Deliverable D2.3

Integrated MIRANDA platform

Editor	G. Koutsimpiris (SPH), A. Priovolos (SPH), G. Katevas (SPH), E. Kouletou (SPH)
Contributors	CNR, ONE, LOG, POLITO, NASK, AIT, PLX, UBI
Version	1.1
Date	3rd March 2026
Distribution	PUBLIC (PU)
Classification	UNCLASSIFIED (U)

This project has received funding from the European Union's Horizon Europe Research and Innovation program under grant agreement No. 101168144. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Cybersecurity Competence Centre. Neither the European Union nor the granting authority can be held responsible for them.

Authors

CNR	CONSIGLIO NAZIONALE DELLE RICERCHE
Matteo Repetto, Daniele Canavese	
ONE	ONESOURCE
Luis Cordeiro, Jorge Proença, Artur Dias, Luis Rosa, Miguel Domingues, Ricardo Martins, Wilson Ferreira	
LOG	LOGSTAIL
Ioannis Avdoulas, Georgios Deligiorgis	
POLITO	POLITECNICO DI TORINO
Alessio Sacco, Daniele Bringhenti	
NASK	NAUKOWA I AKADEMICKA SIEC KOMPUTEROWA - PAŃSTWOWY INSTYTUT BADAWCZY
Mateusz Krzysztoń	
AIT	AIT - AUSTRIAN INSTITUTE OF TECHNOLOGY GmbH
Max Landauer, Viktor Beck, Peter Leitmann	
PLX	PLAIXUS
Mike Karamousadakis	
UBI	UBITECH
Sofianna Menesidou	

Reviewers

CNR	CONSIGLIO NAZIONALE DELLE RICERCHE
Daniele Canavese	
AIT	AIT - AUSTRIAN INSTITUTE OF TECHNOLOGY GmbH
Max Landauer	

Copyright and Disclaimer

Funded by
The European Union

The MIRANDA project is funded by the European Union. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Cybersecurity Competence Centre. Neither the European Union nor the granting authority can be held responsible for them.

Version History

Rev. N	Description	Author	Date
0.1	Initial draft	G. Koutsimpiris (SPH)	10/11/2025
0.2	Last Contributions from partners	G. Koutsimpiris (SPH)	10/02/2026
0.3	Draft ready for internal review	G. Koutsimpiris (SPH)	16/02/2026
0.4	Last draft after internal review	G. Koutsimpiris (SPH)	25/02/2026
1.0	Formatting and quality check	M. Repetto (CNR)	26/02/2026
1.1	Final revision	G. Koutsimpiris (SPH), M. Repetto (CNR)	03/03/2026

List of Acronyms

Acronym	Meaning
ABAC	Attribute-Based Access Control
GA	Grant Agreement
CA	Consortium Agreement
CMS	Cloud Management Software
IdM	Identity Management
RBAC	Role-Based Access Control
GNN	Graph Neural Network
SCG	Service Context Graph
CTI	Cyber Threat Intelligence
IoC	Indicators of Compromise
ML	Machine Learning
AI	Artificial Intelligence
LLM	Large Language Model

Executive Summary

The Integrated MIRANDA Platform is a comprehensive cyber-intelligence system combining multiple modular components into a unified, containerized environment. It adopts modern DevOps and CI/CD practices (GitLab, Docker, Kubernetes) to automate building, testing, and deployment of each component, ensuring consistent and secure operation. Core functions include automated data ingestion from OSINT and CTI sources, contextual mapping of network assets, attack-path modeling, anomaly detection in logs, and trust-based analysis to support decision-making.

All components are operating under a unified architecture. Incoming data streams (logs, alerts, external feeds) are ingested through Kafka-based pipelines. The Context Discovery and Attack Modeling modules build a dynamic Service Context Graph (SCG) of active services and compute potential attack paths. Detection modules (e.g. DetectMate) apply machine learning to analyze log data and flag anomalies. External intelligence is integrated via an OWASP/MISP/OpenCTI pipeline that enriches the intelligence picture with known vulnerabilities and threat indicators. The integrated front-end visualizes the context graph, detected threats, and aggregated reports.

Key outcomes include a fully operational CI/CD pipeline with integrated security scanning (static code analysis, dependency vulnerability scanning, and code-quality metrics) and automated deployment. For example, Context Discovery now completes in under 6 seconds for full-use-case environments, demonstrating that performance targets are met. The CTI integration component unifies OWASP, MISP, and OpenCTI data flows to produce actionable threat intelligence and each module's functionality is validated against its requirements through automated tests.

Ongoing improvements are planned for each component, for example audit logging will be added for end-to-end traceability, performance tuning will improve response times, and new features (such as on-demand AI-based report summarization and enhanced federation capabilities) are under development. Because the platform is fully containerized and based on open standards (e.g. OpenC2, CycloneDX), it can incorporate future enhancements seamlessly. In summary, the MIRANDA platform now provides a self-contained, production-ready cyber-intelligence environment that satisfies the project's objectives.

MIRANDA uses a fully automated CI/CD pipeline on GitLab. Every code change is built, tested, and containerized with integrated security checks (static analysis, vulnerability scans) to ensure only vetted releases are deployed. All components run as Docker containers orchestrated by Kubernetes in a VPN-protected environment, with TLS-secured credentials. The infrastructure (cloud VMs with multi-core CPUs and roughly 32–64 GB RAM) hosts GitLab, container registries, and the Kubernetes cluster. Automated pipelines push container images to an internal registry

and deploy them to an internal Kubernetes cluster, achieving reproducible builds and a secure development and deployment environment.

MIRANDA continuously constructs a Service Context Graph (SCG) of all active assets and their relationships (for example, which devices are hosted on which servers and how network segments are connected). A Context Discovery service gathers inventory from cloud APIs, network controllers, and security management systems, and populates the SCG data model. Operators view this context through a web GUI that displays interactive graph and table views of services, devices, and security attributes.

Building on the SCG, the Attack & Threat Modeling module generates attack-path trees. It computes possible sequences of exploits or misconfigurations that an adversary could follow through the network. For example, if a database service has an unpatched vulnerability, the module traces which downstream assets could be affected. This meets requirements for dynamically displaying vulnerabilities and analyzing threat propagation. The combined context and threat modules provide a real-time map of the system and highlight how threats could traverse it. Planned enhancements include adding more “actuators” (data collectors) for additional devices and integrating actuator outputs into the MIRANDA Connector for automated enforcement.

All operational data (system logs, alerts, telemetry) flows through a Kafka-based pipeline for real-time processing. A key component is the Log Parser, which consumes raw log entries and applies predefined templates to extract structured fields (timestamp, source IP, event type, etc.), then republishes the data for analysis.

DetectMate provides automated anomaly detection using machine learning. During an offline training phase, a Graph Neural Network model learns normal patterns of user commands and system events. In production mode, DetectMate continuously ingests new log data and flags any behaviors that deviate from the learned baseline (for example, unexpected commands or traffic patterns). Detected anomalies are published as alerts. This satisfies the requirement for continuous real-time detection of suspicious activity.

MIRANDA enriches its situational awareness with external intelligence from two channels. First, the CTI Integration module (developed by SPH) combines OWASP, MISP, and OpenCTI platforms. An automated process fetches vulnerability data from OWASP, while threat indicators (IoCs) from partners are ingested into MISP. Connectors synchronize MISP with OpenCTI so that events and indicators flow seamlessly between them, ensuring traceability from source to analysis. This unified pipeline yields actionable threat intelligence that enhances situational awareness.

Second, the Taranis-AI component (developed by AIT) collects open-source intelligence (OSINT) from configurable news and social-media sources. Collected items are processed by a series of enrichment bots: a wordlist bot tags security-related terms; an IOC bot extracts known

indicators; an NLP-based tagger identifies entities; a story-clustering bot groups related items; and a summary bot generates concise textual summaries. Analysts can browse and filter the enriched items via the Taranis web interface.

Both intelligence streams feed into the platform's reporting capabilities. Operators can generate structured reports that integrate OSINT summaries, CTI data, and any detected anomalies. The admin interface allows configuration of report templates and attributes. By combining multiple external sources with automated analysis, the system meets requirements for multi-source threat data collection and enables more effective decision-making.

The platform includes mechanisms for evaluating trustworthiness of network entities. A Trust Assessment Framework ingests evidence (such as IDS alerts, scan results, and anomaly scores) and applies policy rules to compute trust levels for users or devices. For example, if a service repeatedly triggers security alerts, its trust score will decrease, possibly triggering countermeasures.

A Trusted Computing Base (TCB) underpins platform security. The TCB provides cryptographic services (AES encryption, RSA/ECDSA signing, secure random generation) and an eBPF-based Linux tracer for system attestation. Critical data flows (e.g. configuration files, context graphs) can be signed and verified to ensure integrity. Planned work will extend attestation coverage and add more monitored attributes.

Crucially, the MIRANDA Connector serves as the execution agent for security actions. It runs as a lightweight Python agent subscribing to the platform's OpenAPI command bus. Validated OpenC2 commands from controllers (e.g. "block IP address") are translated by the Connector into device-specific instructions (e.g. firewall or packet-filter rules). This provides a uniform interface to diverse security functions, fulfilling the system's control requirements. The Connector is implemented for Python 3.10+ and can be deployed from source or as a Docker container.

This report describes an initial integrated MIRANDA platform that meets its design goals. The automated DevOps pipeline ensures each component is built, tested, and deployed with security checks. The assembled system provides end-to-end cyber situational awareness by continuously mapping network context (SCGs), modeling attack paths, detecting anomalies in logs, and incorporating external threat intelligence. Early performance results (e.g. context mapping in seconds) indicate that system requirements are met.

Future work will refine and extend the platform. Audit logging will be added for full end-to-end traceability across workflows, and optimizations will improve performance and responsiveness. Additional intelligence feeds and analytics (such as AI-driven report generation) are under development. Because the design is modular, containerized, and standards-based, new capabilities can be added with minimal disruption. In summary, the integrated MIRANDA

platform now provides a self-contained, production-ready cyber-intelligence solution that consolidates multiple tools into one unified workflow.

Table of Contents

List of Tables

List of Figures

1. INTRODUCTION

This deliverable document the integrated MIRANDA platform, consolidating how the project's diverse technical components are brought together into a coherent, deployable, and testable system. Its scope is twofold:
(a) to describe the common integration backbone (DevSecOps/CI/CD and the shared development & testing environment) that enables partners to build, validate, and deploy components consistently.
(b) to provide component-level technical status for the integrated platform, including each module's purpose, interfaces, deployment approach, validation/testing evidence, and planned next steps.

1.2 Purpose of the document

The motivation is to demonstrate that the platform is not just a collection of independent prototypes, but an operationally integrated ecosystem that supports rapid, reliable and secure delivery across the consortium. Concretely, the document emphasizes standardized CI/CD pipelines, automated security checks (e.g., static analysis and vulnerability scanning), reproducible containerized deployments, and controlled access to shared infrastructure—so that integration can proceed continuously, risks can be surfaced early, and delivered software artifacts can be trusted and traced.

1.3 Document Structure

The document is organized as follows:

- Introduction: states the purpose of the deliverable, its relationship to other work packages/deliverables/activities and explains how the content is organized.
- Integration platform – CI/CD platform: presents the project's DevOps/CI/CD approach and methodology, the security practices applied in the development & testing environment (DevSecOps gates, signing, runtime monitoring), the toolchain used, and the underlying infrastructure and access controls (e.g., VPN access and Kubernetes RBAC).
- Components information: the largest part of the deliverable, providing a consistent per-component view (prototype description, communication interfaces, deployment details, individual component testing, and future updates) for the integrated building blocks of the platform—covering integration/control components (e.g., Connector), context discovery and attack/threat modelling, data handling and ingestion pipelines, prediction, threat-feed ingestion, sandboxing, GUI/visualisation, trust and trusted computing base elements, automated detection and response, threat hunting, and actionable CTI/OSINT capabilities.
- Conclusions: summarizes the integration status and outlines forward-looking work to mature and extend the integrated platform.

2. INTEGRATION PLATFORM - CI/CD PLATFORM

2.1 DevOps and CI/CD practices

The MIRANDA project adopts DevOps principles and Continuous Integration/Continuous Deployment (CI/CD) practices to ensure rapid, reliable, and secure delivery of software components across all work packages. The CI/CD platform serves as the central integration backbone, enabling automated building, testing, security scanning, and deployment of project components.

The platform integrates security checks early in the development lifecycle rather than as a final gate. This includes Static Application Security Testing (SAST), dependency vulnerability scanning, container image scanning, and software signing—all executed automatically within the CI pipeline. Application deployments are managed declaratively through Git repositories following GitOps principles, with Argo CD handling synchronisation between the desired state defined in Git and the actual state running in the Kubernetes cluster. Container images are tagged by commit SHA and cryptographically signed to ensure traceability and prevent tampering. Pipelines enforce quality and security thresholds, blocking builds with high or critical vulnerabilities before artifacts can be deployed. Runtime security monitoring through Falco provides continuous visibility into application behaviour and potential threats throughout the development and testing lifecycle.

2.2 Methodology

The MIRANDA CI/CD platform implements a structured development methodology with each MIRANDA component repository adhering to a standardised pipeline structure that is adapted to accommodate the specific requirements of its technology stack and dependencies (e.g., Java, Python). This standardisation ensures consistency across the project while maintaining the flexibility necessary to address technology-specific build and testing requirements.

The automated pipeline orchestrates a sequential validation workflow that subjects every code change to comprehensive quality and security analysis before integration. Initially, static code analysis tools including SonarQube examine source code for quality issues, bugs, and security vulnerabilities, while dependency scanners such as OWASP Dependency-Check and Snyk identify known vulnerabilities and license compliance issues within project dependencies. Following successful static analysis, the pipeline proceeds to construct container images. Each constructed image receives a cryptographic signature, establishing verifiable integrity for deployment artifacts. Subsequently, Trivy performs comprehensive vulnerability scanning of the constructed container image, examining both operating system packages and embedded application dependencies. The pipeline can be easily configured to halt execution and prevent deployment if HIGH or CRITICAL severity vulnerabilities are detected during this scanning phase.

Upon successful completion of all validation stages, the pipeline initiates deployment through Argo CD, which operates according to GitOps principles by continuously synchronizing the desired application state defined in Git repositories with the actual running state in the

Kubernetes cluster. This approach provides inherent rollback capabilities, as reverting to a previous application version requires only reverting the corresponding Git commit. At the conclusion of each pipeline execution, aggregated security and quality metrics are collected and presented in consolidated reports, providing development teams and project stakeholders with visibility into code quality trends and security posture.

Repository protection rules should be defined to prevent direct commits to main, requiring that modifications proceed through merge requests that undergo both peer review and automated pipeline validation. This dual-gate approach combines human expertise and automated analysis for quality assurance. The platform supports semantic versioning for release management, with Git tags capable of triggering specialized pipeline workflows for version-specific build and deployment procedures. Kubernetes rolling update mechanisms enable deployments during routine updates, while the GitOps model facilitates rapid rollback through declarative Git operations when issues are identified in deployed versions

2.3 Security practices for the CI/CD platform and Development & Testing Environment

Sensitive credentials including API tokens, signing keys, and registry passwords are stored as protected and masked CI/CD variables in GitLab, scoped to specific branches and environments to minimize exposure. CI service accounts and tokens operate under the principle of least privilege, with permissions limited to the minimum required for their function. For example, the Argo CD CI user possesses only sync permissions for specific applications rather than cluster-wide administrative access. All external communications with image registries and artifact repositories enforce TLS verification to prevent man-in-the-middle attacks.

The platform implements verification mechanisms to ensure that deployed container images originate from trusted sources and remain unmodified during storage and distribution. Cryptographic signatures created during the build process are verified before deployment to confirm that images have not been tampered with and originate from authorized build processes. This verification mechanism prevents the deployment of unauthorized or modified container images that could introduce security vulnerabilities or malicious code.

In terms of runtime, Falco monitors the system calls and kernel events to detect anomalous runtime behavior including privilege escalations, unexpected network activity, and file access violations. Kubernetes network policies restrict inter-pod communication to authorized paths only, implementing network segmentation at the workload level.

2.4 CI/CD tools

The CI/CD platform is built on a set of open-source and industry-standard tools.

2.4.1. CI Server

GitLab CI/CD serves as the CI server and orchestration engine for the MIRANDA platform. Kubernetes-based GitLab Runners execute pipeline jobs using the Kubernetes executor for scalable, isolated execution. Pipelines are defined in specialised files within each component

repository, supporting multi-stage workflows, artifact management, caching, and conditional logic. GitLab integrates Git repository hosting, merge request workflows, issue tracking, and container registry functionality in a unified platform accessible to all project partners.

2.4.2. Docker / Kubernetes

Kubernetes serves as the container orchestration platform hosting both CI/CD tooling and deployed MIRANDA components across development, testing, and integration environments. Tools are deployed in dedicated namespaces to isolate concerns and apply fine-grained role-based access control. Container images are currently built using Buildah, a rootless and daemonless OCI-compliant image builder that operates without requiring Docker daemon privileges.

2.4.3. Image/Artifact Registry

GitLab Container Registry stores and distributes container images with native GitLab CI/CD integration using project or group deploy tokens for authentication. Images follow the naming convention `registry.gitlab.com/<group>/<project>:<tag>`, with tag typically being commit SHAs for immutability or semantic versions for releases. Configurable cleanup policies can be set to manage storage by automatically removing old or untagged images.

2.4.4. Git

GitLab provides Git repository hosting and version control for all MIRANDA components. Each component repository maintains a standardised structure including source code, Dockerfiles, Helm charts, and pipeline configuration. Protected branches enforce merge request workflows with mandatory pipeline validation before integration. Merge requests serve as code review gates with automatic pipeline execution, while Git tags mark releases and trigger version-specific deployment pipelines.

2.4.5. Monitoring

Currently, Falco is leveraged to provide runtime security monitoring and threat detection for the Kubernetes cluster. Deployed across all cluster nodes, Falco monitors system calls, file access, network activity, and process execution to detect suspicious behavior patterns in real-time. Configured rules alert on privilege escalations, unauthorized file modifications, unexpected network connections, and execution of sensitive binaries. Falco can forward alerts to centralized logging systems or expose metrics for integration with monitoring tools.

2.4.6. SAST

The platform integrates multiple security analysis tools to provide comprehensive vulnerability detection:

- SonarQube performs continuous code quality and security analysis, detecting code bugs and security vulnerabilities across multiple programming languages. GitLab pipelines

invoke a scanner to analyse code and upload results to the SonarQube server, where quality gates can block merge requests based on configurable thresholds. Pipelines can fetch key metrics via the SonarQube API for consolidated reporting.

- Snyk and OWASP Dependency-Check scan application dependencies for known vulnerabilities by cross-referencing databases including Snyk's vulnerability database and the National Vulnerability Database (NVD). These tools identify vulnerable dependencies in package managers (e.g. npm, pip, Maven), generating reports with severity ratings, CVSS scores, CVE identifiers, and remediation guidance.
- Trivy performs container image vulnerability scanning as a critical deployment gate. Executed after image builds, Trivy scans both operating system packages and application dependencies within container images, supporting multiple Linux distributions. Pipelines are configured to fail automatically when HIGH or CRITICAL severity vulnerabilities are detected, preventing vulnerable images from deployment.

2.5 Infrastructure

The ONE testbed represents a research infrastructure designed to enable developing, testing, and validating emerging technologies. The testbed consists of a hybrid set of bare-metal computing nodes and a virtualized infrastructure. The latter uses Proxmox VE as a KVM-based hypervisor of a fleet of virtual machines hosted within a server rack. These servers are interconnected via a high-speed fibre channel to a centralized storage switch, which links to a NAS system equipped with SSD and HDD drives. This ensures high-performance and scalable storage access that is suitable for persistent workloads and large-scale data processing scenarios.

On top of the virtualized layer, the testbed features a fully operational multi-node Kubernetes cluster. This cluster serves as the primary platform for deploying and managing containerized applications, with built-in support for persistent storage via NFS, traffic management and service exposure through NGINX ingress controllers (HTTP/HTTPS), facilitated by MetalLB for load balancing and exposing additional types of network traffic (e.g. UDP). Moreover, the ONE testbed also integrates Cloud Infrastructure Management tools such as OpenStack and Rancher, which enable dynamic provisioning of virtual machines and Kubernetes clusters on-demand and Identity and Access Management (IAM) tools such as Keycloak to restrict access and manage authorisation. Furthermore, the ONE testbed also features 5G-specific technology, such as Open5GS, OAIBox and OpenSlice, to support the testing of 5G-enabled scenarios.

ONE testbed comprises various NVIDIA GPUs, enabling real-time core and edge processing for inference GPU-intensive tasks. Furthermore, with support for satellite connectivity through systems such as Starlink, the testbed can emulate distributed environments, adding a layer of realism to mobility and edge use case testing. To connect and control external user equipment, the testbed supports many devices, including smart sensors, cameras, gateways, VR/AR headsets, and more. This ensures compatibility with real-world IoT applications, spanning smart cities, Industry 4.0, healthcare, and immersive technologies.

The remote access to the testbed is securely managed through OpenVPN, allowing remote users to connect to their isolated environments and interact with the deployed services.

2.5.1. Hardware

The virtualized infrastructure hosted within OneSource premises (described in Table 1) can be modified and extended accordingly to comply with additional requirements and constraints (e.g., OS, service exposure, integration with different types of technologies).

Table 1. Cloud servers' hardware specifications

Infrastructure Type	Description
Physical Server	2x Dell PowerEdge M520 with an Intel(R) Xeon(R) CPU E5-2430L (24 cores), 128 GB RAM
Physical Server	Dell PowerEdge M630 with an Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz (32 cores), 256GB RAM
Physical Server	HPE ProLiant DL630P Gen8 with an Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz (32 cores), 196 RAM
GPU	2x L4 GPU with 24 GB VRAM and bandwidth of 300GB/s
GPU	RTX 4070 with 12 GB VRAM and a bandwidth of 500GB/s
Storage	The servers described above are mounted in a V-RTX unit connected directly to HDD local storage (12TB) through fiber-channel. Additionally, the servers are connected to a storage switch through fiber-channel, which is connected to a shared SSD storage unit (6TB).
Hypervisor Technology	The physical infrastructure relies on Proxmox virtualization, which hosts the MIRANDA testbed (the virtualized infrastructure can be extended to accommodate different requirements and architectures, including Operative System. GPU support, cloud-native approaches).
MIRANDA Virtualized Testbed	1 x VM (hosts CNR specific infrastructure components such as dedicated NFS-Server and the Actuator) • 4 vCPUs • 8 GB RAM • 64 GB storage
MIRANDA Virtualized Testbed	3 x VM (each VM corresponds to a Kubernetes Cluster node, where access is restricted by namespace for the different MIRANDA partner activities, while also hosting the stable versions of the MIRANDA platform components and integration activities) • 6 vCPUs • 12 GB RAM • 64 GB storage The MIRANDA testbed currently includes tools for CI/CD and DevSecOps integration (i.e., ArgoCD, SonarQube, Gitlab Agents/Runners). The testbed also supports different storage types (i.e., NFS, S3 Buckets) and a Kafka cluster for communication within the MIRANDA platform.

