comment	Covered in Demonstrator #	Status
REQ-F-03	Auditing of messages of IBI module will is optional and will be covered in IT-2		Optional/Pending for IT-2
REQ-F-12	Based on latest adjustments in the architecture RTR is not responsible for detecting or determining the nature of the attack. The requirement is optional, and it needs to be updated in IT-2		Not covered.

REQ-F-13	Reconfiguration of the network to mitigate attacks is already covered by the HORSE pipeline	Demonstrator 1 and Demonstrator 5	Covered
REQ-F-14	The horse platform has implemented a methodology for the prediction of an attack	Demonstrator 4	Covered
REQ-F-15	RTR is able to generate enforcement policies that can change the configuration of the network infrastructure.	Demonstrator 3	Covered
REQ-F-16	The Impact Analysis Digital Twin enables the HORSE components to test configuration before its deployment. However, the EM is not responsible for testing and evaluating configurations. The requirement needs to be updated.		Covered
REQ-F-17	Persistence of information is already implemented in the IBI component by storing the security and QoS intents in an internal knowledge base	Demonstrator 2, Demonstrator 3 and Demonstrator 5	Covered
REQ-F-18	The decisions internally taken by the IBI module are shown through a GUI/Dashboard	Demonstrator 2, Demonstrator 3 and Demonstrator 5	Covered

REQ-F-20	The HORSE platform should identify and model the 6G components. The requirement is optional and could be implemented in IT-2		Partially covered
REQ-F-21	The HORSE platform must be able to model the attack	Demonstrator 4	Covered
REQ-F-23	Some components can define access policies, but they are not enforced in real-time in our current demonstrators. Not all components implement access policies. This wills. To be extended in IT-2.		Partially covered
REQ-F-25	Some components can define access policies, but they are not enforced in real-time in our current demonstrators. Not all components implement access policies. This wills. To be extended in IT-2.		Partially covered
REQ-F-26	The RTR is able to provide mitigation action and enforcement policies that can be applied in the network systems.	Demonstrator 3	Covered
REQ-F-27	The user can check the reconfigurations and policies proposed by the IBI in the internal GUI/Dashboard.	Demonstrator 2, Demonstrator 3, Demonstrator 4 and Demonstrator 5	Partially covered

	EM provides an XML file with a list of mitigation actions to mitigate or avoid an attack		
REQ-F-29	Covered by all modules composing the workflow. The RTR is able to generate network configuration policies. The enforcement of the policies is covered by the ePEM and the DOC modules.	Demonstrator 2, Demonstrator 3 and Demonstrator 5	Covered
REQ-F-31	The HORSE platform should use anonymized data to train AI model to detect threats and attacks	Demonstrator 1 and Demonstrator 5	Covered
REQ-F-32	The DTE module can output intents and alarms based on AI/ML to protect the system against potential attacks	Demonstrator 1 and Demonstrator 5	Partially covered
REQ-F-33	The HORSE Digital Twin should be able to consistently repeat specific experiments, and to incorporate controlled variations to experiments execution as requested by its users.	Demonstrator 2 and Demonstrator 4	Covered
REQ-F-34	The HORSE Digital Twin should be capable of deploying different network functions to test them independently or to	Demonstrator 2 and Demonstrator 4	Covered

	model whole functionality sets or planes as a single entity, according to specific experiment.		
REQ-F-35	The anonymization can be performed, but it is not yet executed on the datasets.		Partially covered
REQ-F-36	The HORSE platform must support end-to-end data encryption for data in transit		Not implemented
REQ-F-37	The HORSE platform should allow the user to monitor the status (successful or failed execution) and view an incident summary of all AI pipelines		Not implemented
REQ-F-38	The data retention policies can be defined but are not enforced in real-time on collected datasets		Partially covered

Table 14: Compliance matrix between the functional requirements and functional validation

As Table 14 shows, the validation is fully covering 12 of the 44 (27%) requirements in the validation phase 1 of the first iteration of the HORSE Project, while 19 of 44 (43%) requirements are addressed when considering the ones partially addressed or implemented. We expect to increase this percentage in the second iteration and incorporate the validation of some performance requirements in the validation scenarios and the two general use cases of the project, namely light metro communications case and immersive media case.