	The testbed can be extended throughout the lifecycle of the project, following the timeline and requirements of the MIRANDA platform.

2.5.2. Network, Security and Access

Access to the MIRANDA infrastructure is governed by a role-based permission model aligned with each partner's responsibilities within the project. Permissions are granted according to the specific tasks and roles of individual partners. As a result, certain activities and workloads, such as penetration testing, may require additional safeguards and controlled deployment procedures within the infrastructure.

Access to the MIRANDA platform is protected through a multi-layered security approach.

First access layer – VPN

All user access to platform services is routed through a Virtual Private Network (VPN) based on OpenVPN. VPN access is provisioned on an individual basis upon request and is strictly assigned to a specific user. Multi-Factor Authentication (MFA) is enforced to enhance security. MIRANDA platform services are only exposed to authenticated VPN users.

Second access layer – Kubernetes RBAC

Within the Kubernetes cluster, access is further restricted using Role-Based Access Control (RBAC). Partners are granted permissions strictly according to their operational needs. Direct access to Kubernetes nodes is not permitted for any partner.

Each partner is assigned a dedicated Kubernetes namespace in which they have full administrative control over their own resources. An exception applies to namespaces created and managed by the Sandbox platform. These namespaces operate under distinct security policies and governance rules to ensure strong isolation from the rest of the cluster and to support controlled experimentation and testing activities.

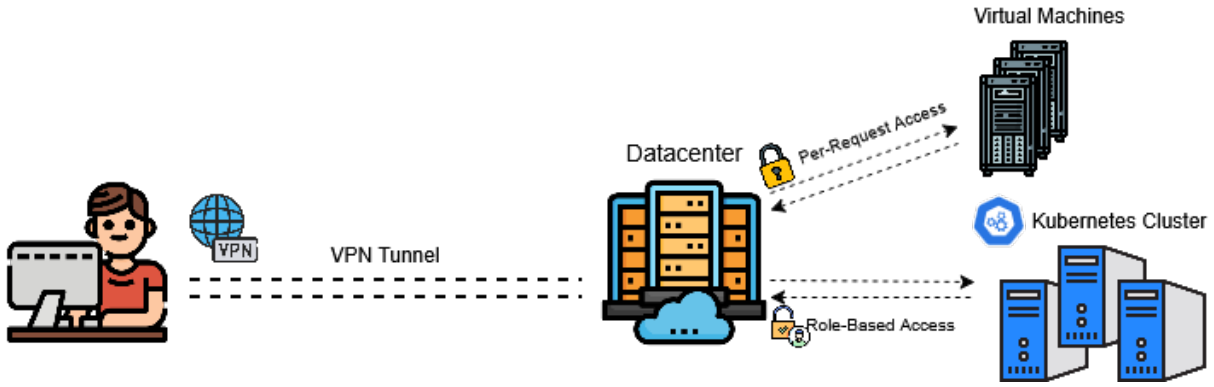

Figure 1. Network, Security and Access

3. COMPONENTS INFORMATION

3.1 MIRANDA Connector

3.1.1. Prototype Description

The MIRANDA Connector is designed as the local agent responsible to run OpenC2 Actuator Managers [1] for all supported Cyber-Security Functions (CSF), according to the original design in D2.1. Its architecture is shown in and has not changed since the initial design.

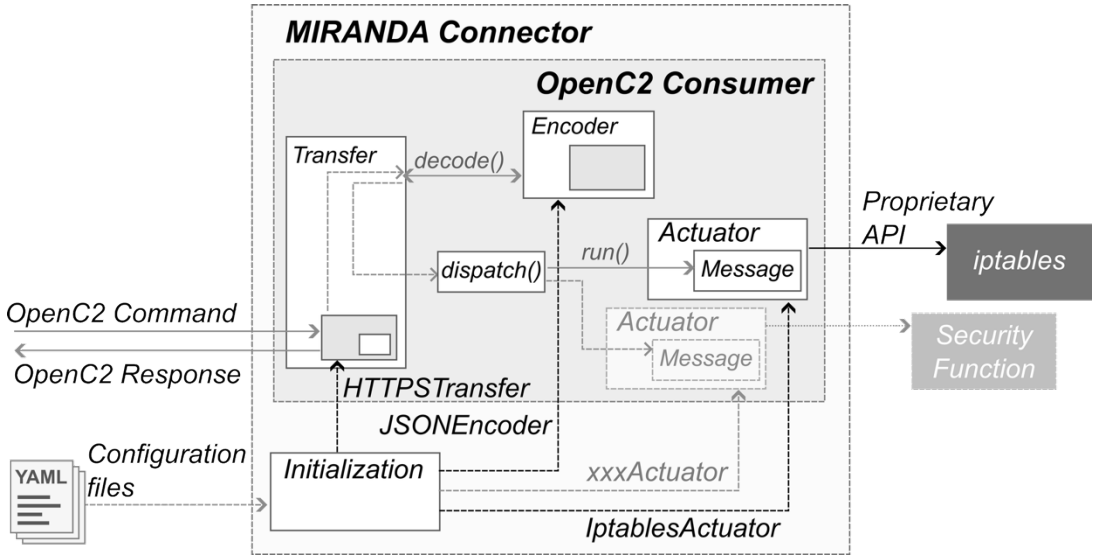

Figure 2. Internal design of the MIRANDA Connector

The set of Actuator Managers currently available is listed in the following table:

Table 2. Actuator Managers

Name	Profile	Security Function	Status
Iptables	slpf	Iptables CLI	Ready
OpenStack SG	slpf	OpenStack Security Groups	Ready
Kubernetes NP	slpf	Kubernetes Network Policies	Ready
Nprobe	x-nfm	nprobe	Ready
PacketBeat	x-nfm	PacketBeat	Ready
CLI	x-rcli	Remote linux shell	Ready
Filebeat	x-fclm	FileBeat	Ready

The MIRANDA Connector takes a set of configuration files and uses them to start the corresponding Actuator Managers. It implements the function of a generic OpenC2 Consumer which receives OpenC2 commands and dispatches them to the intended Actuator [1].

The Connector fulfils the following requirements:

- FR-A-1.11 - Controlling security functions: it implements a local control agent that allows remote dashboard to control security functions with a uniform interface.
- FR-A-1.27 – Packet filtering: the Connector implements the StateLess Packet Filter profile (SLPF) for common firewall engines deployed in the use cases (iptables and integrated filtering capabilities in CMS).
- FR-B-1.2 – Interface to security functions: it provides a uniform control interface to the most common set of security functions, while adopting a standard language that allow extensions to any other kind of function.
- FR-B-1.8 – Security properties: The current list of supported profiles by the Connector includes: firewalling (slpf), logging (x-flcm), network flows (x-nfm), remote cli (x-rcli). The latter can be used as generic way to control any kind of security function that does not have its own profile definition yet.
- NON-FR-B-1.24 - Latency of commands to security functions: the exchange of OpenC2 commands with the Connector takes no more than 100 ms in case of low network latency. However, the overall delay may take up to 2-4 seconds due to the response time of the security function (this falls outside the scope and design of the Connector).

3.1.2. Communication interfaces

The MIRANDA Connector implements the OpenC2 language specification over HTTP and MQTT.

The OpenC2 specification [2] defines the common syntax and serialization formats for both the request and body of messages. A Command follows typical language patterns, with a subject (Actuator), a verb (Action), an object (Target), and complements (Arguments). It contains:

- Action: the instruction, task or activity to be performed (e.g., start, stop, locate, set, update, create).
- Target: the object affected by the execution of the instruction (e.g., a file, an email, a network connection, a device).
- Arguments: additional information on how the command is performed (e.g., time interval, duration, periodicity).
- Actuator: the entity that executes the action (e.g., firewall, Intrusion Detection System).

Both Targets and Actuators can be specified with different levels of granularity, namely they may identify either a specific object, or a list or a group of objects. The Language Specification defines a standard set of actions and a baseline collection of targets. This means that the set of actions must be the same for every implementation and no additions are allowed. Instead, new targets can be defined for specific cybersecurity functions, to better account for their specific purpose.

A Response contains the following elements:

- Status: An integer that represents the exit status of the command execution (mostly following the HTTP response codes enumeration).

- Status text: A human-readable description of the status.
- Results: a list of key/value pairs produced by the execution of the command.

Each element in OpenC2 Commands and Responses is defined in terms of primitive types (e.g., binary, integer, string) and derived structures (e.g., array, map, enumerated). Structures are recursively used and coupled with semantic constraints to derive typical data objects, as IP/MAC addresses, emails, domain names, dates and times, digests, etc. The semantics of Request and Response bodies consumed/generated by the Connector are described either in standard profiles (e.g., SLPF [5]), or by extensions contributed by the project.[1]

According to the OpenC2 design, the MIRANDA Connector is able to manage multiple serialization formats, including the mandatory options of JSON for HTTP and JSON/CBOR for MQTT.

The following table shows the HTTP method and endpoint implemented by the Connector, according to the specification [3]:

Table 3. HTTP method and endpoint implemented by the Connector

#	Method	REST Endpoint	Description	Request Body	Response Body

1	POST	/.well-known/openc2	Common OpenC2 endpoint exposed to reach any Actuator run by the Connector	{
				"headers": {
				"request_id": "...",
				"created": 0000,
				"from": "...",
				"to": ["...", "..."]
				},
				"body": {
				"openc2": {
				"action": "...",
				"target": "...",
				"args": "...",
				"actuator": "..."
				}}}
					{
					"headers": {
					"request_id": "...",
					"created": 0000,
					"from": "...",
					"to": ["...", "..."]
					},
					"body": {
					"openc2": {
					"status": "<code>",
					"status_text": "text",
					"results": "..."
					}}}

Additionally, the Connector expects the following headers to be present in commands:

- Content-type: application/openc2+json;version=1.0
- Accept: application/openc2+json;version=1.0

[1] New profiles are part of the otupy framework and described in its documentation: https://otupy.readthedocs.io/.

- Data: … (optional).

and insert the following headers in responses:

- Content-type: application/openc2+json;version=1.0
- Data: …

The following table shows the MQTT topics the Connector listens to and publishes to, and the message structure used (same as for HTTP), according to the specification [4]:

Table 4. MQTT topics the Connector listens to and publishes

Topic	Description	Message Sample

oc2/cmd/all
oc2/cmd/ap/[actuator profile]
oc2/cmd/device/[device id]
	The Connector waits for commands on these topics.	{
		"headers": {
		"request_id": "…",
		"created": 0000,
		"from": "…",
		"to": ["…", "…"]
		},
		"body": {
		"openc2": {
		"action": "…",
		"target": "…",
		"args": "…",
		"actuator": "…"
		}}}

oc2/rsp
oc2/rsp/[producer id]
	The Connector sends responses on these topics	{
		"headers": {
		"request_id": "…",
		"created": 0000,
		"from": "…",
		"to": ["…", "…"]
		},
		"body": {
		"openc2": {
		"status": "<code>",
		"status_text": "text",
		"results": "…"
		}}}

The Connector receives and conveys OpenC2 messages in the payload of MQTT PUBLISH control packets, with the following properties:

- Payload Format Indicator [Property 0x01]: 0x00 for binary data; 0x01 for UTF-8 encoded strings;
- Content Type [Property 0x03]: "application/openc2";
- User Property [Property 0x26]: two User Properties (UTF-8 string pairs) as follows:
  - "msgType": "req" (request), "rsp" (response), or "ntf" (notification)
  - "encoding": "json" or "cbor"

The current Connector implementation uses the following MQTT parameters recommended by the specification:

- Connection:
  - Clean Session: 0
  - Will Flag: 0
  - Will QoS: 0
  - Will Retain: 0
  - VH Session Expiry: 60
- Subscription:
  - Maximum QoS: 2
  - No Local: 1 (true)
  - Retain as Published: 1
  - Retain Handling: 0

3.1.3. Deployment details

The MIRANDA Connector is a Python application that can be easily run either by installing the component from the PyPi repository or by deploying a container.

From the **PyPi repository**, it can be installed (in a virtual environment) by running:

```
% pip install otupy
```

Then run the Connector application (see below for the configuration files):

```
% python3 -m otupy.apps.connector.connector [ -c | --config <config file>]
```

If no configuration file is explicitly provided on the command line, the default *connector.yaml* will be looked for in the same directory where the command is run.

Alternatively, the **docker image** already include the command to run the Connector. The Dockerfile is really simple, because it builds on a common otupy docker image:

```
ARG VERSION="latest"

FROM ghcr.io/mattereppe/otupy:$VERSION

RUN cp -a /opt/otupy-venv/lib/python3*/site-packages/otupy/apps/connector/connector.py .

CMD ["python3", "connector.py", "-c", "/config/connector.yaml"]
```

Docker images for the Connector are automatically built for any new release, and available from the main otupy repository. To build a custom version for a specific version:

```
% docker build -t connector --build-arg VERSION="x.y.z"
```

To deploy the Connector from the images available from the repository:

- put all configuration files in a local folder (e.g., myfolder) and mount it under the /config folder of the container;
- expose the internal port (set in the configuration files, e.g.: 8080) to the desired port available to external components (e.g.: 8080).

```
% docker run -v ./myconfig:/config -p 8080:8080 --name connector -it ghcr.io/mattereppe/connector:latest
```

Docker compose files and Kubernetes manifests can be created according to the previous command line.

Concerning the **configuration**, multiple yaml files must be created:

- a global configuration file for the connector, which should be named "connector.yaml" for the docker image;
- a specific configuration file for each Actuator instance that will be activated by the Connector, collected inside a common folder.

The global configuration file has the following structure:

```
id: connector@mirandaproject.eu
consumer:
   host: '127.0.0.1'
   port: 8080
   endpoint: "/.well-known/openc2"
   encoding: "json"
   transfer: "http"
configs: "configs"

logger:
   <Logger configuration>
```

where:

- `id`: OpenC2 identifier for the Connector
- `consumer`: group of configuration keys for the OpenC2 Consumer communication stack:
  - host: IP address to listen to (also reported when necessary to the Producer) (default: 127.0.0.1)
  - `port`: TCP port to bind to (default: 80)
  - `endpoint`: default to "/.well-known/openc2" and should not be changed
  - `encoding`: Serialization format; allowed values: "json", "cbor", "yaml", "xml" (default to json)
  - `transfer`: Transfer protocol (default to https)
- `configs`: Config folder with actuator-specific configuration files
- `logger`: Logging framework configuration. See the Logger documentation (defaults to logging INFO messages on the console)

Actuator-specific configuration files have the following general structure:

```
<actuator-id>:
  actuator: "actuator_name"
  profile: "slpf"
  owner: "acme"
  specifiers:
    …
  auth:
    …
```

```
  config:
    cacert: "/etc/ssl/certs/acme.crt"
```

where:

- `<actuator_id>` is an identifier of the instance (custom selection, do not refer to any internal data)
  - `actuator`: the name of the actuator (used to register it in the internal actuator list); current allowed values: ….
  - `profile`: name of the registered profile (currently: "slpf", "x-ctxd", "x-rcli", "x-nfm", "x-fclm")
  - `owner`: owner of the security function (informative only)
  - `specifiers`: OpenC2 identification of the actuator instance (depends on the specific actuator profile)
  - `auth`: authentication information needed to control the security function (depends on the specific function implementation)
  - `config`: actuator-specific configuration (e.g., custom CA files, options, etc.)
- additional <key, value> pairs depending on the specific profile/actuator

The syntax for each specific actuator implementation can be found in the documentation of the source code.

3.1.4. Individual component testing

Table 5. Component Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-Connector-01	Check the correctness of the OpenC2 interface.	Achieved
TC-Connector-02	Check the correct functioning of SLPF actuators.	Achieved
TC-Connector-03	Check the correct functioning of MIRANDA actuators	Achieved

Table 6. Test Case-Connector-01

Test Case ID	**TC-Connector-01**	**Component**	Connector
Description	Check the Connector accepts good json samples and rejects bad json samples. Check full HTTP and OpenC2 json data against third-party schemas.		
Tested by	CNR (component owner)		
Associated Requirements	FR-A-1.11, FR-B-1.2, NON-FR-B-1.24		
Pre-condition(s)	Connector implementation and mock-up actuator.		
Test steps			

1	Create otupy objects from valid/invalid data
2	Deserialize good json commands/responses in otupy libraries
3	Deserialize bad json commands/responses in otupy libraries
4	Serialize good json samples in otupy libraries
5	Serialize/deserialize good/bad samples between two otupy agents (generic Producer, Connector) (bad data samples are artificially injected in the communication)
Input data	Valid/Invalid parameters for Targets, Arguments, Actuators: https://github.com/mattereppe/otupy/tree/jsonvalidation/validation/otupy/test_types. Good/Bad OpenC2 json samples from the following repository: https://github.com/bberliner/openc2-json-schema.
Results	The otupy framework used to implement the Connector passed more tests than other implementations [7]. Failed tests are due to questionable interpretation of the standard (classification of good/bad samples from the third party), as explained in detail in [7].
KPIs	Target -> 100% good samples accepted, 100% bad samples rejected, <100 ms processing latency | Measured -> 99.89% (Step 1), 98.40% (Step 2), 90% (Step 3), 100% (Step 4), 99.28% (Step 5), 2/35 ms processing latency (HTTP/MQTT)
Test Case Result	Pass

Table 7. TC-Connector-02

Test Case ID	TC-Connector-02	Component	SLPF Actuators
Description	Verify only valid commands are sent/received by SLPF actuators. Verify rules are correctly enforced by the different actuators (iptables, OpenStack Security Groups, Kubernetes Network Policies).		
Tested by	CNR (component owner)		
Associated Requirements	FR-A-1.27, NON-FR-B-1.24		
Pre-condition(s)	Connector implementation; SLPF profile and actuators. Two virtual machines and 2 Kubernetes pods. Hping3 packet generator, tcpdump to monitor network packets.		
Test steps			
1	Send valid and invalid slpf commands, and verify HTTP 501 error code (Not implemented) is received for invalid patterns.		
2	Send traffic ICMP/TCP/UDP from hping3 on Virtual Machine 1 and collected packets with tcpdump on Virtual Machine 2. Send commands to the iptables actuator to: a) Alternatively		

	allow/deny ingress traffic on Virtual Machine 2 for the hping3 traffic; b) Alternatively allow/deny egress traffic on Virtual Machine 1 for the hping3 traffic.
3	Send traffic ICMP/TCP/UDP from hping3 on Virtual Machine 1 and collected packets with tcpdump on Virtual Machine 2 in OpenStack. Send commands to the Security Groups actuator to: a) Alternatively allow/deny ingress traffic on Virtual Machine 2 for the hping3 traffic; b) Alternatively allow/deny egress traffic on Virtual Machine 1 for the hping3 traffic.
4	Send traffic TCP/UDP from hping3 on Pod 1 and collected packets with tcpdump on Pod 2 in Kubernetes. Send commands to the Network Policies actuator to: a) Alternatively allow/deny ingress traffic on Pod 2 for the hping3 traffic; b) Alternatively allow/deny egress traffic on Pod 1 for the hping3 traffic.
5	Use the same commands of Step 1 to measure the average response latency.
Input data	Valid/Invalid slpf commands. Sequence of slpf commands to add/remove rule.
Results	The SLPF implementation correctly distinguishes between valid and invalid commands. It enforces rules with a few seconds delay, depending on the specific actuator [8].
KPIs	Target -> 100% good samples accepted, 100% bad samples rejected, 100% rules added/removed, < 2s overall latency | Measured -> 100% good samples accepted, 100% bad samples rejected, 100% rules added/removed, 0.8s/2s/0.5s average latency (iptables, Security Groups, Network Policies).
Test Case Result	Pass

Table 8. Test Case-Connector-03

Test Case ID	TC-Connector-03	Component	RCLI/NFM/FCLM Actuators
Description	Check the functional correctness of the 3 profiles and 4 actuators developed in MIRANDA (rcli, nfm, fclm).		
Tested by	CNR (component owner)		
Associated Requirements	FR-B-1.8, NON-FR-B-1.24		
Pre-condition(s)	RCLI/NFM/FCLM profiles designed and implemented. Nprobe, PacketBeat, FileBeat, linux shell actuators implemented.		
Test steps			
1	Serialize/deserialize valid and invalid commands for the 3 profiles		
2	Measure the average time to process commands		
Input data	Valid/Invalid command samples for the 3 profiles		

Results	The 4 actuators complete correctly the command. The average overall latency is much larger than expected, due to the execution of some commands (e.g., upload of file, starting application). This can be accepted, since the internal processing latency of the actuator itself is far below the target threshold (the overall performance may be improved with asynchronous operation). The conformance with good/bad samples is rather below the target threshold, but it can be accepted since failures concern end cases (and the evaluation set is unbalanced).
KPIs	Target -> 100% passed (good samples accepted, bad samples rejected), <100 ms processing latency | Measured -> 86.88% RCLI, 78.08% NFM, 88.68% FLCM (Step 1), 277 ms RCLI, 1.52s NFM, 1s FLCM (Step 2),
Test Case Result	Pass

3.1.5. Next steps and future development updates

As next steps, a few missing actuators will be completed that are necessary to support the Use Cases, as listed in the following table:

Table 9. Actuators that are necessary to support the Use Cases

Name	Profile	Security Function	Status
Azure	slpf	Azure firewall configuration	Untested
Azure Flux	slpf	Azure firewall configuration with Flux	Planned
eBPF	x-ebpf	eBPF kernel and userspace programs	Ongoing
Fprobe	x-nfm	fprobe	Ongoing

It is also planned to provide a REST API for the dynamic configuration of the Connector, namely activation/deactivation of actuator instances.
Additionally, RBAC/ABAC control access mechanisms will be integrated with an identity management framework to manage cross-domain operation in multi-ownership environments.

3.2 Context Discovery

3.2.1. Prototype Description

The Context Discovery component creates an inventory of run-time digital assets and their relationships, which is the basis to identify threats and to understand the potential propagation of attacks. It is made of two components (see Figure 3):

- specific context actuators designed to be run by the **MIRANDA Connector**, which are responsible to report the description of digital components (e.g., by querying CMS to discovery deployed resources);

- a **Discovery** application that recursively queries context actuators based on the relationships between services, to build a map of the overall dependencies.

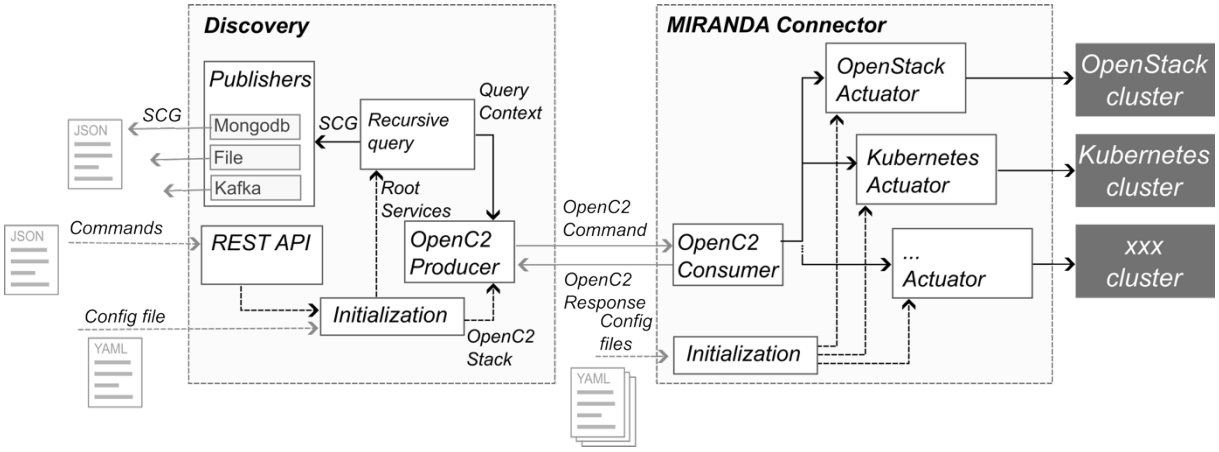

Figure 3. Context Discovery architecture

The Context Discovery requires additional actuators in the MIRANDA Connector to proxy proprietary APIs of CMS or other kind of data source. They act similarly to plain actuators for security functions and have a common profile that conforms to the OpenC2 language specification, even if they are not really true security functions. The current implementation provides the following actuators:

Table 10. Context Discovery additional actuators

Name	Profile	CMS	Status
CTXDOpenstack	x-ctxd	OpenStack	Ready
CTXDKubernetes	x-ctxd	Kubernetes	Ready
CTXDDocker	x-ctxd	Docker	Under revision
CTXDProxmox	x-ctxd	Proxmox	Under revision
CTXDAzure	x-ctxd	Azure	Ongoing

The Discovery application starts from a list of input *Root Services* and queries their known OpenC2 Actuators to discover the current set of resources and their dependencies. The results are then delivered through a set of publishers to different sinks (databases, message buses, files). The current publishers enable writing to MondoDB, Kafka, and plain system files. Through the configuration file, the Discovery application starts a single discovery process for one or more root services. Through its REST API, the Discovery application can start multiple independent discovery processes for different sets of root services, query for running processes, and stop them. Each discovery process can run one-shot or loop for a predefined or unlimited number of times.

The Context Discovery implementation contributes to fulfilling the following system requirements:

- FR-A-1.8 – Displaying service context graphs: The Context Discovery provides the data to be displayed, including services, security functions, relationships.
- FR-A-1.10 – Navigating service context graphs: The SCG includes the description of digital resources ("*services*") and various kinds of relationships/dependencies between them ("*links*", e.g., hosted on, controlled by, protected by).
- FR-B-1.6 – Service types: The current list of service types includes: Application, Computer, VM, Pod, Container, Web Service, Cloud, Network (ethernet, 802.11, 802.15, zigbee, vlan, vpn, lorawan, wan), IoT, Operating System.
- FR-B-1.7 – Service relationships: The current list of links include: API, hosting, packet_flow, control, protect.
- FR-B-1.11 – Context API: The Context API is based on OpenC2, which is an open and public standard. The data abstraction is currently based on a custom data model developed internally to the Consortium, but the next version will extend the existing xBOM standards for better interoperability with existing tools.
- FR-B-1.13 – Context discovery implementation: The current implementation supports Kubernets, OpenStack, Docker, and Proxmox. Azure support is on-going. Chirpstack will be added in the following.
- NON-FR-B-1.16 – Time to discover the context: The time to discover the context is currently 2-6 seconds, depending on the delay of the CMS APIs. The latency introduced by the Context Discovery application itself is only a few hundred milliseconds for local network connections.

3.2.2. Communication interfaces

There are two communication interfaces used by the overall component:

- the REST API exposed by the Discovery application for its configuration;
- the OpenC2 interface between the Discovery and the MIRANDA Connector.

The OpenC2 interface is an internal interface between different components of the overall Context Discovery service. It uses a custom x-ctxd profile defined here.

The REST API to configure the Discovery application is described in the following table:

Table 11. REST API to configure the Discovery application

#	Method	REST Endpoint	Description	Request Body	Response Body

1	POST	/start	Start a discovery process from the list of provided service roots, and returns the id of started process.	{
				name: "...",
				frequency: 00,
				loop: 0,
				publishers: {
				mongodb: {... },
				file: ...,
					{
					"end_time": 00,
					"frequency": 0,
					"id": 000,
					"loop": 0,
					"start_time": 000
					}

				kafka: {...}, }, services: [{...}, {...} }	
2	GET	/threads	Request a list of active processes (includes processes that have already terminated their loop cycles)	–	[000, 000]
3	POST	/stop	Stop one or more discovery threads listed in the body. Returns the id of the discovery threads stopped.	[000, 000]	[000, 000]
4	POST	/clean	Clean up all discovery jobs. Returns the list of stopped threads.	–	[000, 000]

In addition to input data, the Discovery application publishes SCGs. The output format largely follows the data model of Context Discovery profile but it is packed in a common root element as follows (only json output is currently implemented, additional formats may be added in the future):

```
{
   "creator": "…",          Process that originated the record
   "jsonschema": "…",       Name of the json schema used for this record
   "date": "…",             Timestamp of this record creation
   "services": [            List of services discovered
      "source": {           Source of the data record (currently only OpenC2)
         "host": "…",       Hostname/IPv4/IPv6 of the OpenC2 actuator
         "port": 0000,      Port number
         "endpoint": "…",   Endpoint to contact the actuator (fixed for OpenC2)
         "encoding": "…",   Serialization format used by the OpenC2 Consumer
         "transfer": "…",   Transfer protocol used by the OpenC2 Consumer
         "profile": "…",    OpenC2 profile of the actuator
         "actuator": {…}    Identifier of the OpenC2 actuator (profile-
specific)
      }
      "service": {          Description of the digital service
         …                  See the ctxd service data model
      }
   ]
```

```
  "links": [              List of dependencies between reported services
    "source": {           Source of the data record (currently only OpenC2)
      … Same structure as above
    }
    "link": {             Description of the dependency
      … See the ctxd link data model
    }
  ]
}
```

Note that the current definition of *source* targets OpenC2 actuators, but it is general enough to fit other sources in the future.

The authoritative and updated json schema of data published by the Discovery is kept in the source code repository: schemas/json/ctxd.

The publishing options (MongoDB collection, Kafka topic, file name) are fully configurable (see Sec. 3.2.3). Below there is an example for Kafka:

Table 12. Example publishing options for Kafka

Topic	Description	Message Sample
discovery.scg	Example of message published on Kafka to notify new SCG discovered or updated	{ "date": 1767272009709, "creator": "test-discovery", "jsonschema": http://mirandaproject.eu/ctxd/v1.0/schema.json, "services": [{ "source": { "host": "127.0.0.1", "port": 8080, "endpoint": "/.well-known/openc2", "encoding": "json", "transfer": "http", "actuator": { "asset_id": "ctxd-kubernetes-example" } }, "service": { "name": { "local": "kubernetes"

```
        },
        "type": {
          "cloud": {
            "description": "Kubernetes cloud",
            "name": "kubernetes",
            "type": "IaaS"
...
}
  "links": [
    {
      "source": {
        "host": "127.0.0.1",
        "port": 8080,
        "endpoint": "/.well-known/openc2",
        "encoding": "json",
        "transfer": "http",
        "actuator": {
          "asset_id": "ctxd-kubernetes-example"
        }
      },
      "link": {
        "name": {
          "local": "kubernetes"
        },
        "description": "Kubernetes hosted on kube0",
        "link_type": "hosting",
        "peers": [
```


3.2.3. Deployment details

The Context Discovery is made by two distinct applications, both developed within the same Python library. One of them is the MIRANDA Connector, which is deployed the same way as already described in Sec. 3.1.3 (the actuator-specific configuration files must be provided).

The Discovery application can be run either by installing the component from the PyPi repository or by deploying a container.

From the **PyPi repository**, it can be installed (in a virtual environment) by running:

```
% pip install otupy
```

Then run the Discovery application (see below for the configuration files):

```
% python3 -m otupy.apps.ctxd.discovery [ -c | --config <config file>] [--api] [ -p | --port <port number ] [ --host <hostname | IP>]
```

The `--api` flag starts the API service. The config file is always expected and read (default: *discovery.yaml*), and merged with config files provided by the API at runtime. The `--port` and `--host` options are used to bind the API service to specific interfaces and ports, and they only make sense when the `--api` flag is set.

Alternatively, the **Docker image** already includes the command to run the Discovery. The Dockerfile builds on a common otupy docker image:

```
ARG VERSION="latest"

FROM ghcr.io/mattereppe/otupy:$VERSION

RUN cp -a /opt/otupy-venv/lib/python3*/site-packages/otupy/apps/ctxd/discovery.py .

CMD ["python3", "discovery.py", "-c", "/config/discovery.yaml", "--host", "0.0.0.0", "-p", "80", "--api"]
```

The Docker image always starts the API service.

Docker images for the Connector are automatically built for any new release, and available from the main otupy repository. To build a custom version for a specific version:

```
% docker build -t discovery --build-arg VERSION="x.y.z"
```

To deploy the Discovery applications from the images available from the repository:

- put the configuration file in a local folder (e.g., config-ctxd) and mount it under the /config folder of the container;
- expose the internal port (set in the configuration files, e.g.: 80) to the desired port available to external components (e.g.: 8081).

```
% docker run -v ./config-ctxd:/config -p 8081:80 --name discovery -it ghcr.io/mattereppe/discovery:latest
```

Docker compose files and Kubernetes manifests can be created according to the previous command line.

As far as the **configuration**, the following structure is used both for the configuration file and the API service:

```
name: "…"
frequency: 0
loop: 1
publishers:
  mongodb:
    host: '127.0.0.1'
    port: 27017
    db_name: "miranda"
    collection: "contextdata"
    user: "miranda"
    pass: "xxxx"
  file: Null
```

```
  kafka:
    host: '127.0.0.1'
    port: 29092
    topic: "demo"
#  security_protocol: "SSL"
    sasl_mechanism: Null
    sasl_plain_username: Null
    sasl_plain_password: Null
    ssl_cafile: 'ca-cert'
services:
  - host: '127.0.0.1'
    port: 8080
    endpoint: "/.well-known/openc2"
    encoding: "json"
    transfer: "http"
    actuator:
      asset_id: "ctxd-kubernetes-example"
  - host: '127.0.0.1'
    port: 8080
    actuator:
      asset_id: "ctxd-openstack-example"

logger:   <Logger configuration>
```

where:

- `name`: An identifier used to distinguish context originated by different discovery processes
- `frequency`: The time interval before starting a new round of queries (the real interval will be longer because the timer starts after receiving the last answer). Run one-shot if sets to 0.
- `loop`: Number of times to repeat the discovery. Loops forever if set to -1, does not run if set to 0.
- `publishers`: a dictionary of places where the context data are published after each round. The configuration changes according to the specific publisher:
  - `mongodb`: A dictionary with configuration to write data to MongoDB
    - `host`: IP address or hostname of the server hosting the database
    - `port`: Port number where the mongodb service listens to
    - `db_name`: Name of the internal database to store data
    - `collection`: Name to be used for the collection of documents
    - `user`: username to connect to the database
    - `pass`: password to connect to the database
  - `kafka`: A dictionary with the configuration to publish to Kafka brokers
    - `host`: IP address or hostname of the server hosting one bootstrap server
    - `port`: Port number where the bootstrap server listens to
    - `topic`: The topic used to publish data
    - `security_protocol`: Security protocol used to connect to Kafka (PLAINTEXT, SSL, SASL_PLAINTEXT, SASL_SSL - see Kafka documentation)
    - `sasl_mechanism`: Username/password authentication mechanism (PLAIN, GSSAPI, OAUTHBEARER, SCRAM-SHA-256, SCRAM-SHA-512). Only valid for security protocols SASL_PLAINTEXT or SASL_SSL
    - `sasl_plain_username`: Username when sasl mechanism is enabled
    - `sasl_plain_password`: Password when sasl mechanism is enabled

        - `ssl_cafile`: CA file used to sign the Kafka certificate
        - `ssl_check_hostname`: Enable (True) or disable (False) server name validation in the certificate file
    - `file`: A dictionary with configuration to write to file (append mode)
        - `name`: Filename (created if does not exist)
        - `path`: Filesystem path to the file (current directory if not given)
- `services`: A list of "root services" to query, each element indicated as their Consumer endpoints:
    - `host`: Hostname/IPv4/IPv6 of the OpenC2 actuator
    - `port`: Port number
    - `profile`: OpenC2 profile of the actuator
    - `encoding`: Serialization format used by the OpenC2 Consumer
    - `transfer`: Transfer protocol used by the OpenC2 Consumer
    - `endpoint`: Endpoint to contact the actuator (fixed for OpenC2)
    - `actuator`: Identifier of the OpenC2 actuator (profile-specific)
- `logger`: The configuration for the *Logging* framework. See the module documentation

The authoritative and updated description is kept with the source documentation here.

Actuator-specific configuration files for the Connector have the following general structure:

```
<actuator-id>:
  actuator: "actuator_name"
  profile: "ctxd"
  owner: "acme"
  specifiers:
    domain: null
    asset_id: "firewall:1"
  auth:
    …
  config:
    cacert: "/etc/ssl/certs/acme.crt"
```

where:

- `<actuator_id>` is an identifier of the instance (custom selection, do not refer to any internal data)
    - `actuator`: the name of the actuator (used to register it in the internal actuator list); current allowed values: ….
    - `profile`: name of the registered profile (currently: "slpf", "x-ctxd", "x-rcli", "x-nfm", "x-fclm"
    - `owner`: owner of the security function (informative only)
    - `specifiers`: OpenC2 identification of the actuator instance
        - `domain`: currently not used
        - `asset_id`: a string identifier for the actuator instance
    - `auth`: authentication information needed to control the CMS (depends on the specific CMS)
    - `config`: actuator-specific configuration (e.g., custom CA files, options, etc.)
- additional <key, value> pairs depending on the specific actuator

Actuator-specific configuration files

- Docker compose file
- Dockerfile
- Configuration file

3.2.4. Individual component testing

Table 13. Component Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-CTXD-01	OpenC2 profile verification.	Achieved
TC-CTXD-02	OpenStack actuator: collection of running VMs, OpenStack services, and links between VMs and physical servers (aka "hypervisors), link to OpenStack SG for firewalling.	Achieved
TC-CTXD-03	Kubernetes actuator: collection of running containers under a single and multiple workspace, and links between containers and computing nodes, link to K8S Network Policies for firewalling.	Achieved
TC-CTXD-04	Docker actuator: collection of running containers, and links between containers and hosting nodes.	To be tested
TC-CTXD-05	Proxmox actuator: collection of running containers, and links between containers and hosting nodes.	To be tested
TC-CTXD-06	Azure actuator: collection of running containers and links between containers and computing nodes, link to Azure firewall.	To be tested
TC-CTXD-07	Checking the correct publishing of the SCG.	Achieved
TC-CTXD-08	Discovery control APIs.	Achieved

Table 14. Test Case-CTXD-01

Test Case ID	**TC-CTXD-01**	**Component**	MIRANDA Connector + Discovery
Description	Implementation of the OpenC2 x-ctxd profile and verification of the correct communication between the Discovery and the Connector. The validation included the correct serialization/deserialization of all data types defined		
Tested by	CNR (component owner)		
Associated Requirements	FR-B-1.11		
Pre-condition(s)	OpenC2 ctxd profile defined and implemented. Connector and Discovery applications up and running		
Test steps			
1	Test data types with pytest.		
2	Create json schemas for expected data on the internal and external interface.		

3	Test internal OpenC2 communication against the json data schema.
Input data	Pytest scripts and json good/bad samples Step 1. Test samples and scripts available at: https://github.com/mattereppe/otupy/tree/main/tests/ctxd_types Step 2. Test samples and scripts available at: https://github.com/mattereppe/otupy/tree/main/tests/json/ctxd Step 3. - Kubernetes service: https://github.com/mattereppe/cloud-native-5g-testbed - Local experimental testbed with Kubernetes installed in 3 OpenStack VMs - Code with the necessary validation hooks available in the otupy branch "jsonvalidation": https://github.com/mattereppe/otupy/tree/jsonvalidation - Configuration files for MIRANDA Connector (including actuators) and Discovery Application.
Results	OpenC2 data types correctly instantiated from different inputs. Json serialization verified.
KPIs	Target -> 100% success | Measured -> 100% success
Test Case Result	Pass

Table 15. Test Cases -CTXD-02 & CTXD-03

Test Case ID	TC-CTXD-02, TC-CTXD-03	Component	MIRANDA Connector + Discovery
Description	A local testbed was setup including a Kubernetes installation in OpenStack VMs. A Kubernetes service was deployed and the Discovery service was run to discover both Kubernetes and OpenStack resources.		
Tested by	CNR (component owner)		
Associated Requirements	FR-A-1.8, FR-A-1.10, FR-B-1.6, FR-B-1.7, FR-B-1.13		
Pre-condition(s)	OpenC2 ctxd profile defined and implemented. Connector and Discovery applications up and running.		
Test steps			
1	Run the MIRANDA Connector.		
2	Run the Discovery service.		
3	Test different Discovery configurations (one-shot, periodic).		
Input data	Connector configuration. OpenStack and Kubernetes actuator configurations. Discovery configurations.		
Results	Data is correctly retrieved from Kubernetes and OpenStack CMS. The SCG is visualized to show the different components and their relationships [6].		

KPIs	Target -> 100% service/link discovered | Measured -> 100%
Test Case Result	Pass

Table 16. Test Case-CTXD-04

Test Case ID	**TC-CTXD-04**	**Component**	MIRANDA Connector + Discovery
Description	A local testbed is setup with Docker VMs. A few containers are deployed and the Discovery service is run to discover resources.		
Tested by	CNR (component owner)		
Associated Requirements	FR-A-1.8, FR-A-1.10, FR-B-1.6, FR-B-1.7, FR-B-1.13		
Pre-condition(s)	OpenC2 ctxd profile defined and implemented. Connector and Discovery applications up and running.		
Test steps			
1	Run the MIRANDA Connector.		
2	Run the Discovery service.		
Input data	Connector configuration. Docker actuator configuration. Discovery configuration.		
Results	TBD.		
KPIs	Target -> 100% service/link discovered | Measured -> TBD		
Test Case Result	Pass		

Table 17. Test Cases -CTXD-02 & CTXD-05

Test Case ID	**TC-CTXD-02, TC-CTXD-05**	**Component**	MIRANDA Connector + Discovery
Description	A local testbed is setup including a Kubernetes installation in Proxmox VMs. A Kubernetes service is deployed and the Discovery service was run to discover both Kubernetes and Proxmox resources.		
Tested by	CNR (component owner)		
Associated Requirements	FR-A-1.8, FR-A-1.10, FR-B-1.6, FR-B-1.7, FR-B-1.13		
Pre-condition(s)	OpenC2 ctxd profile defined and implemented. Connector and Discovery applications up and running.		
Test steps			
1	Run the MIRANDA Connector.		

2	Run the Discovery service.
Input data	Connector configuration. Proxmox and Kubernetes actuator configurations. Discovery configurations. Kubernetes service: https://github.com/mattereppe/cloud-native-5g-testbed.
Results	TBD.
KPIs	Target -> 100% service/link discovered | Measured -> TBD
Test Case Result	Pass

Table 18. Test Case CTXD-06

Test Case ID	TC-CTXD-06	Component	MIRANDA Connector + Discovery
Description	The Discovery process is tested in the Azure testbed of Use Case #2.		
Tested by	CNR (component owner)		
Associated Requirements	FR-A-1.8, FR-A-1.10, FR-B-1.6, FR-B-1.7, FR-B-1.13		
Pre-condition(s)	OpenC2 ctxd profile defined and implemented. Mindicity UC deployed and running in Azure. Connector and Discovery applications up and running.		
Test steps			
1	Run the MIRANDA Connector.		
2	Run the Discovery service.		
Input data	Connector configuration. Azure actuator configurations. Discovery configuration.		
Results	TBD.		
KPIs	Target -> 100% service/link discovered | Measured -> TBD		
Test Case Result	Pass		

Table 19. Test Case-CTXD-07

Test Case ID	TC-CTXD-07	Component	Discovery
Description	The Discovery process publishes the SCG on files, MongoDB, and Kafka.		
Tested by	CNR (component owner)		
Associated Requirements	FR-B-1.13		

Pre-condition(s)	A running instance of Kafka. A running instance of MongoDB. A running instance of Connector + Discovery. A testbed with OpenStack/Kubernetes resources.
Test steps	
1	Start a MondoDB docker container.
2	Start a Kafka docker container
3	Run the MIRANDA Connector
4	Run the Discovery service.
5	Check the data pushed to file, MondoDB, Kafka.
Input data	Connector configuration. Discovery configuration.
Results	Json data is compliant to the schema provided to partners. Data are incrementally added to MongoDB and the file. Data are pushed and retained in Kafka.
KPIs	Target -> 100% json correctness | Measured -> 100% correctness.
Test Case Result	Pass

Table 20. Test Case-CTXD-08

Test Case ID	TC-CTXD-08	Component	Discovery
Description	Verify the correctness of the control API exposed by Discovery application.		
Tested by	CNR (component owner)		
Associated Requirements	FR-B-1.13		
Pre-condition(s)	A running instance of Connector + Discovery. A testbed with OpenStack/Kubernetes resources.		
Test steps			

1	Run the MIRANDA Connector
2	Run the Discovery service.
3	Start 3 discovery processes (oneshot).
4	Query running discovery processes.
5	Stop 1 process, stop 2 processes.
6	Start 2 more discovery processes.
7	Cleanup all processes
Input data	Connector configuration. Discovery configuration. Postman commands.
Results	Discovery processes are correctly started/reported/terminated.
KPIs	-
Test Case Result	Pass

3.2.5. Next steps and future development updates

The actuators currently under revision or development will be completed.