5 Conclusion

The HORSE project has completed its first iteration, integrating various components to form a cohesive cybersecurity platform for 5G/6G networks. This milestone marks a significant step in developing a robust solution that addresses cybersecurity challenges in next-generation networks.

On the one hand, the development and integration process of the HORSE components revealed critical insights into the system's architecture, highlighting areas that require updates and improvements. On the other hand, despite the intrinsic complexity of integrating diverse modules, the adoption of a Continuous Integration/Continuous Deployment (CI/CD) approach allowed us to manage the process effectively, preparing the HORSE platform to run five demonstrators and allowing the functional validation of the components.

Moving forward, the next steps in the HORSE development will involve addressing the identified updates in the architecture and incorporating functionalities in the components that will address the identified pending requirements.

Future updates and enhancements will be guided by the work carried out in Work Package 2 (WP2), a strategic move that ensures the HORSE architecture evolves to meet the dynamic needs of cybersecurity in 5G/6G networks. The ongoing collaboration and rigorous validation efforts will continue to propel the project toward its goal of delivering a comprehensive and effective cybersecurity solution. Overall, the progress made in this first release sets a solid foundation for future developments, promising continued advancements and innovations in the HORSE project.

6 References

[1] HORSE Project, “D2.1 HORSE Landscape: Technologies, State of the Art, AI Policies and Requirements”. June 2023. Available at: https://www.horse-6g.eu/?wpdmdl=491&ind=1698663024683, Retrieved on 23/06/2024.

[2] HORSE Project, “D2.2 HORSE Architectural Design (IT-1)”. June 2023. Available at: https://www.horse-6g.eu/?wpdmdl=492&ind=1698663068817, Retrieved on 23/06/2024.

[3] H. Krekel, B. Oliveira, R. Pfannschmidt, F.Bruynooghe, B. Laugher, F. Bruhin, 2004, Available at: https://www.pytest.org/en/8.2.x/, Retrieved on 23/06/2024.

[4] University of Stuttgart, “The Design Structure Matrix (DMS) home page”, Available at: https://dsmweb.org/introduction-to-dsm/, Retrieved on 24/11/2023.

[5] Internet Systems Consortium, “Bind 9”, Available at: https://www.isc.org/bind/. Retrieved on 23/06/2024.

[6] R. R. S, R. R, M. Moharir and S. G, "SCAPY- A powerful interactive packet manipulation program," 2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS), Bangalore, India, 2018.

Annex A

First Thoughts on General KPIs

As mentioned in the previous deliverable D5.1, the validation of the IT-1 HORSE release will be supported by demonstrations and functional tests assuring that all developed modules and all developed components are properly designed and properly integrated, in accordance with what has been specified. However, for the second Intermediate release (IT-2), it is defined that the validation phase will cover technical requirements and the KPI analysis and definition has been performed in the scope of the WP5 activities.

Several Indicators were investigated and the evaluation of the HORSE platform's performance, will be measured using the following KPI's:

- Impact on the performance of the HORSE platform
- Detection Rate/True Positive Rate
- Mean Time to Detect (a specific attack) - MTTD
- Mean Time to React (to a specific attack) - MTTR
- Time to deactivate the mitigation action

KPI's definition

- Impact on the performance of the HORSE platform: Some measurements (latency and bandwidth) must be performed to verify the behavior of the network when the HORSE modules are running (activated) or not running (de-activated). This will lead to a better perception if there is a network degradation in the presence of HORSE modules activities.
- Detection Rate (DR) is also called True Positive Rate (TPR) - Is the ratio between the number of correctly predicted attacks (positive) and the total number of attacks. The Date Rate can be expressed mathematically as

$$DR = \frac{\mathrm{TP}}{\mathrm{TP+FN}},$$

  where TP is the number of correctly predicted attacks, and FN is the number of observations that are predicted as normal but are attacks.
- MTTD: For each attack being tested the time to detect the alarm must be measured and registered. Quick detection reduces the window of exposure.
- MTTR: For each attack being tested, the time to activate the mitigation actions must be measured and registered. Rapid response is crucial to mitigate damage. It must also be measured the time to generate an external alarm. This alarm should be generated to ensure that the Operation Control and Security Centers are aware of an eventual intrusion in the system.
- Time to deactivate the mitigation action- For each attack being tested the time to deactivate the mitigation actions must be measured and registered (if it is considered that the attack is not present in the system)

A best possible balance among these KPI's will lead to an effective cybersecurity solution. For instance, no mitigation action can be effective if the network is already in chaos, motivated by a late detection of an attack.