More significantly, a new profile is currently under development that will use the Cyclone DX for both saving the data and retrieving data from the Connector. This will affect the implementation of both the Discovery and Actuators, and a different profile will be proposed to avoid overlapping with the current implementation.

Finally, the application will be integrated into the same IdM framework as the Connector for cross-domain usage.

3.3 Attack & Threat Modelling

3.3.1. Prototype Description

The Attack & Threat Modelling component is responsible for generating Attack Path Trees, which model multi-step attack kill chains and are essential for understanding and proactively predicting the propagation of attacks. The Attack Path Trees are computed by the component after formally inferring, through the application of a set of formal derivation rules, security

properties of the entities and the conditions for a successful attack to be perpetrated in the system.

This component satisfies the following requirements:

- Threat Analysis: the Attack & Threat Modelling component fully supports the SWI dialect of Prolog, using it to define the model and the formal derivation rules governing the threat analysis process.
- FR-A-1.9 - Displaying vulnerabilities and threats – the component computes security properties such as vulnerability and compromission for each service and relationship in the SCG, as well as the threats associated with them.

3.3.2. Deployment details

The component is available for usage in two modalities:

- Docker
  - Prerequisites: Docker and Docker Compose
  - Run:
    - `docker-compose up –build`
  - Port exposed: 5001
- Local Installation
  - Prerequisites: Python3, SWI-Prolog, Graphviz
  - Run:
    - `./setup.sh`
    - `./start.sh`
  - Port exposed: 5001

3.3.3. Individual component testing

Table 21. Component Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-AM-01	Run Derivation Engine	Achieved
TC-AM-02	Retrieve Attack Path Tree	Achieved

Table 22. Test Case-AM-01

Test Case ID	**TC-AM-01**	**Component**	Attack & Threat Modeling
Description	Run the Derivation Engine and test correctness and self-consistency of Derivation Rules		
Tested by	POLITO		
Associated Requirements	FR-A-1.2		

Pre-condition(s)	Component up and running, Knowledge Base defined
Test steps	
1	Send Knowledge Base to the component
2	Run Derivation Engine
3	Retrieve list of derived facts
Input data	Knowledge Base in Prolog syntax
Results	The Derivation Engine run successfully, with no error during execution
KPIs	Errors during inference of facts? Target -> No | Measured -> No
Test Case Result	Pass

Table 23. Test Case-AM-02

Test Case ID	**TC-AM-02**	**Component**	Attack & Threat Modeling
Description	Run the Derivation Engine and retrieve Attack Paths Tree		
Tested by	POLITO		
Associated Requirements	FR-A-1.2, FR-A-1.9		
Pre-condition(s)	Component up and running, Knowledge Base defined		
Test steps			
1	Send Knowledge Base to the component		
2	Run Derivation Engine		
3	Retrieve list of derived facts		
4	Send request for Attack Path Tree for a derived fact		
5	Retrieve Attack Path Tree		
Input data	Knowledge Base in Prolog syntax, derived fact		
Results	The Attack Path Tree has been successfully generated by the component, visualizing the complete path that led to the derivation of asked fact		
KPIs	Errors during generation of Attack Path Tree? Target -> No | Measured -> No Errors during visualization of Attack Path Tree? Target -> No | Measured -> No		
Test Case Result	Pass		

3.3.4. Next steps and future development updates

For the next steps of development:

- Integration with other components for automatic generation of Knowledge Base
- Finalize REST API interface

3.4 Data Handling Pipeline

3.4.1. DHP Actuator & Pipeline Engine

3.4.1.1. Prototype Description

The OpenC2 Pipeline Orchestrator & Engine is an enterprise-grade, dynamic ETL (Extract, Transform, Load) component within the MIRANDA Data Handling Pipeline. Its primary purpose is to enable the real-time ingestion, transformation, and routing of data streams (e.g., logs, events, netflows) without requiring manual, static configuration files. Instead, it is fully controllable via the OpenC2 (Open Command and Control) standard, allowing security operators and automated orchestration tools to deploy data privacy and routing policies on-the-fly.

Architecture

The architecture is divided into two tightly integrated planes:

1. **Control Plane (DHP Actuator):** A RESTful API built with Python and FastAPI. It acts as the OpenC2 listener, receiving standardized JSON payloads via POST requests.
2. **Data Plane (Logstash Engine):** A high-performance Elastic Logstash engine that monitors the configuration directory. Upon receiving a dynamically compiled pipeline from the Actuator, it reloads seamlessly to apply the new rules in-memory.

The Anatomy of a Pipeline Command

Every command sent to the pipeline via the OpenC2 API follows a strict JSON payload structure, which consists of two main sections:

3. **Headers:** Contains metadata for the request, including a unique request_id for tracking and debugging.
4. **Body:** Contains the standard OpenC2 message structure (openc2 -> request). For pipeline creation, the action is strictly set to "start", and the target is defined as "x-dpp:pipeline".

The core definition of the x-dpp:pipeline object revolves around three key arrays:

1. **Sources:** Defines where the data originates. The pipeline natively supports local file ingestion (e.g., app.log) and Apache Kafka topics (supporting Single-Cluster and Cross-Cluster Replication).
2. **Transformations:** An array of operation rules defining how the data is processed. These steps are executed sequentially.
3. **Sinks:** Defines the output destinations, such as writing structured JSON to local files or producing events to downstream Kafka topics.

Supported Transformation Operations The engine supports a comprehensive suite of data manipulation operations to structure, enrich, and protect flowing data:

- **Parse:** Extracts structured data from raw text (e.g., log lines) using predefined grok patterns.
- **Enrich:** Adds new static fields or tags to events (e.g., tagging the data_source).
- **Transform:** Modifies the data type of existing fields (e.g., converting a string-based HTTP response code to an integer).
- **Mask:** Obfuscates sensitive fields (e.g., emails, IP addresses) replacing them with asterisks to ensure GDPR compliance and data privacy.
- **Normalize:** Cleanses data by converting fields to lowercase, stripping whitespaces, and renaming keys for structural uniformity.
- **Filter & Validate:** Evaluates logical conditions (e.g., [level] == 'DEBUG'). *Filter* drops events that match the condition, while *Validate* retains only the events that satisfy it.
- **Route:** Dynamically tags events based on logical conditions to route them to specific downstream sinks.
- **Aggregate & Buffer:** Stateful operations that correlate streaming events. The *Buffer* operation batches multiple single events into structured arrays based on configurable sizes or timeouts, significantly optimizing downstream storage ingestion.

3.4.1.2. Communication interfaces

The OpenC2 Pipeline Orchestrator relies on two primary communication interfaces: a RESTful API for control plane operations (orchestration and command reception) and Apache Kafka for data plane operations (high-throughput data streaming).

Table 38. DHP REST API Interfaces

Method	REST Endpoint	Description	Request Body	Response Body
POST	/.well-known/openc2	The primary OpenC2 listener endpoint. It receives standardized commands to dynamically configure, start, or modify Logstash processing pipelines.	OpenC2 JSON Payload (contains action: start, target: x-dpp:pipeline, sources, transformations, sinks).	JSON Object containing the execution status (e.g., 200 OK), status text, and the dynamically assigned pipeline_id.

Table 39. DHP Kafka InterfacesTopic

Topic	Description	Message Sample
Dynamic Input Topic(s)	The pipeline subscribes to dynamically configured Kafka topics (defined via the OpenC2 payload) to ingest raw data or events from other MIRANDA components.	{"log_level": "ERROR", "user_email": "admin@miranda.eu", "message": "Connection timeout"}

Dynamic Output Topic(s)	The pipeline produces the transformed, masked, or aggregated events to specific Kafka topics (sinks) for downstream processing or storage.	{"log_level": "ERROR", "user_email": "*****", "tags": ["routed_critical_alerts"]}

3.4.1.3. Deployment details

The OpenC2 Pipeline Orchestrator & Engine is fully containerized to ensure portability, scalability, and ease of deployment across different environments. The deployment relies on Docker and Docker Compose to orchestrate the interconnected microservices.

Docker Compose

The prototype is deployed using a multi-container Docker Compose configuration that provisions the following core services:

- **dhp-api:** The FastAPI-based Control Plane that exposes the OpenC2 endpoint on port 8080.
- **logstash-service:** The Data Plane engine responsible for executing the pipelines. It is explicitly configured with config.reload.automatic: true to seamlessly apply new OpenC2 rules without requiring container restarts.
- **kafka & zookeeper:** Local message brokers used for streaming data ingestion and routing (although the pipeline is fully capable of connecting to remote, cross-cluster Kafka instances).

Security and Volume Management

Following DevSecOps best practices, the services are configured to run with least-privilege access. Specifically, the Logstash container runs as a non-root user (UID 1000) to mitigate host-level security risks. Shared Docker volumes (/app/pipeline/) are utilized to bridge the dhp-api and logstash-service. When the API generates a new Logstash DSL configuration, it writes it to this shared volume, triggering an immediate, zero-downtime pipeline reload by the Logstash engine.

Dockerfile

The deployment utilizes two primary images:

1. **DHP Actuator:** Built from a lightweight Python base image (python:3.10-slim). It installs the necessary dependencies (FastAPI, Uvicorn, OpenC2 schemas) and exposes the REST API.
2. **Logstash Engine:** Built on top of the official Elastic image (logstash:8.x). The Dockerfile is customized to install required external plugins, such as logstash-filter-aggregate, which is essential for the stateful batching and buffering operations.

Execution Instructions

To build the images and start the entire Data Handling Pipeline infrastructure in detached mode, navigate to the root directory containing the docker-compose.yml and execute:

```
docker-compose up
```

To verify that the API is actively listening for OpenC2 commands, operators can monitor the logs using:

```
docker logs -f data-handling-pipeline-api
```

3.4.1.4. Individual component testing

The following test cases demonstrate the validation of the OpenC2 Pipeline Orchestrator's core functionalities, ensuring that OpenC2 commands are properly parsed and that the Logstash Engine executes the dynamic data transformations (Masking, Routing, Buffering) accurately.

Table 40. DHP Component Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-DHP-01	OpenC2 Command Parsing and Dynamic Logstash DSL Generation	SUCCESS
TC-DHP-02	Data Privacy (Mask operation) and Normalization	SUCCESS
TC-DHP-03	Stateful Batching (Buffer operation) and Aggregation	SUCCESS
TC-DHP-04	Conditional Routing (Tagging) and Kafka Integration	SUCCESS
TC-DHP-05	Cross-Cluster Kafka Replication (Kafka Bridge)	SUCCESS
TC-DHP-06	API Payload Validation and Error Handling	SUCCESS
TC-DHP-07	Conditional Event Filtering and Noise Reduction	SUCCESS

Table 41. Test Case DHP-01

Test Case ID	TC-DHP-01	Component	OpenC2 Pipeline Orchestrator
Description	Verify that the DHP Actuator correctly receives an OpenC2 start command via the REST API and dynamically generates a valid Logstash pipeline configuration file.		
Pre-condition(s)	DHP API and Logstash Engine containers are running.		
Test steps			
1	Send a POST request to /.well-known/openc2 with a valid OpenC2 JSON payload containing sources, transformations, and sinks.		

2	Check the API response.
3	Verify the creation of the .conf file in the shared Docker volume.
Results	API returns 200 OK with a generated pipeline_id. The Logstash configuration file is successfully created and syntactically correct.
Test Case Result	Pass

Table 42. Test Case DHP-02

Test Case ID	TC-DHP-02	Component	Logstash Engine (Data Plane)
Description	Validate the correct execution of the mask and normalize operations to ensure data privacy (GDPR compliance) and uniform data structuring.		
Pre-condition(s)	Pipeline deployed via TC-DHP-01 containing mask (e.g., emails) and normalize (e.g., lowercase) rules.		
Test steps			
1	Ingest sample JSON events containing sensitive information and uppercase fields.		
2	Observe the output sink (file or Kafka).		
Results	The defined sensitive fields are obfuscated with ***** and specified fields are converted to lowercase. The original event structure is safely removed (event.original cleanup).		
Test Case Result	Pass		

Table 43. Test Case DHP-03

Test Case ID	TC-DHP-03	Component	Logstash Engine (Data Plane)
Description	Test the buffer operation, verifying that streaming events are accurately grouped into stateful arrays based on a specified batch size or timeout.		
Pre-condition(s)	Pipeline deployed containing the buffer operation with a dynamic task_id (e.g., %{host}) and timeout_field.		
Test steps			
1	Push multiple single events sequentially into the input source.		
2	Wait for the batch size limit to be reached or the timeout window to expire.		
Results	The output sink receives a single, consolidated JSON event containing an array of the ingested messages, grouped correctly by their dynamic task_id. Batching significantly reduced the event output rate, optimizing downstream ingestion.		
Test Case Result	Pass		

Table 44. Test Case DHP-04

Test Case ID	TC-DHP-04	Component	Logstash Engine (Data Plane)
Description	Verify conditional logic (route operation) and end-to-end integration with external Apache Kafka brokers.		
Pre-condition(s)	Pipeline configured with a Kafka input topic, a Kafka output topic, and a route condition (e.g., [level] == 'ERROR').		
Test steps			
1	Use a Kafka Producer to send mixed messages (INFO, ERROR) to the input topic.		
2	Use a Kafka Consumer to monitor the output topic.		
Results	Messages successfully traverse the Logstash Engine. Only the messages meeting the ERROR condition receive the specific routing tag (e.g., routed_critical).		
Test Case Result	Pass		

Table 45. Test Case DHP-05

Test Case ID	TC-DHP-05	Component	OpenC2 Pipeline Orchestrator
Description	Verify the DHP's capability to act as a cross-cluster bridge, consuming streams from one Kafka broker and producing to an entirely isolated Kafka broker in real-time.		
Pre-condition(s)	Two separate Kafka clusters (e.g., kafka-source:9092 and kafka-dest:29092) accessible by the Logstash container.		
Test steps			
1	Deploy a pipeline via OpenC2 targeting the source broker in the sources array and the destination broker in the sinks array.		
2	Produce messages to the source broker.		
3	Consume messages from the destination broker.		
Results	Data is seamlessly forwarded from the input cluster to the output cluster without any data loss or connection drops. The pipeline successfully maintained concurrent connections to multiple isolated brokers, effectively acting as a data bridge.		
Test Case Result	Pass		

Table 46. Test Case DHP-06

Test Case ID	TC-DHP-06	Component	DHP Actuator (Control Plane)
Description	Ensure the REST API possesses robust validation mechanisms to gracefully reject malformed OpenC2 commands or unsupported operations without crashing.		
Pre-condition(s)	DHP API is running and actively listening for commands.		
Test steps			
1	1. Dispatch an OpenC2 JSON payload containing a syntax error (e.g., missing brackets).		
2	2. Dispatch a structurally valid payload requesting a non-existent operation (e.g., "operation": "delete").		
Results	The API does not crash. It intercepts the invalid requests and returns an HTTP 400 Bad Request. No corrupted Logstash configuration is generated.		
Test Case Result	Pass		

Table 47. Test Case DHP-07

Test Case ID	TC-DHP-07	Component	OpenC2 Pipeline Orchestrator
Description	Test the dynamic filter operation to ensure that noisy or irrelevant events are dropped from the pipeline early, saving downstream bandwidth and storage.		
Pre-condition(s)	Pipeline deployed with a filter operation containing a specific drop condition (e.g., [level] == 'DEBUG').		
Test steps			
1	Stream a mixed batch of log events containing INFO, ERROR, and DEBUG levels.		
2	Monitor the output sink.		
Results	All DEBUG events are instantly dropped in-memory. Only INFO and ERROR events reach the final destination.		
Test Case Result	Pass		

3.4.2. Log Parser

The data handling pipeline contains the log parser of the anomaly detection tool DetectMate (previously AMiner). The DetectMate is a framework for developing and deploying anomaly detection algorithms for log data anomaly detection.

3.4.2.1. Prototype Description

Log parsing is the process of structuring semi-structured logs into a structured representation.

The process of the log parser is:

1. Receive configuration from user via API interface
2. Read incoming logs from Kafka MQ
3. Parse logs
4. Send parsed logs to Kafka MQ

The log parser applies predefined log templates to the logs to extract the variable part of logs and identify the log type of a log. This information is important for downstream tasks like anomaly detection.

The architecture is simple. It contains a connector to the Kafka MQ and a set of parsing algorithms. Based on the configuration file and the predefined templates the parser adapts itself to the logs and is able to parse them.

The log parser is relevant to the following requirements:

- NON-FR-B-1.35
- NON-FR-B-1.34
- SR-SPL-04
- FR-A-1.5
- FR-A-1.4

3.4.2.2. Communication interfaces

Table 24. DHP Communication interfaces

#	Method	REST Endpoint	Description	Request Body	Response Body

1	GET	/admin/status	Returns status (running state, and current configurations)	None	"{
					""status"": {
					""component_type"": ""string"",
					""component_id"": ""string"",
					""running"": true|false
					},
					""settings"": {
					""key"": ""value""
					},
					""configs"": {

					""key"": ""value"" } }"
2	POST	/admin/start	Starts the data processing engine thread.	None	{"message": "..."}
3	POST	/admin/stop	Stops the data processing engine thread.	None	{"message": "..."}
4	POST	/admin/reconfigure	Dynamically updates service parameters.	{"config": dict, "persist": bool}	{"message": "..."}
5	POST	/admin/shutdown	Gracefully terminates the entire service process.	None	{"message": "..."}

Table 25. Communication interfaces

Topic	Description	Message Sample
input	The input (in JSON format) the log parser requires from Kafka MQ.	{ "__version__": "string", "logID": 0, "log": "string", "logSource": "string", "hostname": "string"}
output	The output (in JSON format) the log parser sends to Kafka MQ.	{ "__version__": "string", "parserType": "string", "parserID": "string", "EventID": 0, "template": "string", "variables": ["string"], "parsedLogID": 0, "logID": 0, "log": "string", "logFormatVariables": { "key": "value" }, "receivedTimestamp": 0, "parsedTimestamp": 0}

3.4.2.3. Deployment details

First, clone DetectMateService and navigate into the repository:

```
git clone https://github.com/ait-detectmate/DetectMateLibrary.git
cd DetectMateService
git checkout demo_m
```

We recommend using uv to manage the environment and dependencies.

1. Create and activate a virtual environment with uv:

```
uv venv
source .venv/bin/activate
```

2. Install the project:

```
uv pip install .
```

If you prefer plain pip, you can set things up like this instead:

```
# Create a virtual environment
python -m venv .venv
# Activate it
source .venv/bin/activate
# Install the project in editable mode with dev dependencies
pip install .
```

To run the service with custom variables, we can define settings. For example, create a file named settings.yaml:

```
component_name: my-first-service
component_type: core # or use a library component like "detectors.RandomDetector"
log_level: INFO
log_dir: ./logs
manager_addr: ipc:///tmp/detectmate.cmd.ipc
engine_addr: ipc:///tmp/detectmate.engine.ipc
```

The config.yaml file that defines the parser for JSON logs looks like this:

```
parsers:

    JsonMatcherParser:
        method_type: matcher_parser
        auto_config: False
        log_format: "<Content>"
        time_format: null
        params:
            remove_spaces: True
            remove_punctuation: True
            lowercase: True
            path_templates: data/miranda_templates.txt

    JsonParser:
        method_type: json_parser
        time_format: null
        auto_config: False
        params:
            timestamp_name: "time"
            content_name: "message"
            content_parser: JsonMatcherParser
```

The JSON parser requires predefined template for the logs which must be stored in data/miranda_templates.txt.

The Dockerfile is given as:

```
FROM python:3.12-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
    git && \
    rm -rf /var/lib/apt/lists/*

COPY --from=ghcr.io/astral-sh/uv:latest /uv /usr/local/bin/uv
#RUN pip install uv

COPY . /app
COPY pyproject.toml .

RUN uv pip install --system -e .

CMD ["uv", "run", "demo/detectmate_parser.py"]
```

The docker-compose file is given as:

```
services:
  app1:
    build:
      dockerfile: demo/Dockerfile.app1
      context: ..
    environment:
      KAFKA_SERVER: kafka:9092
    depends_on:
      - kafka
  kafka:
    hostname: kafka
    container_name: kafka
    image: apache/kafka-native
    ports:
      - "9092:9092"
    environment:
      # Configure listeners for both docker and host communication
      KAFKA_LISTENERS: 'PLAINTEXT://kafka:9092,CONTROLLER://kafka:9093'
      KAFKA_ADVERTISED_LISTENERS: 'PLAINTEXT://kafka:9092'
                                        KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
'CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT'

      # Settings required for KRaft mode
      KAFKA_NODE_ID: 1
      KAFKA_PROCESS_ROLES: broker,controller
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      KAFKA_CONTROLLER_QUORUM_VOTERS: '1@kafka:9093'

      # Listener to use for broker-to-broker communication
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT

      # Required for a single node cluster
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

To run everything:

```
docker-compose up
```

3.4.2.4. Individual component testing

Table 26. Component Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-LP-01	Parsing log lines.	Achieved
TC-LP-02	Deployment and successful operation with Kafka	To be tested

Table 27. Test Case Log Parser-01

Test Case ID	**TC-LP-01**	**Component**	**Log Parser**
Description	To test the functionality of the log parser we let it parse log lines.		
Tested by	AIT		
Associated Requirements	SR-SPL-08		
Pre-condition(s)	input data and configuration given		
Test steps			
1	The input log are ingested by the core function of the parser.		
2	Monitor errors while processing the logs.		
3	The output of the core function is verified and correct according to a human validator.		
Input data	A log file (Mindicity logs), type: JSON		
Results	The parser was successful, ran without errors and parsed all logs of the file.		
KPIs	All logs parsed correctly? Target -> True | Measured -> True		
Test Case Result	Pass		

Table 28. Test Case Log Parser-02

Test Case ID	**TC-LP-02**	**Component**	**Log Parser**
Description	The log parser is deployed with Docker and receives logs from Kafka and sends the parsed logs to Kafka MQ.		
Tested by	AIT		
Associated Requirements	FR-B-1.27, SR-SPL-08		

Pre-condition(s)	input data and configuration given
Test steps	
1	Deployment via Docker
2	Start of the DetectMate with correct settings and parser config.
3	Simulated Kafka MQ sends data to the deployed component.
4	Log parser parses data and sends parsed logs to Kafka MQ.
5	Simulated receiver receives the logs
6	Parsed logs are validated by human validator.
Input data	Type and format of data
Results	Screenshots, descriptions, etc.
KPIs	No errors occurred? Target -> True | Measured ->
Test Case Result	Pass / Failed

3.4.2.5. Next steps and future development updates

The component will be made more robust and efficient. The current parsing method is reliable but could be optimized for efficiency. More sophisticated parsers will be added to enable not only parsing (== log template matching with existing templates) but also online template extraction so that templates do not have to be provided by the user.

3.4.3. NetFlow Parser

3.4.3.1. Prototype Description

The prototype implements a **high-performance, asynchronous NetFlow/IPFIX collection and processing pipeline** designed for continuous network traffic monitoring.

The architecture is **single-service, event-driven**, optimized for high-throughput UDP ingestion:

Collector Layer

- Listens on UDP ports for:
- NetFlow v5 (UDP 2055)
- IPFIX (UDP 4739)
- Decoding is protocol-specific and implemented via dedicated decoder modules.
- Uses asyncio-based non-blocking I/O to handle high packet rates.

Processing & Normalization Layer

- Decoded packets are transformed into a unified `Flow` data model.
- Sampling correction is applied when exporters use packet sampling.
- Flow records are enriched with:

- GeoIP (city and ASN)
- Internal vs external traffic classification
- Device and interface metadata

Queueing & Backpressure Layer

- Uses a bounded asynchronous queue to absorb traffic bursts.
- Implements controlled data dropping when queue limits are exceeded to protect memory.

Output Layer

- Supports multiple output backends:
- Kafka (primary)
- File (JSON Lines)
- Standard output
- Output selection is fully configuration-driven.

Deployment Layer

- Containerized using Docker
- Network-isolated using Docker bridge networking
- Stateless runtime, configuration injected at startup
- This architecture prioritizes **throughput, resilience, and operational simplicity** over feature-heavy orchestration.

3.4.3.2. Communication interfaces

Table 29. Network Interfaces

#	Protocol	Port	Description
1	NetFlow v5	2055/UDP	Receives NetFlow v5 packets
2	IPFIX	4739/TCP	Receives IPFIX packets

Table 30. Kafka Interfaces

Topic	Description	Message Sample

netflows	Enriched flow records	{
		"ts_start": 1769373462.618,
		"ts_end": 1769373462.618,
		"src_ip": "172.16.30.114",
		"dst_ip": "172.16.30.23",
		"src_port": 9201,
		"dst_port": 36564,
		}

3.4.3.3. Deployment details

Docker Compose

The prototype is deployed using a single-service Docker Compose configuration:

- Exposes UDP port 2055 for NetFlow ingestion
- Uses a dedicated bridge network with static IP allocation
- Mounts configuration file as read-only
- Automatic restart policy enabled
- Network isolation ensures the collector is not exposed beyond required ports.

Dockerfile

The Dockerfile builds a lightweight Python runtime image that:

- Installs only required dependencies
- Runs the application as a single foreground process
- Uses environment variables for runtime behaviour
- Avoids unnecessary build artifacts

Configuration File

All runtime behaviour is defined in *config.yaml*, including:

- Listener ports and protocols
- Output backends and parameters
- Kafka broker configuration
- Enrichment data sources
- Time zone behaviour
- Device and interface mappings

This enables environment-specific tuning without code changes.

3.4.3.4. Individual component testing

Table 31. Component Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-NFlow-01	Receive and decode NetFlow v5 packets	To be tested
TC-NFlow-02	Receive and decode IPFIX packets	To be tested
TC-NFlow-03	Normalization of decoded flow records	To be tested
TC-NFlow-04	Apply GeoIP and ASN enrichment	To be tested
TC-NFlow-05	Publish flows to Kafka	To be tested

Table 32. Test Case NetFlow Collector-01

Test Case ID	TC-NFlow-01	Component	NetFlow Collector
Description	Verify correct reception and decoding of NetFlow v5 packets.		
Tested by	Logstail		
Associated Requirements	FR-DATA-01, FR-DATA-02, NFR-SYS-01, NFR-SYS-02		
Pre-condition(s)	• Docker container running • UDP port 2055 exposed • Valid NetFlow exporter configured		
Test steps			
1	Send NetFlow v5 packets to UDP port 2055		
2	Verify packets are received by the collector		
3	Validate decoded flow fields		
4	Confirm flow is forwarded to output queue		
Input data	Binary NetFlow v5 packets		
Results	Decoded flow records visible in Kafka topic or output file		
KPIs	Target → ≥ 99% successful decoding Measured → To be measured		
Test Case Result	Pass / Failed		

Table 33. Test Case NetFlow Collector-02

Test Case ID	TC-NFlow-02	Component	NetFlow Collector
Description	Verify correct reception and decoding of IPFIX packets.		
Tested by	Logstail		
Associated Requirements	FR-DATA-01, FR-DATA-02, NFR-SYS-01, NFR-SYS-04		
Pre-condition(s)	• Docker container running • UDP port 4739 exposed • Valid IPFIX exporter configured		
Test steps			
1	Send IPFIX packets to UDP port 4739		
2	Verify packets are received by the collector		
3	Validate decoded IPFIX fields		

4	Confirm records are normalized into Flow model
Input data	Binary IPFIX packets
Results	Normalized flow records available in output pipeline
KPIs	Target → ≥ 99% successful decoding Measured → To be measured
Test Case Result	Pass / Failed

Table 34. Test Case NetFlow Collector-03

Test Case ID	TC-NFlow-03	Component	NetFlow Collector
Description	Verify normalization of decoded flow records into the unified Flow schema.		
Tested by	Logstail		
Associated Requirements	FR-DATA-03, FR-DATA-04, NFR-SYS-02		
Pre-condition(s)	• Collector receiving valid flow data • Processing queue active		
Test steps			
1	Ingest decoded NetFlow/IPFIX records		
2	Apply normalization logic		
3	Validate mandatory Flow fields		
4	Forward normalized flow to enrichment stage		
Input data	Decoded flow objects		
Results	Flow records conforming to unified schema		
KPIs	Target → 100% schema compliance Measured → To be measured		
Test Case Result	Pass / Failed		

Table 35. Test Case NetFlow Collector-04

Test Case ID	TC-NFlow-04	Component	NetFlow Collector
Description	Verify correct enrichment of flow records with GeoIP and ASN data.		
Tested by	Logstail		
Associated Requirements	FR-DATA-05, NFR-SYS-03, NFR-SYS-02		

Pre-condition(s)	• GeoIP and ASN databases available Normalized flows entering enrichment stage
Test steps	
1	Process normalized flow record
2	Resolve GeoIP information
3	Resolve ASN information
4	Append enrichment fields to flow
Input data	Normalized flow records
Results	Flow records enriched with GeoIP and ASN metadata
KPIs	Target → 95% schema compliance Measured → To be measured
Test Case Result	Pass / Failed

Table 36. Test Case NetFlow Collector-05

Test Case ID	TC-NFlow-05	Component	NetFlow Collector
Description	Verify correct publishing of flow records to Kafka.		
Tested by	Logstail		
Associated Requirements	FR-DATA-06, NFR-SYS-04, NFR-SYS-01		
Pre-condition(s)	• Kafka broker available • Topic configured • Output enabled in config		
Test steps			
1	Generate valid flow records		
2	Forward flows to output module		
3	Publish flows to Kafka topic		
4	Consume messages to verify content		
Input data	Enriched flow records		
Results	Flow records successfully published and consumable		
KPIs	Target → 99.8% schema compliance Measured → To be measured		
Test Case Result	Pass / Failed		

3.4.3.5. Next steps and future development updates

Planned improvements include:

- REST API for operational monitoring and integration with other components.

3.5 Prediction

3.5.1. Prototype Description

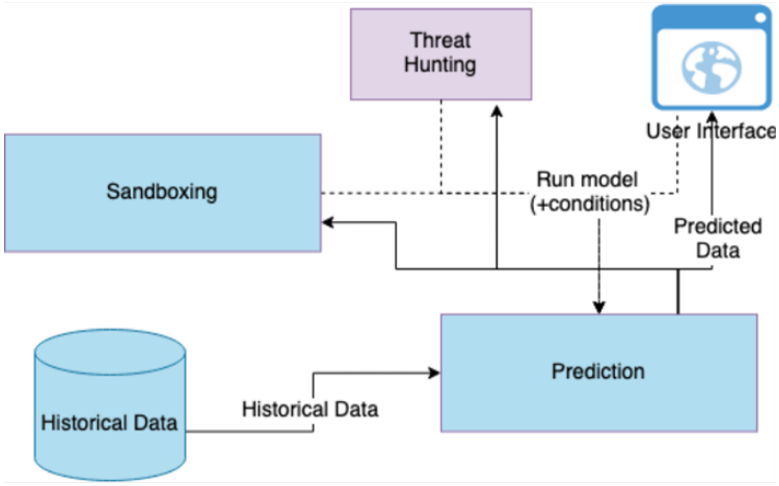

Figure 4. Architecture (System Perspective)

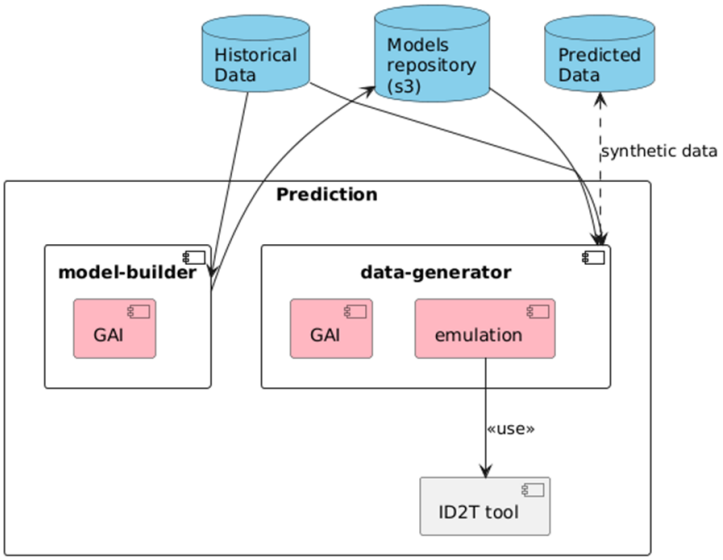

Figure 5. Component Architecture

Functionalities:

1. Generation of synthetic raw traffic produced by a LoRa network based on historical data
2. Predict LoRa network traffic in the given conditions, not observed in the past

3. Generation of synthetic TCP/IP traffic (aggregated TCP/IP flows) based on historical data
4. Generation of synthetic raw TCP/IP traffic during attacks (not observed in the past) over historical benign data

Map with requirements:

- FR-A-1.12 (Making hypotheses and running predictions)

3.5.2. Communication interfaces

Table 37. Communication interfaces

#	Method	REST Endpoint	Description	Request Body	Response Body
1	POST	/prediction/tasks/models	Build a generative model from historical data. Returns task_id for async tracking. Model stored in repository upon completion.	{ "model_type": "GAN|CGAN|VAE", "data_type": "tcp_flow|tcp_raw|lora_raw", "prediction_type": "new_device|new_traffic|augmentation", "data_query": { "source": "database", "filter": {...} } }	{ "task_id": "uuid", "status": "queued" }
2	GET	/prediction/models/{model_id}	Get model metadata and training status.	n/a	{ "model_id": "uuid", "status": "training|ready|failed", "model_type": "GAN", "data_type": "tcp_flow", "created_at": "ISO8601" }
3	GET	/prediction/tasks/{task_id}	Check async task status (model training or data generation).	n/a	{ "task_id": "uuid", "status": "queued|running|completed|failed"

					"result_tag": "string|null", "error": "string|null", "model_id": "uuid", }

4	POST	/prediction/tasks/generate	Generate synthetic data. Method: GAN (trained model) or EMULATION. Results stored in DB	{
				"generation_type": "GAN|EMULATION",
				"data_type": "tcp_flow|tcp_raw|lora_raw",
				"model_id": "uuid (required if GAN)",
				"source_query": { "filter": {...} },
				"config": {
				"count": 1000,
				"attacks": [...] (if EMULATION),
				"device_profile": "known|new"
				},
				"data_to_augment": {...} (opt)
				}
					{
					"task_id": "uuid",
					"status": "queued",
					"result_tag": "string|null"
					}

Notes:

- query – used to retrieve historical data. Returns batch of data that should be used for building model.
- authentication related fields were omitted for better clarity
- generated data is stored in database with result_tag for client retrieval

3.5.3. Deployment details

Containers

The prediction component consists of the following containers:

- Model builder – a container responsible for building generative models for network data prediction. It uses historical data to build the model. The model is stored in s3-based repository.
- Data generator – a container responsible for generating (predicting) new data. It uses generative model or historical data to achieve the goal.
- ID2T tool – a container wrapping the tool for injection network traffic in historical data.

They are defined using Dockerfiles, which can be found in the dedicated service directories. As an example, we show the Dockerfile of the main Prediction container:

```
FROM python:3.12-slim

ENV DEBIAN_FRONTEND=noninteractive \
    PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1 \
    PIP_NO_CACHE_DIR=1 \
    OPENBLAS_NUM_THREADS=1 \
    OMP_NUM_THREADS=1 \
    MKL_NUM_THREADS=1 \
    NUMEXPR_NUM_THREADS=1

RUN apt-get update \
    && apt-get install -y --no-install-recommends libgomp1 \
    && rm -rf /var/lib/apt/lists/*

# Create an user
# Seems like this one needs to 1000
ARG UID=1000
ARG GID=1000

# Create group and user by numeric IDs
RUN groupadd -g ${GID} -r usergroup \
    && useradd -r -u ${UID} -g ${GID} -d /home/user user

# Create home and app directories
RUN mkdir p /home/user /app \
    && chown ${UID}:${GID} -R /home/user /app

USER ${UID}
WORKDIR /app

# Set PATH and PYTHONPATH
ENV PYTHONPATH="$PYTHONPATH:/app"
ENV PATH="$PATH:/home/user/.local/bin"

COPY --chown=$UID:$GID requirements.txt .
RUN pip --no-cache-dir install --upgrade pip && pip install --no-cache-dir -
r requirements.txt

COPY --chown=$UID:$GID src src

ENTRYPOINT ["python", "-u", "src/app.py"]
```

As you can see, it will create a python environment and run the main app process as a non-root user.

Configuration

Each of the containers is being configured via a dedicated configuration file, which should be mounted in a container folder as a `config.yaml` file. Below you can see an example of the main prediction container configuration:

```
app:
  logging_level: INFO

elasticsearch:
  host: "@format {env[ELASTICSEARCH_HOST]}"
```

```
  username: "@format {env[ELASTICSEARCH_USERNAME]}"
  password: "@format {env[ELASTICSEARCH_PASSWORD]}"

  ssl:
    ca_certs: "@format {env[ELASTICSEARCH_CA_CERTS]}"
    verify_certs: true
```

Step-by-step guide to run prediction components locally

1. Clone the repository:

```
git clone https://gitlab.intra.miranda.onesource.pt/miranda/.git
```

2. Go to the component's main directory:

```
cd miranda/prediction
```

3. Set environment variables in the .env file:

```
ELASTICSEARCH_HOST=https://hostname:9200
ELASTICSEARCH_CA_CERTS=/path/to/ca.crt
ELASTICSEARCH_USERNAME=<username>
ELASTICSEARCH_PASSWORD=<password>
(others in the future)
```

4. Prebuild docker images:

```
docker compose -f docker-compose.yaml build
```

5. Run containers using docker compose:

```
docker compose -f docker-compose.yaml --env-file .env up
```

3.5.4. Individual component testing

Table 38. Component Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-PRED-01	Verify successful creation of GAN model	To be tested
TC-PRED-02	Verify synthetic data generation using GAN method and correct storage in database	To be tested
TC-PRED-03	Verify attack emulation (EMULATION type) injects attacks into source traffic and stores results	To be tested
TC-PRED-04	Verify API error handling for invalid requests	To be tested

Table 39. Test Case PREDICTION-01

Test Case ID	TC-PRED-01	Component	Prediction
Description	Verify successful creation of GAN model		
Tested by	NASK		

Associated Requirements	FR-A-1.12
Pre-condition(s)	Prediction module is deployed and running; Database contains historical LoRa traffic
Test steps	
1	Send POST request to /prediction/models to build generative model (GAN, LoRa data, new_traffic)
2	Verify HTTP response code is 202 Accepted
3	Verify response contains task_id (UUID format) and status: "queued"
4	Poll GET /prediction/tasks/{task_id} until status changes from "queued" to "completed"
5	Verify task response contains result_tag with model reference
6	Verify result contains traffic for new time period
7	Call GET /prediction/models/{model_id} and verify status: "ready"
Input data	JSON request body with model_type, data_type, prediction_type, data_query parameters (GAN, LoRa, new_traffic, query)
Results	Model successfully created and stored in repository
KPIs	Response time < 500 ms; Building model time < 1 min
Test Case Result	Pass / Failed

Table 40. Test Case PREDICTION-02

Test Case ID	TC-PRED-02	Component	Prediction
Description	Verify synthetic data generation using GAN method and correct storage in database		
Tested by	NASK		
Associated Requirements	FR-A-1.12		
Pre-condition(s)	Prediction module deployed and running; Trained GAN model for LoRa traffic exists		
Test steps			
1	Send POST request to /prediction/generate to generate synthetic network data (GAN, LoRa raw, model UUID)		
2	Verify HTTP response code is 202 Accepted		
3	Verify response contains task_id, status: "queued" and result_tag		
4	Poll GET /prediction/tasks/{task_id} until status: "completed"		
5	Query database using result_tag from response		
6	Verify database contains records tagged with result_tag		

7	Verify generated records have valid LoRa structure
Input data	JSON request body with model type, type of traffic, model UUID
Results	Synthetic traffic successfully created and stored in database
KPIs	Response time < 500 ms; Generating data time < 10 sec
Test Case Result	Pass / Failed

Table 41. Test Case-PREDICTION-03

Test Case ID	TC-PRED-03	Component	Prediction
Description	Verify attack emulation injects attacks into source traffic and stores results		
Tested by	NASK		
Associated Requirements	FR-A-1.12		
Pre-condition(s)	Prediction module deployed and running; Database contains benign TCP/IP raw traffic;		
Test steps			
1	Send POST request to /prediction/generate to generate synthetic network data (EMULATE, TCP/IP raw, query to benign data)		
2	Verify HTTP response code is 202 Accepted		
3	Verify response contains task_id and result_tag		
4	Poll GET /prediction/{task_id} until status: "completed"		
5	Query database using result_tag		
6	Verify result contains original benign traffic with injected attack patterns		
7	Verify attack labels are present in result metadata		
Input data	JSON request body with query		
Results	Synthetic traffic successfully created and stored in database		
KPIs	Response time < 500 ms; Generating data time < 10 sec		
Test Case Result	Pass / Failed		

Table 42. Test Case PREDICTION-04

Test Case ID	TC-PRED-04	Component	Prediction
Description	Verify API error handling for invalid requests		
Tested by	NASK		
Associated Requirements	FR-A-1.12		

Pre-condition(s)	Prediction module is deployed and running;
Test steps	
1	Send POST /prediction/models with missing required field (model_type); Verify HTTP 400 and error message indicates missing field
2	Send POST /prediction/models with invalid model_type; Verify HTTP 400 and error lists valid options
3	Send POST /prediction/generate with generation_type: "GAN" but without model_id; Verify HTTP 400 and error indicate model_id required for GAN
4	Send POST /prediction/generate with non-existent model_id; Verify HTTP 404 and error message
5	Send GET /prediction/tasks/{invalid_uuid}; Verify HTTP 404
Input data	Malformed/invalid JSON requests
Results	Invalid requests rejected with appropriate HTTP codes and error messages
KPIs	100% invalid requests correctly rejected
Test Case Result	Pass / Failed

3.5.5. Next steps and future development updates

As the next steps:

- the REST API will be provided and tested.
- ID2T tool will be integrated with the data-generator component.

3.6 Threat Feed Connector

The Threat Feed Connector is a modular, REST based threat intelligence aggregation service designed to ingest data from multiple external threat feeds (e.g. CVE, MISP, AlienVault), normalize them into a unified data model, and expose a centralized API for consumption by other MIRANDA platform components.

3.6.1. Prototype Description

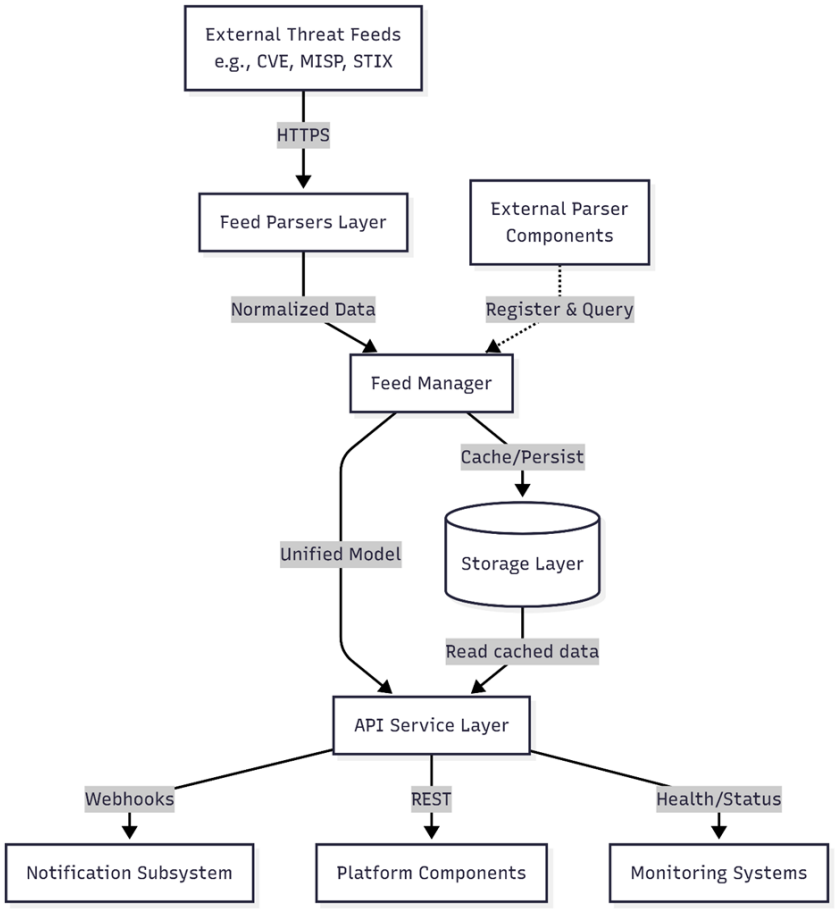

Figure 6. Threat Feed Connector internal architecture diagram

The connector is organised as a set of modular subsystems that together provide ingestion, normalisation, routing and notification capabilities. At a high level these include an API that exposes health, status, feed listing and query operations. A feed management layer routes queries to registered external workers, falling back to local parsers. Pluggable parsers implement feed-specific collection and normalisation logic. A component registry and heartbeat mechanism tracks external parser workers. A durable cache enables offline queries and deduplication. A unified data model presents consistent ThreatData objects to consumers. An outbound notification subsystem delivers webhook events. These pieces are intentionally decoupled so they can be replaced or scaled independently.

3.6.2. Communication interfaces

Table 43. Threat Feed Connector Communication interfaces

#	Method	REST Endpoint	Description	Request Body	Response Body
1	GET	/feeds	List registered feeds		{ "feeds": ["CVE", "MISP", "STIX", "AlienVault"] }
2	POST	/feeds/reload	Reload parser registry		{ "feeds": ["CVE", "MISP"], "status": "reloaded" }
3	GET	/data	Fetch normalized threat data (supports query params: `feed`, `keyword`, `fromDate`, `toDate`, `source`)		[{"id": "CVE-2025-1234", "source": "CVE", "title": "...", "description": "...", "published": "2025-01-15T00:00:00Z", "severity": "High", "references": ["..."],]
4	GET	/data/{feed}	Fetch data from specific feed		Similar to "/data"
5	POST	/registerComponent	Register external parser component	{"ip": "10.0.0.5", "port": 8080, "feed": "my-stix", "feedDescription": "...", "feedType": "stix"}	{"status": "registered", "feedID": "10.0.0.5:8080:my-stix"}
6	GET	/infoComponents	List registered components		[{"feedID": "10.0.0.5:8080:my-stix", "ip": "10.0.0.5", "port": 8080, "feed": "my-stix", "last_seen": "2025-09-08T12:00:00Z", "failure_count": 0,

					"unhealthy": false}]
7	DELETE	/components/{feedID}	Remove registered component		{"status": "deleted", "feedID": "..."}
8	GET	/components/{feedID}/fetch	Proxy fetch to component		[{"id": "...", "source": "...", ...}]
9	GET	/health	Service liveness check		{"status": "ok"}

3.6.3. Deployment details

The Threat Feed Connector can be deployed in three ways: local development setup, containerised deployment with Docker, or deployment on Kubernetes.

For Docker, build the docker image