Therefore, evaluating these metrics collectively, is crucial for crafting a practical and effective solution, allowing fine-tunning and further improvements during the life cycle of HORSE project.

References

- A. Alhomoud, R. Munir, J. P. Disso, “Performance Evaluation Study of Intrusion Detection Systems”, Procedia Computer Science, 2022. DOI: https://doi.org/10.1016/j.procs.2011.07.024.
- Saylor Academy, “Intrusion Detection Systems”, CS406: Information Security. Available at: https://learn.saylor.org/mod/book/view.php?id=29755&chapterid=5439.
- V. Ford, A. Siraj. "Applications of machine learning in cyber security", *Proceedings of the 27th international conference on computer applications in industry and engineering*. Available at: https://www.researchgate.net/publication/283083699_Applications_of_Machine_Learning_in_Cyber_Security
- T. Subbulakshmi, S. M. Shalinie, and A. Ramamoorthi, “Detection and Classification of DDoS Attacks using Machine Learning Algorithms”, European Journal of Scientific Research, 2010.

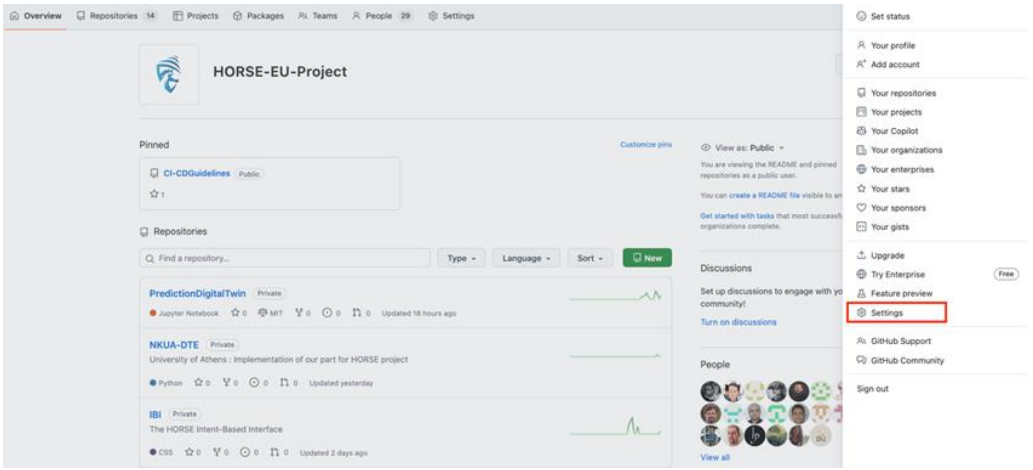
Annex B

This annex provides a detailed guide for configuring authentication tokes and secrets in GitHub to enable developers to build container images of the HORSE components and the proper functioning of integration tests.