```
docker build -t <image-name>:<tag> .
```

Run a single container with environment variables:

```
docker run -d \
  -p <host-port>:8000 \
  -e CVE_FEED_URL="<nvd-api-endpoint>" \
  -e CVE_FEED_TYPE="api" \
  -e STIX_ENABLED="true" \
  -e STIX_FEED_URL="<stix-feed-url>" \
  -e ALIENVAULT_ENABLED="true" \
  -e LOG_LEVEL="INFO" \
  -v <local-data-path>:/data \
  --name threat-feeds \
  <image-name>:<tag>
```

Alternatively, use Docker Compose for easier configuration management. A `docker-compose.yml` file is provided in the repository. Edit the file to configure image name, ports, enabled feeds, and environment variables, then deploy:

For Kubernetes, create namespace and secrets for sensitive configuration and for external feed integration, create secrets with credentials:

```
kubectl create namespace <namespace>

kubectl create secret generic <misp-secret-name> \
  --from-literal=MISP_URL=<misp-instance-url> \
  --from-literal=MISP_API_KEY=<api-key> \
  -n <namespace>
```

If using a private container registry, create an image pull secret with appropriate credentials.

Apply the Kubernetes the manifests from the repository, with the deployment, service and configmap manifests configuring the service with environment variables and secret references. The Ingress manifest exposes the service externally. Configure the hostname, TLS certificates, and ingress class according to the cluster setup.

3.6.4. Individual component testing

Table 44. Threat Feed Connector Component Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-TFC-01	CVE feed returns normalized data with keyword filter from a valid API	Achieved
TC-TFC-02	Feed integration with valid external APIs key	To be tested
TC-TFC-03	STIX feed parsing of indicator objects	Achieved
TC-TFC-04	Data storage and cached queries	Achieved

Table 45. Test Case Threat Feed Connector-01

Test Case ID	TC-TFC-01	Component	Threat Feed Connector
Description	Verify that the CVE feed parser can fetch data from NVD API, apply keyword filtering, and return normalized objects.		
Tested by	ONE		
Associated Requirements	FR-B-1.46		
Pre-condition(s)	Service running, with `CVE_FEED_URL` pointing to a valid API		
Test steps			
1	Send GET request: `/data?feed=CVE&keyword=ransomware`		
2	Verify response is a JSON array		
3	Verify each item has required fields: `id`, `source`, `title`, `description`, `published`, `severity`, `references`		
4	Verify at least one result matches the keyword "ransomware" in title or description		
Input data	Query parameters: `feed=CVE`, `keyword=ransomware`		
Results	200 OK, JSON with items		
KPIs	TBD		
Test Case Result	TBD		

Table 46. Test Case Threat Feed Connector-02

Test Case ID	TC-TFC-02	Component	Threat Feed Connector
Description	Verify that external feed parsers (MISP/AlienVault) can authenticate with valid API keys, fetch threat data, and return normalized objects.		
Tested by	ONE		
Associated Requirements	FR-B-1.46		
Pre-condition(s)	Service running, MISP instance accessible with valid API key configured via environment variables or Kubernetes Secrets, internet access to feed endpoints		
Test steps			
1	Send GET request: `/data?feed=MISP&keyword=malware`		
2	Verify each item has required fields: `id`, `source`, `title`, `description`, `published`, `severity`, `references`		
3	Verify at least one result matches the keyword "malware" in title or description		
4	Check logs for successful MISP authentication and data fetch		
Input data	Query parameters: `feed=MISP`, `keyword=malware`		
Results	HTTP 200 OK, JSON array with normalized ThreatData items from MISP, no authentication errors in logs		
KPIs			
Test Case Result	To be tested		

Table 47. Test Case Threat Feed Connector-03

Test Case ID	TC-TFC-03	Component	Threat Feed Connector
Description	Verify that the STIX feed parser can fetch STIX 2.1 bundles, parse indicator objects, and return normalized threat data with correct mapping of STIX properties.		
Tested by	ONE		
Associated Requirements	FR-B-1.46		
Pre-condition(s)	Service running, `STIX_ENABLED=true`, valid STIX 2.1 feed URL configured, internet access to STIX feed endpoint		
Test steps			
1	Send GET request: `/data/STIX`		
2	Verify each item has required fields: `id`, `source`, `title`, `description`, `published`, `severity`, `references`		
3	Send GET request: `/data?source=stored` (cached query)		

4	Verify response and that STIX properties are mapped to normalized schema (pattern → description, created → published)
Input data	Path parameter: `STIX`
Results	HTTP 200 OK, with JSON array with normalized ThreatData items parsed from STIX 2.1 bundle
KPIs	TBD
Test Case Result	Achieved

Table 48. Test Case Threat Feed Connector-04

Test Case ID	TC-TFC-04	Component	Threat Feed Connector
Description	Verify that fetched threat data is stored in the file-backed cache, deduplicate, and can be retrieved via cached queries without re-fetching from external sources.		
Tested by	ONE		
Associated Requirements			
Pre-condition(s)	Service running, at least one feed configured and operational, storage configured with write permissions on storage directory		
Test steps			
1	Send GET request: `/data?feed=CVE&keyword=ransomware` (normal fetch)		
2	Verify response contains threat data items		
3	Verify that storage file contains JSON array with fetched items		
4	Send GET request: `/data?source=stored` (cached query)		
5	Verify response returns previously stored items without re-fetching		
Input data	Query parameters: `feed=CVE`, `keyword=ransomware`, `source=stored` (cached)		
Results	HTTP 200 OK for both live and cached queries, storage file populated with deduplicated items, cached query returns data without external API calls		
KPIs	TBD		
Test Case Result	Achieved		

3.6.5. Next steps and future development updates

- Execute remaining test cases.
- Replace/improve from file-backed storage to PostgreSQL or Redis for distributed caching.
- Expand feed sources.

3.7 Sandboxing

The Sandboxing component is responsible for the management of isolated, reproducible test environments, where service deployments and cyber-defensive mechanisms can be safely evaluated. These sandboxes support the instantiation of full network and application stacks in controlled Kubernetes namespaces, enabling dynamic service deployment and interaction with security functions. Each sandbox is a fully isolated Kubernetes namespace with namespace-scoped RBAC controls, network isolation policies, and dedicated user credentials.

3.7.1. Prototype Description

The Sandboxing platform implements a two-tier architecture where the Northbound API exposes unified REST endpoints to external clients such as dashboards and command-line tools. This layer validates incoming requests, performs authentication and authorization checks through integration with Authentik[2] (an OAuth2/OIDC identity provider), and forwards validated operations to the Sandbox Manager backend. Importantly, the Northbound API does not interact directly with Kubernetes, all cluster operations are delegated to the Sandbox Manager component to maintain a clear separation of concerns and enable centralized policy enforcement.

The Sandbox Manager operates as the core orchestration engine responsible for all Kubernetes interactions. It manages the complete lifecycle of sandbox environments, including namespace provisioning with appropriate metadata annotations and labels, RBAC configuration to bind users to their namespaces, certificate generation for user-specific kubeconfigs, resource deployment through YAML manifests and Helm charts, and network isolation enforcement via NetworkPolicy objects. The Sandbox Manager operates with elevated cluster privileges using a configured admin kubeconfig to create namespaces and provision user credentials, while simultaneously enforcing strict isolation between sandboxes to prevent cross-tenant access.

The Sandboxing platform provides the following capabilities:

- Manage Sandbox Lifecycle: Create and delete isolated Kubernetes namespaces with automatic RBAC provisioning, network policy enforcement, and user credential generation. Each sandbox creation returns a namespace-scoped kubeconfig for immediate user access.
- Create and Manage Users: Administrative endpoints for creating user accounts in the identity provider, listing registered users, and managing user assignments to sandbox environments.
- Assign Users to Sandbox Environments: Grant existing users access to sandboxes for collaborative workflows, generating additional kubeconfigs scoped to the shared namespace, and revoke access when collaboration concludes.
- Deploy Applications using YAML Files: Apply arbitrary Kubernetes manifests including multi-document YAML files with built-in dry-run validation to detect configuration errors before actual deployment. Supports all standard Kubernetes resource types.

[2] https://goauthentik.io/.

- Deploy Applications using Helm Charts: Install, upgrade, list, and uninstall Helm releases within sandbox namespaces with configurable repository URLs, chart versions, and custom values files.
- List Resources of Sandbox Environments: Query all deployed resources within a namespace organized by type (Deployments, Services, Pods, Jobs, NetworkPolicies) with summarized status information including replica counts, readiness states, and IP addresses.
- Query Resource Status: Retrieve detailed status and specification information for specific resources identified by kind and name, providing comprehensive visibility into workload health and configuration.
- Delete Kubernetes Resources: Remove specific resources by kind and name, or delete all resources defined in a YAML manifest file, with RBAC enforcement ensuring users can only delete resources within their assigned namespaces.
- Apply Network Isolation Policies: Define and manage NetworkPolicy objects to control ingress and egress traffic patterns between pods and external endpoints. Default-deny policies are automatically applied on sandbox creation.
- Execute Attacks on Sandbox Environments: Deploy security testing workloads such as penetration testing tools, vulnerable applications, or attack simulation frameworks within isolated sandboxes for safe cyber-defensive mechanism evaluation.
- Execute Batch Jobs: Submit and monitor Kubernetes Job resources for computational tasks, with status tracking and log retrieval capabilities for completed or failed jobs.

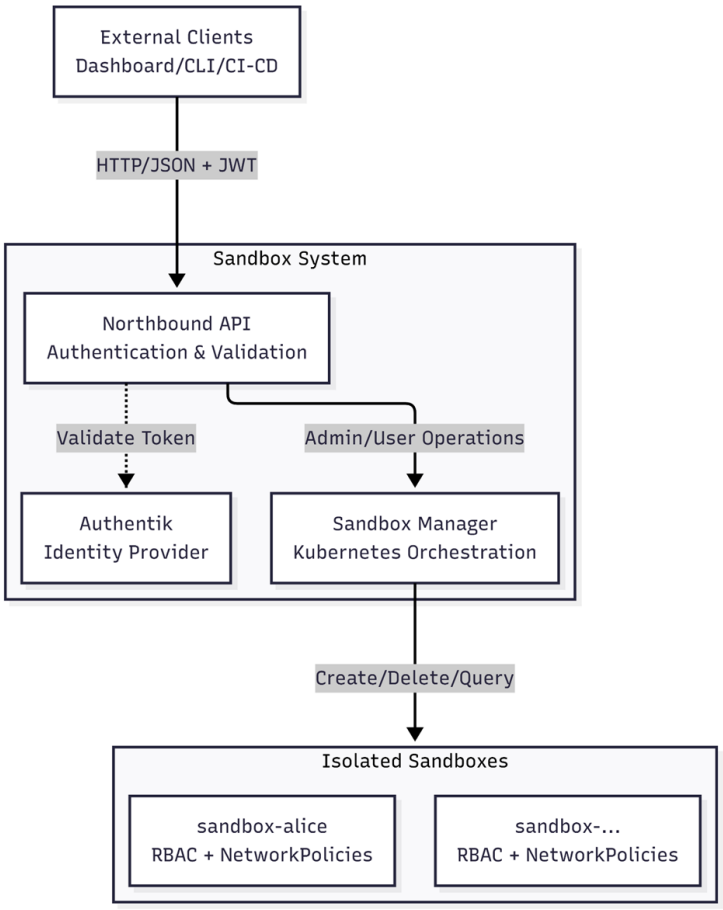

Figure 7. Sandbox internal architecture

3.7.2. Communication interfaces

Table 49. Sandbox Communication Interfaces

#	Method	REST Endpoint	Description	Request Body	Response Body
1	GET	/api/auth/login	Initiates OAuth2 Authentication Flow		Redirect to IAM Provider
2	GET	/api/auth/callback	OAuth2 callback endpoint. Processes authorization code and establishes session.	Code	Create session cookie and redirect to frontend
3	GET	/api/auth/logout	Terminates user session and revokes token.		Redirect to login page
4	GET	/api/auth/refresh	Refresh access token using refresh token		{ “access_token”: <access_token>, “expires_in”: 3600 }
5	GET	/api/auth/verify	Verifies validity of current access token.		{ “valid”: true, “username”: “alice” }
6	GET	/api/auth/userinfo	Retrieves detailed information about authenticated user (including roles)		{ “username”: <username>, “email”: <email>, groups: [“sandbox-admin”]}
7	POST	`/api/admin/sandbox/create`	Creates a new sandbox environment (namespace) with RBAC roles and Authentik groups. **Requires**: `sandbox-admin` role	{"namespace_name":"team1-dev"}	{"status": "success",message":" Namespace team1-dev created"}
8	GET	`/api/admin/sandbox/list`	Retrieves list of all managed sandbox environments. **Requires**: `sandbox-admin` role	-	{"status":"success","namespaces: ["team1-dev", "team2-prod"]}

9	DELETE	`/api/admin/sandbox/delete`	Deletes sandbox environment and revokes all user access. **Requires**: `sandbox-admin` role	{“namespace_name":"team1-dev" }	{“status": "success”, “message":"Namespace team1-dev deleted”}
10	PUT	`/api/admin/sandbox/{username}/update-access`	Updates user permissions across multiple sandbox environments. **Requires**: `sandbox-admin` role	{“username": "alice","sandboxes_permissions":[{"namespace_name":"team1-dev", "roles":["view-pods", "get-pods"] }]}	{"status":"completed”,“updated_namespaces":["team1-dev"],“requested_groups":["sandbox_view_team1-dev"]}
11	GET	`/api/renew`	Regenerates kubeconfig for current authenticated user. **Requires**: `sandbox-user` role		{"status": "success","messag":" Kubeconfig renewed”, “kubeconfig": "LS0tLS1CRU...”}
12	GET	`/api/kubeconfig`	Retrieves existing kubeconfig for current user. **Requires**: `sandbox-user` role		{"status": "success", "message": "Kubeconfig retrieved", "kubeconfig": "LS0tLS1CRU..." }
13	GET	`/api/sandboxes`	Lists all namespaces accessible to current user. **Requires**: `sandbox-user` role		{"status": "success”,“username":"alice","namespaces": ["team1-dev"],"namespace_count": 1 }
15	POST	/api/admin/users/create	Creates new user in Authentik with sandbox access and kubeconfig. Requires: sandbox-admin role	{"username": "johndoe", "password": "SecurePass123!", "is_active": true, "sandboxes_permissions": [{"namespace_name":"team1-dev","roles": ["view_pods"] }]}	{"pk":123,"username":"johndoe", "email":null,"is_active":true,"message:"User created successfully"}
16	GET	/api/admin/users/{username}	Retrieves detailed user information from Authentik.		{"pk":123, "username":"johndoe","email":"john@example.com",

			Requires: sandbox-admin role		"is_active":true, "groups":["sandbox_view_team1-dev"]}
17	GET	/api/admin/users	Lists all users from Authentik. Requires: sandbox-admin role		{"status":"success", "users":[{"pk":123,"username":"johndoe", "is_active": true }], "count": 1}
18	DELETE	/api/admin/users/{username}	Deletes user from Authentik and revokes all sandbox access, certificates, and RoleBindings. Requires: sandbox-admin role		{"status": "success","message": "User johndoe deleted","deleted_bindings": 3}
19	POST	/api/helm/deploy	Deploys Helm chart to namespace. Requires: create role	{"namespace":"team1-dev","release_name":"redis","chart":"bitnami/redis", "repo_url":"https://charts.bitnami.com/bitnami","version":"18.0.0","values":{"replica":{"count": 1 } }}	{"status": "completed", "message":"Helm chart deployed successfully"}
20	DELETE	/api/helm/uninstall	Uninstalls Helm release from namespace. Requires: delete role	{"namespace":"team1-dev","release_name":"redis"}	{"status":"success", "message": "Release 'redis' uninstalled"}
21	POST	/api/helm/upgrade	Upgrades existing Helm release. Requires: update role	{"namespace":"team1-dev","release_name":"redis","chart":"bitnami/redis","version":"18.1.0","values": {"replica": { "count": 2 } }, "atomic":true,"reuse_values": false}	{"status": "success", "message": "Upgraded release 'redis'"}
22	POST	/api/helm/list	Lists Helm releases in namespace or all namespaces. Requires: list role	{"namespace":"team1-dev","all_namespaces": false}	{"status": "success", "releases":[{ "name":"redis", "namespace": "team1-dev", "revision":1,"status":"deployed","chart":"re

					dis-18.0.0","app_version": "7.2.0" }]}
23	GET	/api/resources/{namespace}	Lists all workload resources (Deployments, Services, Pods, etc.) in namespace. Requires: list role		[{"kind": "Deployment", "name":"api","status":{"availableReplicas": 1 } }]
24	GET	/api/status/{namespace}/{kind}/{name}	Retrieves detailed status of specific Kubernetes resource. Requires: get role		{"namespace_name": "team1-dev", "kind": "Deployment", "name":"api", "status":{"availableReplicas":1, "replicas": 1 }}
25	DELETE	/api/resources/{namespace}/{kind}/{name}	Deletes specific Kubernetes resource from namespace. Requires: delete role		{"status": "deleted", "details": {"kind":"Deployment", "name": "api" }}
26	POST	/api/yaml/deploy	Applies YAML manifests to namespace. Supports multi-document YAML. Requires: create role	{"namespace_name": "team1-dev", "yaml_files": "apiVersion: v1\nkind: Pod\nmetadata:\n name: test-pod\nspec:\n containers:\n - name: nginx\n image: nginx"}	{“status": "completed","results":[{"document":1, "status": "applied", "message":"Pod created" }]}
27	DELETE	/api/yaml/delete	Deletes resources defined in YAML manifests. Requires: delete role	{"namespace_name":"team1-dev", "yaml_files": [{"kind":"Pod", "name":"test-pod" }]}	{"status": "completed", "results":[{"document": 1, "status": "deleted","message": "Pod deleted" }]}
28	POST	/api/networkpolicy/apply	Applies NetworkPolicy to namespace for traffic isolation. Requires: create role	{"namespace_name": "team1-dev", "policy": { "apiVersion": "networking.k8s.io/v1", "kind": "NetworkPolicy", "metadata": { "name": "allow-	{"results":[{ "policy":1, "status": "success", "message": "NetworkPolicy applied"}]}

				same-namespace" }, "spec": { "podSelector": {}, "policyTypes": ["Ingress"], "ingress": [{ "from": [{"podSelector": {}}] }] } }}	
29	DELETE	/api/networkpolicy/delete	Deletes NetworkPolicy from namespace. Requires: delete role	{"namespace_name": "team1-dev", "name": "allow-same-namespace"}	{"status": "success", "message": "Deleted NetworkPolicy 'allow-same-namespace' from namespace 'team1-dev'"}
30	POST	/api/jobs/stream	Creates Kubernetes Job and streams logs in real-time. **Requires**: `watch` and `create` roles	{"namespace_name":"team1-dev", "job_yaml":"apiVersion: batch/v1\nkind: Job\nmetadata:\n name: hello-job\nspec:\n template:\n spec:\n containers:\n - name: hello\n image: busybox\n command: [\"echo\", \"Hello\"]\n restartPolicy: Never"}	Streaming text response (logs)
31	GET	/api/security-enablers	Lists available security integrations (e.g., DeepGuardian). Requires: `list` role		```[{"name": "DeepGuardian", "url": "https://deepguardian.example.com" }]```
32	GET	/	Root endpoint providing API identification.		{“message": "Northbound API for Sandbox"}
33	GET	/health	Health check endpoint for monitoring and load balancers.		{"status": "healthy", "service":"northbound-api"}

3.7.3. Deployment details

NorthboundAPI and SandboxManager can be containerized into a Docker image and push to container repository, running following commands:

```
--- build the NorthboundAPI Image ---
docker build -f src/northboundAPI/Dockerfile -t
  registry.onesource.pt/miranda/sandbox/n-api:latest .
--- push the image to registry ---
docker push registry.onesource.pt/miranda/monitoring-
  dashboard/dashboard:latest
--- build the SandboxManager Image ---
docker build -f src/sandboxManager/Dockerfile -t
  registry.onesource.pt/miranda/sandbox/sandboxmanager:latest .
--- push the image to SandboxManager ---
docker push registry.onesource.pt/miranda/sandbox/sandboxmanager:latest
```

In non-local environments, NorthBoundAPI, SandboxManager and IAM Provider (Authentik) are deployed on Kubernetes Cluster using Helm Package present on repository.

Pre-Modifications before deploying using Helm:

- Update the `k8s/sandboxmanager-kubeconfig-secret.yaml` with Kubeconfig of cluster of sandbox environments.

The deploy using Helm Package can be achieved by running existing script on repo.