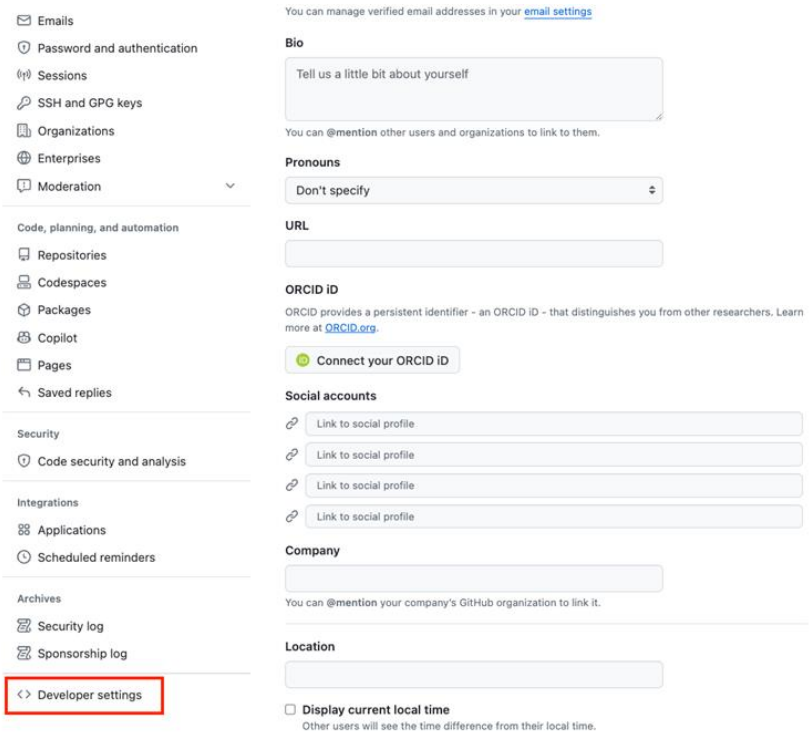
6.1 GHCR_TOKEN

Authentication token for the GitHub Container Registry. This token allows to log in to the container registry and perform operations such as pushing Docker images. Unlike the following three variables, this one is not created as described in the "How to Add Secrets" section, which is intended for repository secrets, but is created at the GitHub user level. Detailed steps are explained below to get the token value: After getting the token value, it is needed to create a new repository secret.

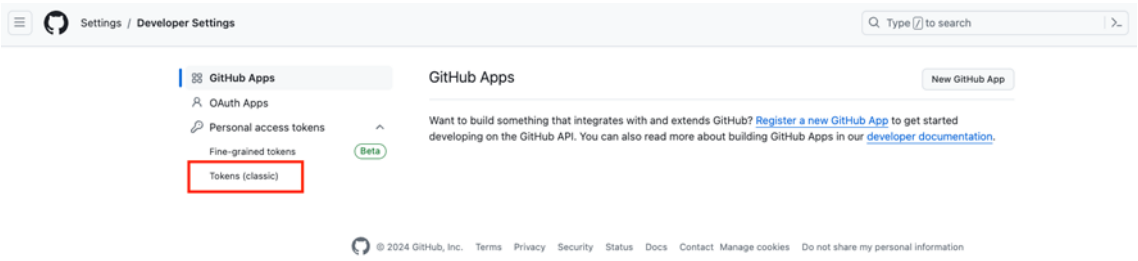
1. Go to the organization Settings:



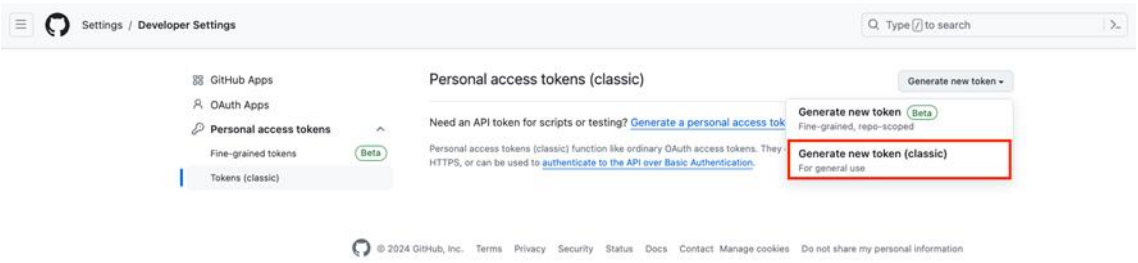
2. Go to the Developer settings:



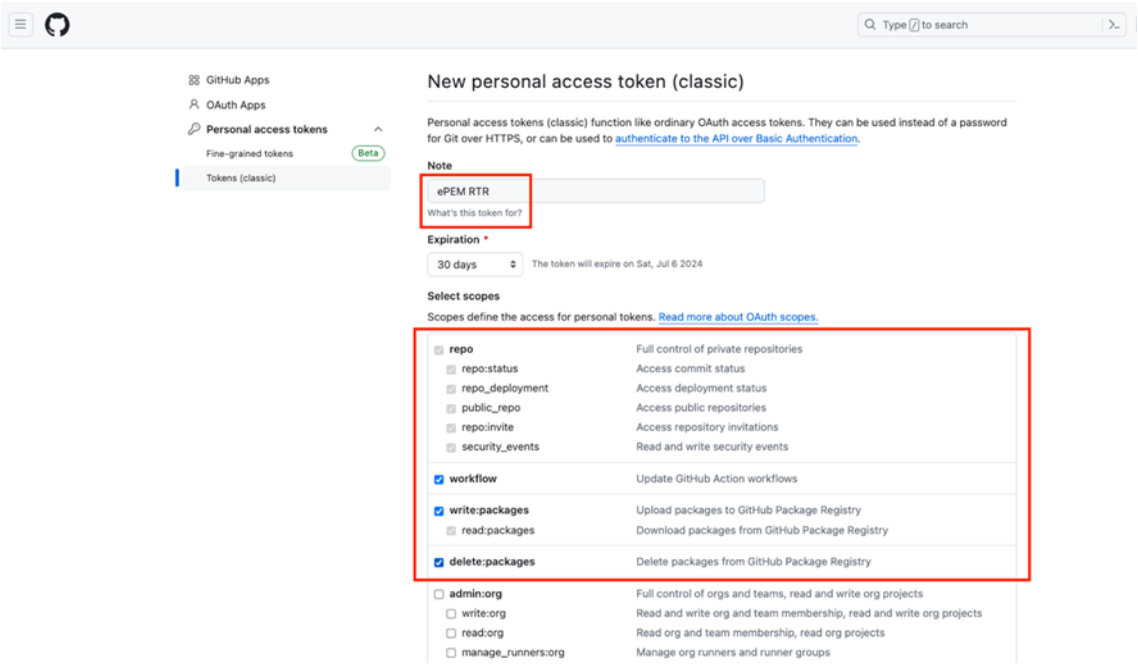
3. Select Tokens (classic) in the Personal access tokens section:



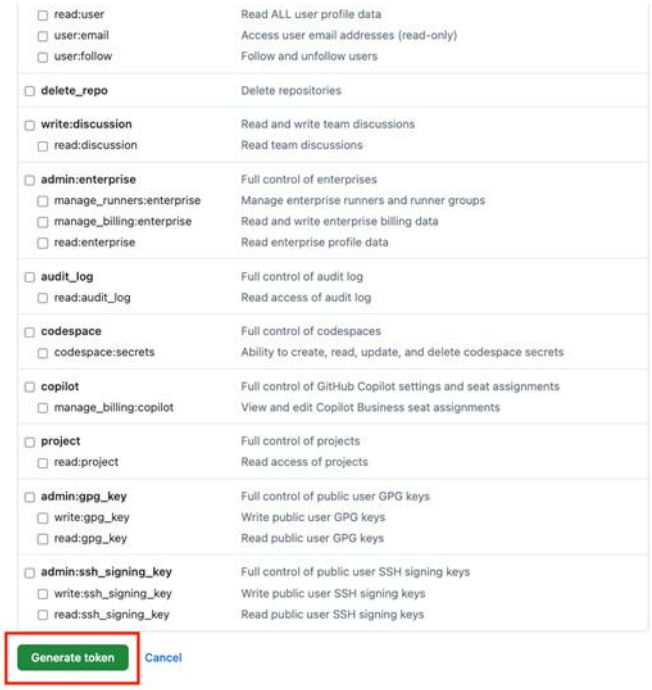
4. Select Generate new token (classic) in the Generate new token section:



5. Add a Note and Select the necessary scopes:



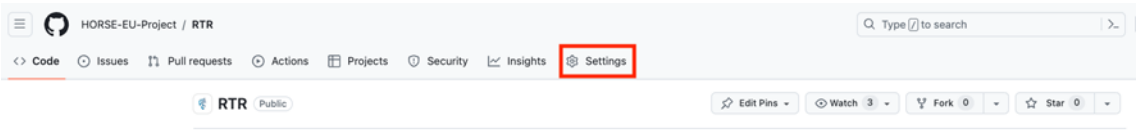
6. Generate token:



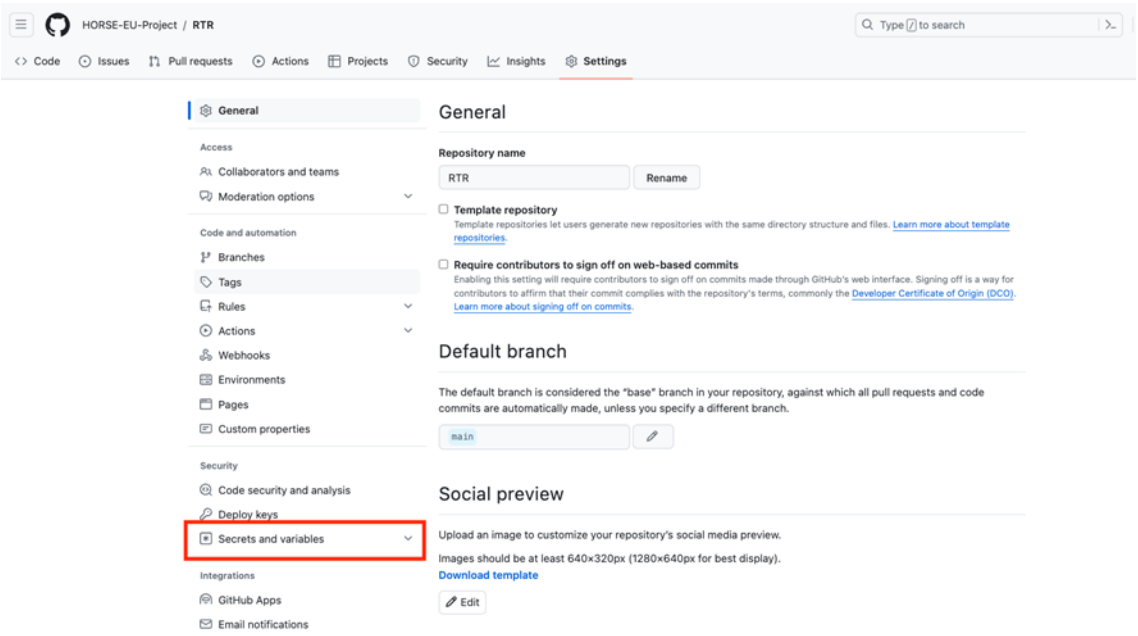
7. After that, the token value will be shown. It is recommended to save it for its use when creating a new repository secret.

6.2 Configure a new repository secret.

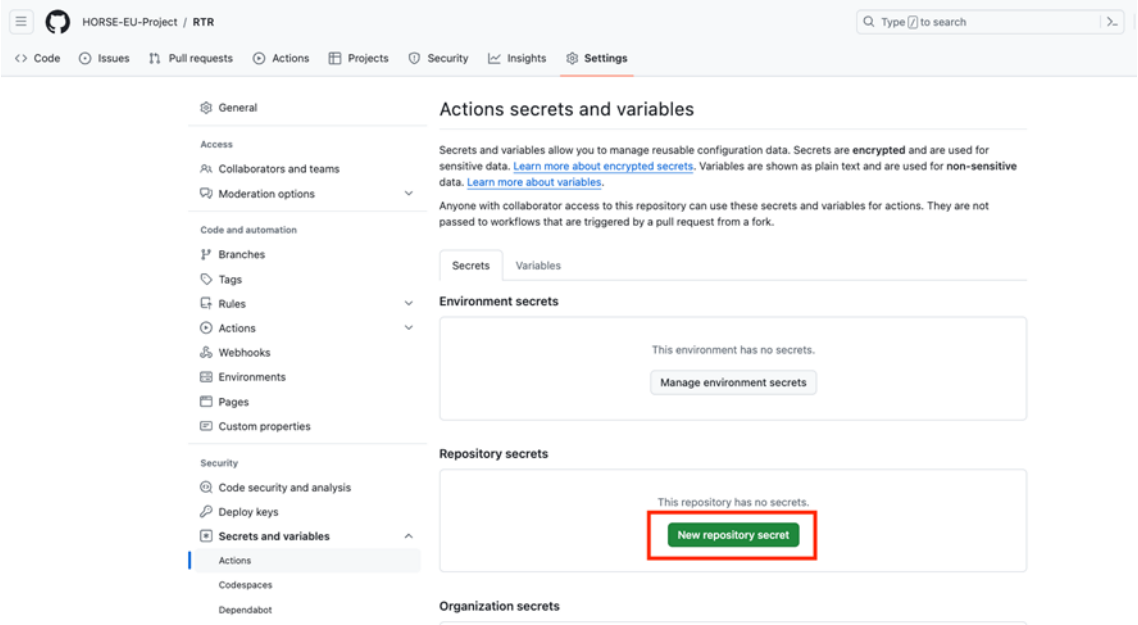
1. Go to the repository Settings:



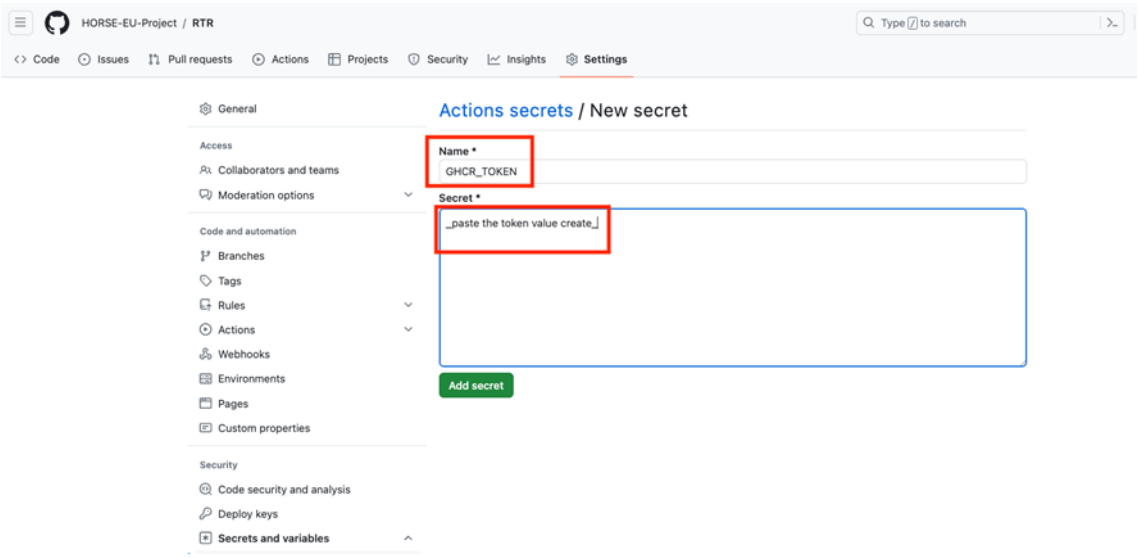
2. Select Actions in the Secrets and variables section:



3. Click on New repository secret:

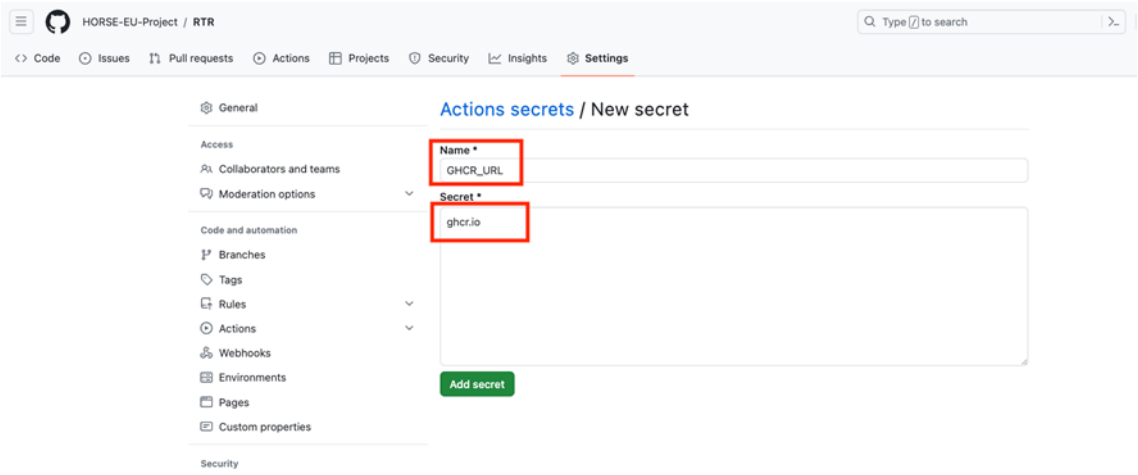


4. Use the desired value for the variable:



6.2.1 GHCR_URL

URL of the GitHub Container Registry. This URL specifies the endpoint of the container registry where the Docker images will be stored. It is used along with the token and username to establish the connection.



6.2.2 GHCR_IMAGE

Name and tag of the Docker image that will be built and published. This secret defines how the Docker image will be tagged during the build process when pushed to the registry.