```
./k8s/install.sh
```

This Helm Package is divided into 2 charts:

- Deployment of NorthboundAPI (with containerized image) as well as creation of service and ingress to access externally the NorthboundAPI
- Deployment of SandboxManager (with containerized image) as well as creation of service.
- Deployment of IAM Provider (Authentik)

Configurations of IAM Provider and IAM Provider – NorthboundAPI Connectivity (currently without automation):

1) Login to Authentik Dashboard (authentik.intra.miranda.onesource.pt) using admin credentials
   a) Username: “akadmin”
   b) Password: get password from “kubectl -n sandbox get secret authentik-secrets -o jsonpath="{.data.AUTHENTIK_BOOTSTRAP_PASSWORD}" | base64 -d“
2) Apply Blueprints Configurations via Authentik Dashboard – Navigate to Customization -> Blueprints -> Create. Import blueprints presented on “k8s/helm/charts/dependencies/blueprints”:
   a) 01-sandbox-users.yaml -> create “sandbox-admin” group and user
3) Set password for “sandbox-admin” on “directory/users”
   a) 02-sandbox-application.yaml -> create provider and application to authentication
   b) 04-sandbox-logout-flow.yaml-> create customizable logout flow
4) Update configurations on NorthboundAPI to communicate with IAM Provider (Authentik)
   a) Get and update OAuth2 Client Secret (to validate provided tokens)

i) Copy Client Secret from Applications -> Sandbox Manager -> Provider
ii) Configure Client Secret of Authentik on NorthboundAPI -> kubectl -n sandbox patch secret northboundapi-secret --type='merge' -p='{"stringData":{"OAUTH_CLIENT_SECRET":"<CLIENT_SECRET_FROM_AUTHENTIK>"}}'

b) Get and update Authentik API Token (to manage Authentik entities)
i) Create token on "Directory->Tokens and App passwords -> Create Token"
ii) Set identifier (`northboundapi-token`, user: `sandbox-admin`, expiration as needed)
iii) Configure API Key of Authentik on NorthboundAPI -> kubectl -n sandbox patch secret northboundapi-secret --type='merge' -p='{"stringData":{"AUTHENTIK_API_TOKEN":"<AUTHENTIK_API_TOKEN>"}}'

5) Apply new configurations on NorthBoundAPI kubectl -n sandbox rollout restart deployment/northboundapi)

3.7.4. Individual component testing

Table 50. Sandbox Components Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-Sandbox-01	Sandbox creation with namespace provisioning and RBAC configuration.	Achieved
TC-Sandbox-02	Create Sandbox User	Achieved
TC-Sandbox-03	Assign User to Sandbox Environment	Achieved
TC-Sandbox-04	YAML and Helm manifest deployment with multi-document support and validation	Achieved
TC-Sandbox-05	Resource management operations (list, query, delete)	Achieved
TC-Sandbox-06	Instantiation of Attacks	Achieved
TC-Sandbox-07	Multi-user namespace isolation and cross-namespace access control	Achieved

Table 51. Test Case Sandbox-01

Test Case ID	TC-Sandbox-01	Component	Sandboxing
Description	Validate sandbox creation workflow including namespace provisioning and RBAC configuration via Northbound API.		
Tested by	ONE		
Associated Requirements	SR-SPL-07		

Pre-condition(s)	Northbound API accessible and authenticated with Admin role
Test steps	
1	Send a POST request to the `api/admin/sandbox/create` endpoint with payload {“namespace_name”:””}
2	Verify HTTP 200 response containing success status and message
3	Validate kubeconfig capabilities and permissions
Input data	{“namespace_name”:””}
Results	API response body
KPIs	
Test Case Result	Pass

Table 52. Test Case Sandbox-02

Test Case ID	TC-Sandbox-02	Component	Sandboxing
Description	Validate user creation through the Northbound API		
Tested by	ONE		
Associated Requirements	SR-SPL-07		
Pre-condition(s)	Northbound API accessible and authenticated with Admin role		
Test steps			
1	Send a POST request to the `api/users/` endpoint with the user data in the request body		
2	Verify HTTP 200 response		
3	Validate that the user has been successfully created in the system, listing users (request `api/users).		
Input data	User details: {username: ‘’, password:””, is_active: true, sandboxes_permissions=[{“namespace_name”: “”, roles: [“view-pods”]]}		
Results	API response created		
KPIs			
Test Case Result	Pass		

Table 53. Test Case Sandbox-03

Test Case ID	TC-Sandbox-03	Component	Sandboxing
Description	Validate assigning a user to one or more sandbox environments (namespaces) with specific roles using the Northbound API.		

Tested by	ONE
Associated Requirements	FR-B-1.40, SR-SPL-07
Pre-condition(s)	- Northbound API is accessible and authenticated with Admin role. - User and sandbox exist.
Test steps	
1	Send a PATCH request to the `api/admin/sandbox/<username>/update-access` with assignment payload.
2	Verify HTTP 200 response with permissions
Input data	Assignment: {username: “alice”, sandboxes_permissions: [{“namespace_name”:””, “roles”: get-pods}]
Results	API response body
KPIs	
Test Case Result	Pass

Table 54. Test Case Sandbox-04

Test Case ID	TC-Sandbox-04	Component	Sandboxing
Description	Validate YAML and Helm manifest deployment with multi-document support and validation via Northbound API.		
Tested by	ONE		
Associated Requirements	FR-B-1.40, NON-FR-B-1.44		
Pre-condition(s)	- Northbound API is accessible and authenticated user has appropriate roles (e.g, ‘create’) in the target namespace.		
Test steps			
1	Send a POST request to `/yaml/` endpoint with multi-document YAML manifest.		
2	Verify HTTP 200 response confirming YAML deployment		
3	Send a POST request to `/helm/` endpoint with Helm chart details.		
4	Verify HTTP 200 response confirming Helm chart installation.		
5	List the resources and verify their existence by sending a GET request to the /api/resources endpoint.		
Input data	Multi-document YAML file with Deployment, Service, and ConfigMap resources. HELM Deployment.		
Results	Screenshots, descriptions, etc.		
KPIs			

Test Case Result	Pass

Table 55. Test Case Sandbox-05

Test Case ID	**TC-Sandbox-05**	**Component**	**Sandboxing**
Description	Validate resource management operations (list, query status, delete) via Northbound API.		
Tested by	ONE		
Associated Requirements	FR-B-1.40		
Pre-condition(s)	- Northbound API is accessible and authenticated user has appropriate roles ('list', 'get', 'delete') in the target namespace. - Existing resources in the sandbox.		
Test steps			
1	Send a GET request to `/resources/{namespace}` endpoint to list all resources.		
2	Verify HTTP 200 response containing a list of allowed resources (Pods, Deployments, Services, etc.).		
3	Send a DELETE request to `/resources/{namespace}/{kind}/{name}` to delete a resource.		
4	Verify HTTP 200 response confirming successful deletion		
5	Verify if deleted resource no longer exists, listing all resources		
Input data	Namespace name		
Results	Screenshots, descriptions, etc.		
KPIs	TBD		
Test Case Result	Pass		

Table 56. Test Case Sandbox-06

Test Case ID	**TC-Sandbox-06**	**Component**	**Sandboxing**
Description	Validate Attack (Job) execution with streaming of logs via Northbound API.		
Tested by	ONE		
Associated Requirements	FR-B-1.40, SR-SPL-08		
Pre-condition(s)	- Northbound API is accessible and authenticated users have job execution permissions in the target namespace (roles create-job, watch-pod, watch-pod, create-pod /all-permissions)		
Test steps			
1	Send a POST request to `/jobs/stream` endpoint with job definition and target namespace.		

2	Validate that logs are received in real-time as the job executes.
4	Confirm job completion status.
Input data	Target namespace. Job manifest with command and resource specifications.
Results	API responses, job creation confirmation, streaming logs output, and job completion status.
KPIs	TBD
Test Case Result	Pass

Table 57. Test Case Sandbox-07

Test Case ID	**TC-Sandbox-07**	**Component**	**Sandboxing**
Description	Validate multi-user namespace isolation and cross-namespace access control using Northbound API.		
Tested by	ONE		
Associated Requirements	FR-B-1.40, SR-SPL-07		
Pre-condition(s)	- Northbound API is accessible and authenticated with Admin role. - Multiple users and namespaces exist.		
Test steps			
1	1. Assign User A to Namespace A with specific roles.		
2	Assign User B to Namespace B with specific roles		
3	Attempt to access Namespace B as User A (should be denied)		
4	Attempt to access Namespace A as User B (should be denied)		
5	Grant User A cross-namespace access to Namespace B.		
6	Verify User A can now access Namespace B with the assigned roles.		
Input data	Usernames: `userA`, `userB` Namespaces: `namespaceA`, `namespaceB`		
Results	Screenshots, descriptions, etc.		
KPIs	TBD		
Test Case Result	Pass / Failed		

3.7.5. Next steps and future development updates

- Confirmation of instantiation of resources
- Automatic and dynamic configurations of IAM Provider
- Add more security validation on the requested manifests

3.8 GUI – User Interface & Visualisations

3.8.1. Logstail Platform GUI extension

3.8.1.1. Prototype Description

The Context Discovery GUI provides a visualisation of the runtime environment by mapping services, workloads, and infrastructure into a single, connected graph. It discovers entities such as hosts, virtual machines, containers, pods, applications, networks, and external peers, then links them using observed relationships like hosting, control, and network communication. The result is a living topology that shows how components are connected and interacting. This makes complex, distributed environments easier to reason about by turning large amounts of operational data into an intuitive visual model as shown by the figure below. By following edges across the graph, an operator can understand which components depend on others, where traffic flows, and how control is exercised across layers of the stack.

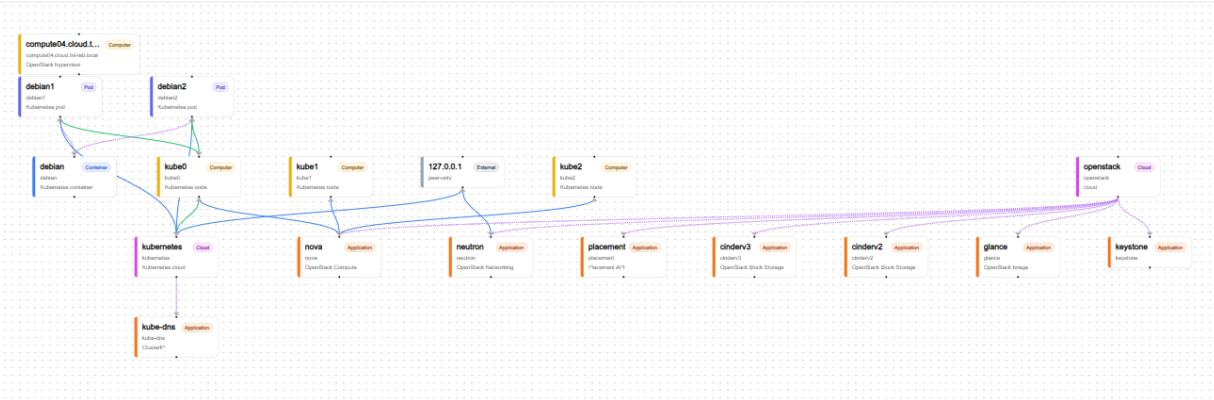

Figure 8. GUI Graph

The table view complements the graph by presenting the same discovered context in a structured, searchable format. Each row represents a discovered service or entity, classified by kind (for example, external peer, application, or compute resource) and enriched with identifying metadata such as hostname and description. This view enables efficient filtering, sorting, and scanning of the environment, making it easier to locate specific components or focus on internal services while retaining visibility of external peers where needed. In addition to identification, the table summarizes observed relationships and security-relevant signals through aggregated metrics. Columns such as edges, control, hosting, packet flow, and protect provide a concise snapshot of how each entity participates in the environment and where interactions or policies are present. Action controls allow operators to pivot directly from discovery into investigation or containment, supporting a smooth transition from high-level visibility to focused analysis.

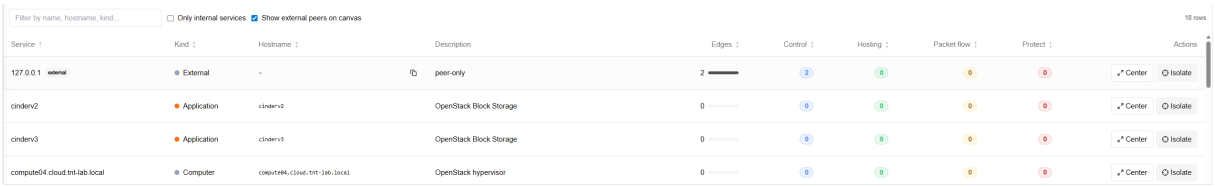

Figure 9. GUI Analysis

Requirements:

- FR-A-1.2 - Threat Analysis: It is necessary that the platform where the tool for threat modeling and analysis runs supports the SWB dialect of Prolog, used for the definition of the model and rules of the analysis.
- FR-A-1.8 - Displaying service context graphs: The system must be able to display interactive service context graphs that show the services, security functions, relationships, and security attributes.
- FR-A-1.9 - Displaying vulnerabilities and threats: The system must be able to show vulnerabilities and threats associated with each service/relationship in the service context graph.
- FR-A-1.10 - Navigating service context graphs: The system should be able to link services, relationships, and security attributes in the service context graphs to external configuration and monitoring tools.
- FR-B-1.3 - Context discovery: The system must automatically discover digital services chains and security functions.
- FR-B-1.5 - Context description: The Context must abstract the service topology as a graph which nodes are different types of services and which links are different kinds of relationships.
- FR-B-1.6 – Service types: The list of service types in Context description must include: cloud infrastructure, application, LoRaWAN, IoT device, WiFi, fiber network.
- FR-B-1.7 – Service relationships: The list of service relationships in Context description must include: hosted/running on, controlled by, VPN, packet flow, API.
- FR-B-1.10 – Dynamic context discovery: The Context must be continuously updated
- FR-B-1.13 – Context discovery implementation: The Context discovery implementation must support Kubernetes, Azure, OpenStack, ChirpStack.
- FR-B-1.14 – Context changes detection: Any change in the topology, composition, security properties, and vulnerabilities must be promptly detected and notified to relevant processes.

The Attack Paths GUI component provides a visual representation of how adversary behaviour can progress through an environment by chaining together identities, assets, artifacts, and infrastructure into a coherent attack sequence. It models attack paths as ordered graphs, where nodes represent relevant entities and edges represent observed or inferred relationships such as authentication, communication, staging, or exfiltration. By aligning these relationships with ATT&CK tactics, the interface shows not just isolated events, but how individual actions combine into a meaningful attack narrative from initial access through to objective.

In addition to visualization, the GUI supports exploration and analysis through filtering, enrichment, and interaction. Users can scope paths by entity type or tactic, inspect

relationships and supporting evidence, and understand where controls or mitigations may break the chain. This allows defenders to reason about attacker movement, identify critical choke points, and assess detection or prevention coverage, turning raw telemetry and inferred facts into an actionable view of adversary risk and potential impact.

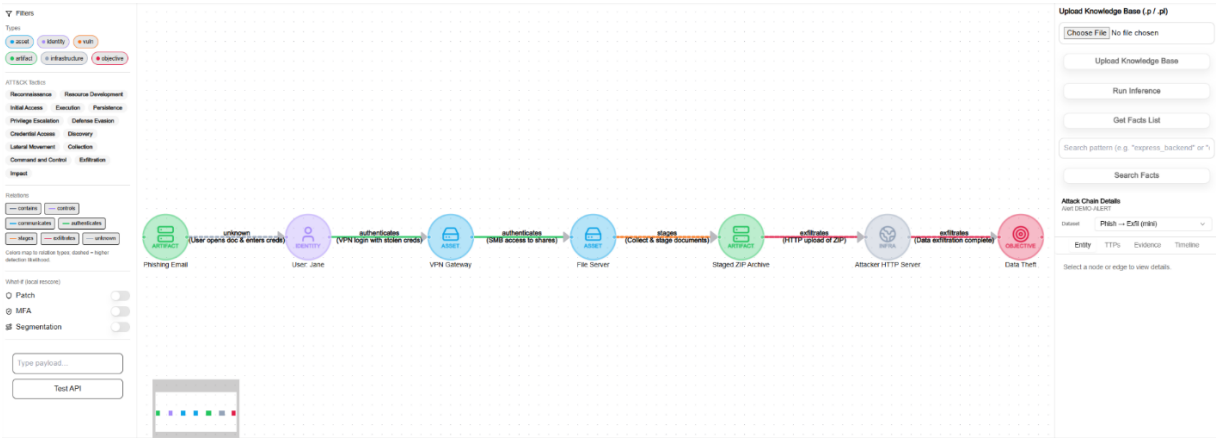

Figure 10. GUI exploration

The Data Handling Pipeline GUI provides a centralized interface for configuring, monitoring, and managing data pipelines that move information from sources through optional transformations to defined sinks. It presents a high-level operational overview, showing pipeline status, source and sink counts, and runtime activity, allowing users to quickly assess whether pipelines are running, idle, or stopped. The main table view lists existing pipelines with key attributes such as source, sink, and status, and supports searching, bulk actions, and lifecycle control, making it easier to operate pipelines at scale.

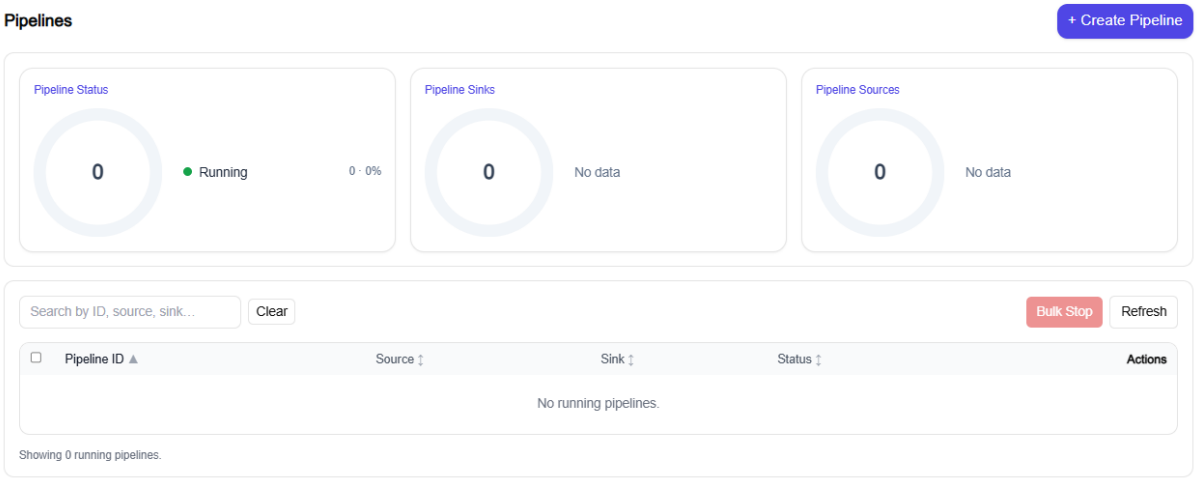

Figure 11. GUI Monitoring

In addition to monitoring, the GUI supports guided pipeline creation through a step-by-step workflow. Users define pipeline metadata, configure data sources, apply transformations, and select output sinks before reviewing and starting execution. This structured approach reduces configuration errors and enforces consistency, while still allowing flexibility in how data is ingested and processed.

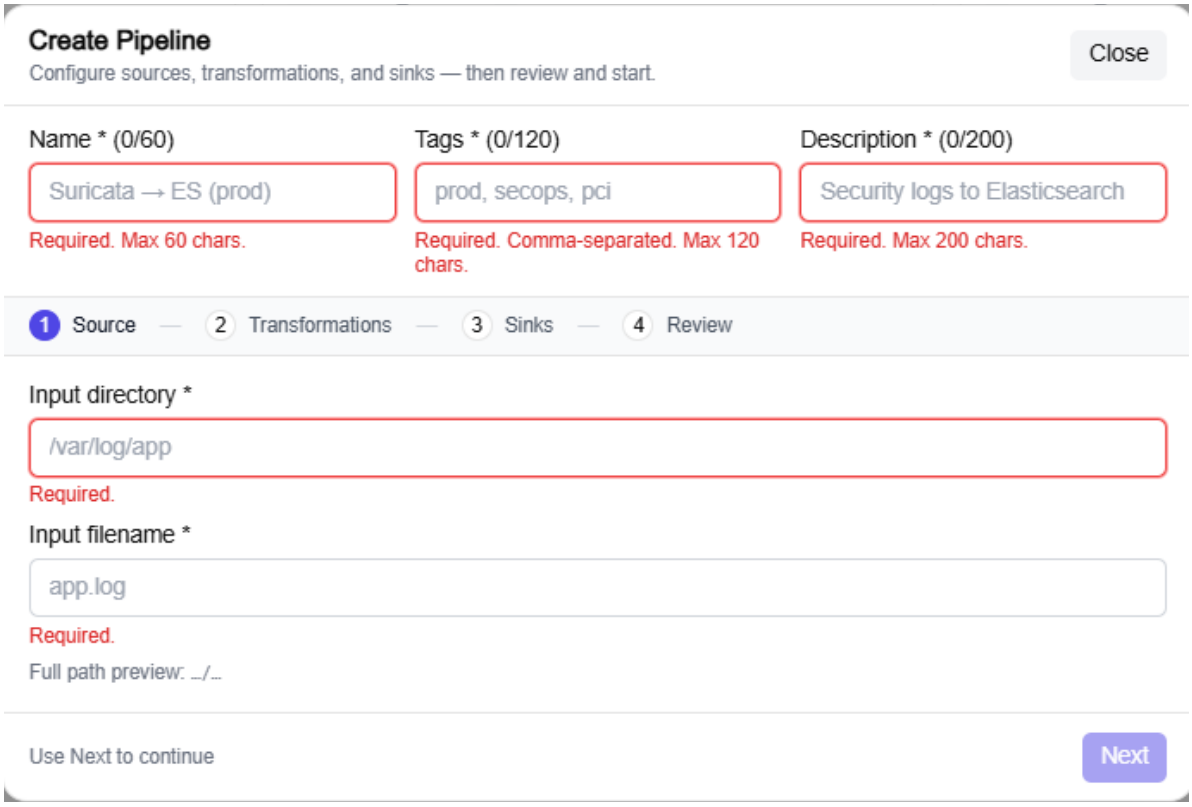

Figure 12. GUI Pipeline

Overall, the Data Handling Pipeline GUI bridges configuration and operations, enabling teams to reliably define data flows and maintain visibility into how data moves through the system.

3.8.1.2. Communication interfaces

The Logstail Platform GUI extensions (Context Discovery, Attack Paths, and Data Handling Pipeline components) are implemented as pure presentation-layer modules within the Logstail Platform. As such, they do not expose external communication interfaces, APIs, or standalone endpoints.

All interactions required for visualization, exploration, and user interaction are internally mediated by the Logstail Platform runtime. Data consumed by the GUI components is provided through platform-managed internal services, ensuring controlled access to contextual, analytical, and operational information without direct external connectivity.

This design choice enforces:

- A clear separation between presentation and backend logic
- Centralized access control and policy enforcement
- Reduced attack surface by avoiding direct external interfaces

From an integration perspective, the GUI extensions rely exclusively on platform-level abstractions, allowing the underlying data sources, analytics engines, and correlation mechanisms to evolve independently without impacting the user interface contract.

3.8.1.3. Deployment details

The Logstail Platform GUI extensions are deployed as part of the broader Logstail Platform ecosystem and follow the same containerized deployment principles as the core platform components.

Deployment artifacts include:

- Container definitions aligned with the platform's standardized build and runtime environment
- Configuration assets that define runtime behaviour, feature enablement, and environment-specific parameters
- Orchestration descriptors enabling consistent deployment across development, testing, and production environments

To protect proprietary platform intellectual property, detailed container specifications, build instructions, and configuration schemas are not exposed in this deliverable. From a conceptual standpoint, the deployment approach emphasizes:

- Reproducibility across environments
- Isolation between components
- Compatibility with cloud-native and on-premise execution models

The GUI extensions are designed to be deployed either alongside backend services or as part of an integrated platform distribution, depending on operational requirements.

3.8.1.4. Individual component testing

During the prototyping phase, the Context Discovery, Attack Paths, and Data Handling Pipeline GUI components were validated through manual functional testing using representative and mocked datasets.

Testing activities focused on:

- Correct rendering of graphs, tables, and interactive elements
- Validation of navigation flows and user interactions
- Consistency between different visual representations of the same contextual data
- Stability and responsiveness under varying data volumes

Mocked data was used to simulate realistic operational scenarios, including complex service topologies, multi-step attack paths, and multiple concurrent data pipelines. This approach allowed early validation of usability, visual clarity, and analytical workflows without dependency on backend engine availability.

The results confirmed that the GUI components correctly support their intended exploratory and analytical use cases at the prototype level.

3.8.1.5. Next steps and future development updates

The next phase of development will focus on tight integration with the Logstail Platform backend engines and underlying discovery, analytics, and data handling mechanisms.

Planned evolution includes:

- Replacing mocked inputs with live platform data streams
- Enabling real-time updates to context graphs and attack paths
- Enhancing visual enrichment using dynamically computed security attributes, risks, and confidence indicators
- Improving scalability and performance for large, highly dynamic environments

Future iterations will also explore advanced interaction capabilities, such as guided investigations, contextual recommendations, and deeper cross-linking between discovery, threat analysis, and operational response workflows.

These enhancements will further strengthen the GUI extensions as first-class analytical interfaces within the Logstail Platform, supporting both exploratory situational awareness and focused security decision-making while preserving the platform's architectural integrity and intellectual property boundaries.

3.8.2. Prototype Description

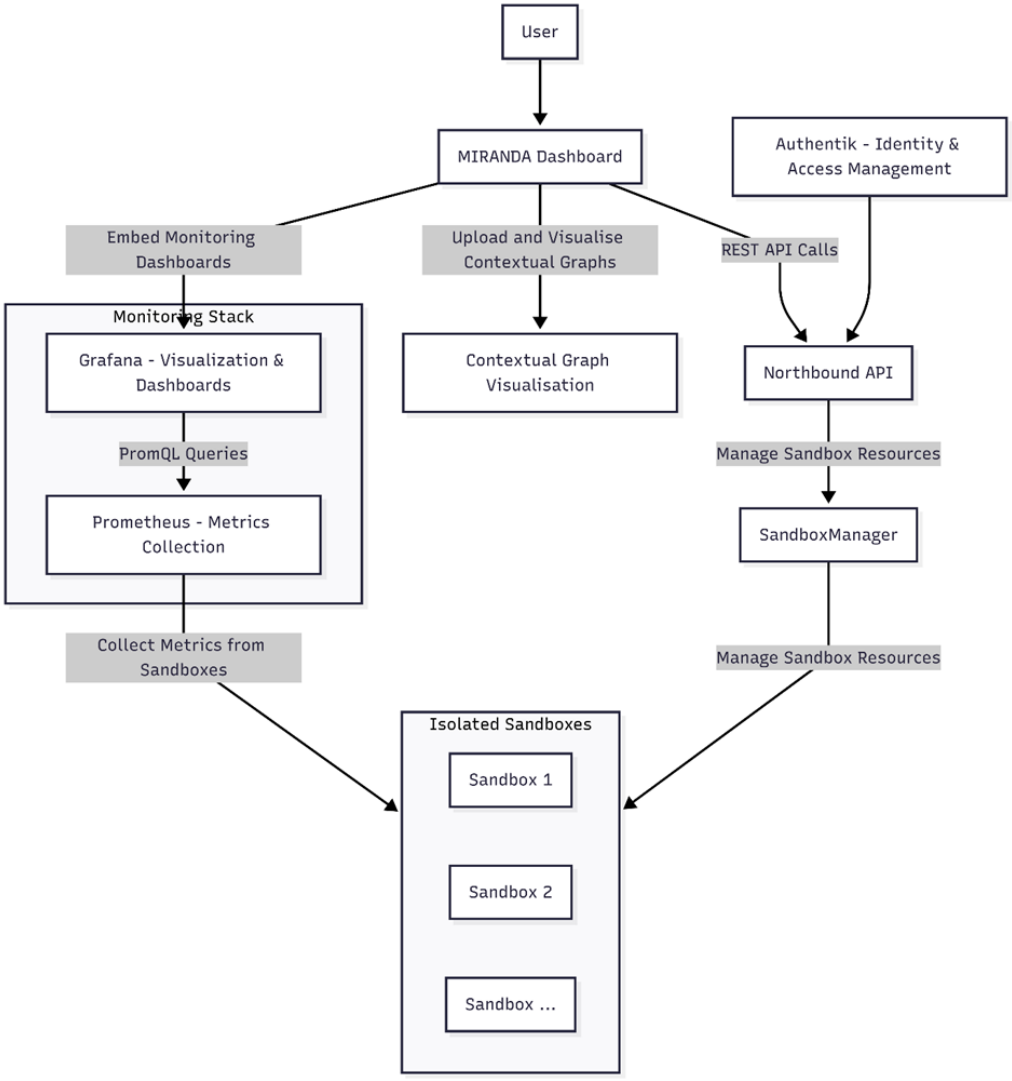

Figure 13. GUI Interfaces

Functionalities

- Collect real-time metrics from Sandbox Environments
- Display real-time metrics from Sandbox Environments
- Visualise and upload Context Graph based on CTXD Spec (4.2)
- Management of Sandbox Environments
- Login

3.8.3. Deployment details

The dashboard can be tested locally running:

```
npm install (install node packages)
npm run dev -- --port 3000
```

The dashboard must run on port 3000 to communicate with the NorthboundAPI (which is responsible for authentication and manage sandbox environments).

The dashboard can be containerized into a Docker image and push to container repository, running following commands:

```
cd miranda-dashboard
--- build the image ---
docker build -t registry.onesource.pt/miranda/monitoring-
dashboard/dashboard:latest
--- push the image to registry ---
docker push registry.onesource.pt/miranda/monitoring-
dashboard/dashboard:latest
```

In non-local environments, dashboards and monitoring stack are deployed on Kubernetes Cluster using Helm Package present on repository. The deploy using Helm Package can be achieved by these commands.

```
helm dependency update ./k8s/helm/charts/monitoring (install monitoring stack
    dependencies)
helm upgrade -i monitoring-dashboard ./k8s/helm --namespace monitoring-
    dashboard
```

Dashboard is accessible at: https://dashboard.intra.miranda.onesource.pt.

Grafana is accessible at: https://grafana.intra.miranda.onesource.pt.

This Helm Package is divided into 2 charts:

- Deployment of MIRANDA Dashboard (with containerized image) as well as creation of service and ingress to access externally the MIRANDA Dashboard
- Monitoring Stack, which is responsible for collecting, aggregating, and visualizing metrics of cluster, including Sandbox Environments.

In a way to customize dashboards of metrics on Grafana (observability visualization tool), dashboards can be added/updated on folder “k8s/helm/monitoring/dashboards”)

Helm is configurable to use HTTPS configurations. To configure TLS configurations, it is necessary to update “./k8s/helm/templates/tls-secret.yaml” to configure certificate configurations. It is necessary to encode .crt on base64 and .key on base64 and update the secret template accordingly.

3.8.4. Individual component testing

Table 58. Individual Component testing

Dashboard: Test Cases		
Test Case ID	**Description**	**Result**
TC- UI - 01	Upload and visualise CTX specification within dashboard	Achieved
TC-UI- 02	Visualise metrics of Sandbox Environments within Dashboard	Achieved
TC-UI-03	Login	Achieved

Table 59. Test Case UI-01

Test Case ID	**TC- UI - 01**	**Component**	**Dashboard**
Description	Verify that the user can upload a CTX (Context Graph) specification file and visualize its contents within the dashboard.		
Tested by	ONE		
Associated Requirements	FR-A-1.8, FR-B-1.6, FR-B-1.7, FR-B-1.46		
Pre-condition(s)	• Dashboard application is up and running • User is authenticated and has access to the dashboard • CTX specification file is available		
Test steps			
1	Navigate to the Context Graph section of the dashboard.		
2	Click the "Upload" button for CTX specification.		
3	Description of steps		
4	Input/upload file CTX Specification		
5	Observe the visualization of context graph.		
Input data	CTX Specification		
Results	• The dashboard displays a visual representation of the context graph based on the uploaded specification. • Any errors (e.g., invalid specification) are clearly reported to the user.		
KPIs	Target -> x% | Measured -> y%		
Test Case Result	Pass		

Table 60. Test Case UI-02

Test Case ID	TC-UI- 02	Component	Dashboard
Description	Verify that the dashboard displays accurate and up-to-date metrics for selected sandbox environments, including resource usage and activity indicators.		
Tested by	ONE		
Associated Requirements	SR-SPL-07		
Pre-condition(s)	• Dashboard application is up and running • Users are authenticated and have access to one or more sandbox environments • At least one sandbox environment is available with metrics data		
Test steps			
1	Navigate to Metrics section		
2	Select a Sandbox Environment from the list		
3	Observe metrics from selected sandbox environment		
Input data	Selected sandbox environment		
Results	• The dashboard displays a clear and accurate visualization of metrics for the selected sandbox environment.		
KPIs	TBD		
Test Case Result	Pass		

Table 61. Test Case UI-03

Test Case ID	TC-UI- 03	Component	Dashboard
Description	Verify that the user can log in to the dashboard, including redirection to the IAM Provider, authentication, and return to the authenticated dashboard.		
Tested by	ONE		
Associated Requirements	-		
Pre-condition(s)	• Dashboard is running • IAM Provider is configured and accessible • User is not authenticated		
Test steps			
1	Access the dashboard URL in a browser		
2	Click the "Login" button or try to access a protected route.		
3	Verify redirection to IAM Provider Login Page		
4	Enter valid user credentials in IAM Provider Login Page		

5	After authentication, verify redirection back to dashboard
6	Verify that the dashboard displays username of user
Input data	Valid user credentials (Username/Password)
Results	• User returns authenticated to the dashboard, with access to protected features.
KPIs	TBD
Test Case Result	Pass

3.8.5. Next steps and future development updates

- Deploy Helm Chart via Dashboard
- Isolated Metrics Visualisation
- Integration of CTXD Service

3.9 Trust Assessment Framework

3.9.1. Prototype Description

The **Trust Assessment Framework (TAF)** enables the characterization and quantification of trust in complex environments, such as those envisioned in MIRANDA. TAF evaluates trust sources to derive trustworthiness evidence and comprises two main operational models: the Intra-Domain TAF and the Inter-Domain TAF.

- The **Intra-Domain TAF** is responsible for enforcing rules based on monitored trustworthiness evidence within the same domain. It receives signed attestation reports from the Attestation Agent as a trusted source in order to make a trust decision.
- The **Inter-Domain TAF** is responsible for enforcing rules based on monitored trustworthiness evidence across different domains. It reserves abstraction of signed attestation reports from a different domain via the Context Discovery and assesses cross-domain trust.

The core components of the Intra and Inter domain Trust Assessment Framework are the following:

- **Trust Model Manager**: This component is responsible for selecting an appropriate, pre-stored trust model based (e.g., from intra-domain model, inter-domain model) on the received trust assessment request and creating or updating trust models during the runtime.
- **Trust Source Manager**: This component is responsible for the collection and analysis of the trust evidence from the different trust sources, such as the Attestation Agent in the context of the intra-domain and the Context Discovery in the context of the inter-domain.

- **Trust Level Expression Engine**: This component utilizes pre-defined rules to calculate the ATL of the device, considering also the corresponding trust model and the aggregated trust opinions based on the trust sources.
- **Trust Decision Engine**: This component is responsible for making a trust decision on the device weather Actual Trust Level (ATL) is higher than the Required Trust Level (RTL).
- **Trust Assessment Manager**: This component is the central component responsible for the coordination of all the internal tasks and interactions among the TAF's components. More specifically, it is responsible to process the changes of a trust model when indicated by the Trust Model Manager and the Trust Source Manager and to schedule the Trustworthiness Level Expression Engine's ATL computations and Trust Decision Engine function calls.

Further, TAF operates either by an ad-hoc request or by a running trust request service. The operation of TAF is supported by the capabilities of the MIRANDA Trusted Computing Base (TCB). Currently, the attestation attributes constitute one of the trustworthiness sources used by the TAF in order to compute the ATL of an element, which in turn leads to the trust-state characterization of the target node. The high-level architecture is depicted in in Figure below.

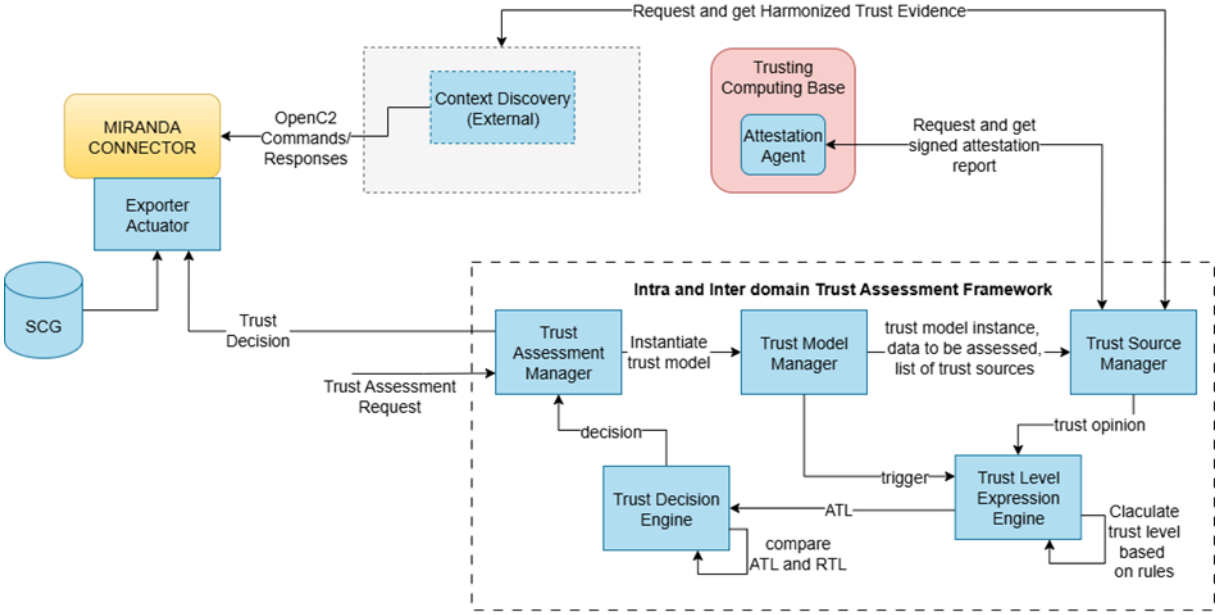

Figure 14. Internal design of the Trust Assessment Framework.

The Trust Assessment Framework contributes to fulfilling the following requirement:

- **SR-SPL-17** – Trusted system context and data export/import: The system must provide authentication and trust assessment of third parties before sharing internal context and data.

3.9.2. Communication interfaces

The Intra- and Inter-domain TAF components communicate with external entities through the distributed event-streaming platform Kafka. To support this communication, several message types have been defined. The following sections list all well-known messages used for this purpose. The messages are organized into two categories: generic message wrappers and message payload (content) definitions.

Table 62. TAF communication interfaces

Topic	Description	Message Sample

GENERIC_REQUEST	This topic triggers the initiation of a trust assessment process providing the following attributes: **sender**: The identifier of the sender of the message.
	serviceType: The service type to be used by the receiver to process the message.
	messageType: The message type to be used by the receiver to process the message.
	responseTopic: The Kafka topic to be used to send a response to this request. These are pre-defined codes.
	requestId: The unique identifier to be repeated in the response for linking request and response.
	message: The actual application message.
		{
		"sender": "0242ac120002-application",
		"serviceType": "TAF",
		"messageType": "TAS_INIT_REQUEST",
		"responseTopic": "0242ac120002",
		"requestId": "4c54a50f8e43",
		"message": {}
		}

GENERIC_RESPONCE	This topic is the response of a trust assessment process with the following attributes:
	sender: The identifier of the sender of the message.
	serviceType: The service type to be used by the receiver to process the message.
	messageType: The message type to be used by the receiver to process the message.
	responseId: The unique identifier copied from the request for linking request and response.
	message: The actual application message.
		{
		"sender": "a77b29bac8f1-taf",
		"serviceType": "TAF",
		"messageType": "TAS_INIT_RESPONSE",
		"responseId": "4c54a50f8e43",
		"message": {}
		}

GENERIC_SUBSCRIPTION_REQUEST	This topic is to subscribe to TAF and receive any update on the trust level providing the following attributes: **sender**: The identifier of the sender of the message. **serviceType**: The service type to be used by the receiver to process the message. **messageType**: The message type to be used by the receiver to process the message. **responseTopic**: The Kafka topic to be used to send a response to this request. These are pre-defined codes.	{ "sender": "0242ac120002-application", "serviceType": "TAF", "messageType": "TAS_SUBSCRIBE_REQUEST", "responseTopic": "0242ac120002", "subscriberTopic": "0242ac120002-updates", "requestId": "4c54a50f8e43", "message": {}

	subscriberTopic: The Kafka topic of this sender to which notification should be published to. **requestId**: The unique identifier to be repeated in the response for linking request and response. **message**: The actual application message.	}

GENERIC_SUBSCRIPTION_RESPONCE	This topic is for TAF to subscribe in the various trust sources (e.g., a failed attestation) with the following attributes:
	sender: The identifier of the sender of the message.
	serviceType: The service type to be used by the receiver to process the message.
	messageType: The message type to be used by the receiver to process the message.
	responseId: The unique identifier copied from the request for linking request and response.
	message: The actual application message.
		{
		"sender": "a77b29bac8f1-taf",
		"serviceType": "TAF",
		"messageType": "TAS_SUBSCRIBE_RESPONSE",
		"responseId": "4c54a50f8e43",
		"message": {}
		}

TAS_INIT_REQUEST	This topic is to initiate a session of the TAF providing the following attributes:
	trustModelTemplate: Indicates the trust model template that should be used as well as its version. The value is string that concatenates a trust model name and a version, joined by an "@" symbol. Depending on the type of trust model template, it is also possible that the application provides a set of additional parameters to the TAF, which are string-based key-value pairs
		{
		"trustModelTemplate": "'INTERDOMAIN0@0.0.1",
		"params": {
		"key": "value"
		}
		}

TAS_INIT_RESPONCE	This topic is the session's response with the following attributes:
	success: In case the session initialization was successful
	sessionId: In case the session initialization was successful and the ID of the newly created session
	error: In case the session initialization was not successful.
	attestationCertificate: Certificate of node based attestation.
		{
		"success": "Session with trust model template 'INTERDOMAIN0.0.1' created.",
		"sessionId": "a6f45092-89c4-4de1-9df5-4d588f5ce8da",
		"attestationCertificate": "VEhJUyBJUyBKVVNUIEEgVEVTVCBDb3JlIEhJUyBKVVNUIEEgVEVTVCBDb3JlPQ=="
		}

TAS_TEARDOWN_REQUEST	This topic is to tear down the request providing the following attribute:	{

	sessionId: The TAF's session Id to be terminated.	"sessionId": "a6f45092-89c4-4de1-9df5-4d588f5ce8da" }

TAS_TEARDOWN_RESPONCE	This topic is the response of the tear down request with the following attributes:
	success: In case the session termination was successful
	error: In case the session termination was not successful.
	attestationCertificate: Certificate of node-based attestation.
		{
		"success": "Session with ID 'a6f45092-89c4-4de1-9df5-4d588f5ce8da' successfully terminated.",
		"attestationCertificate": "VEhJUyBJUyBKVVNUIEEgVEVTVCBDb3JlIEhJUyBKVVNUIEEgVEVTVCBDb3JlPQ=="
		}

TCH_INIT_REQUEST	This topic is for the initiation of requesting the new trustworthiness claim providing the following attribute:
	query: A query that includes the pseudonym identifiers based on which the respective Trustworthiness Claims (TCs) are going to be produced.
		{
		"query": [
		{
		"trusteeIDs": [
		"pseudonym_123"
		]
		}
		]
		}

TCH_INIT_RESPONCE	This topic is for the initiation of receiving the new trustworthiness claim with the following attributes:
	success: In case the pseudonym identifiers have been successfully registered.
	error: In case the pseudonym identifiers have not been successfully registered.
		{
		"success": "Initialization successful"
		}

TCH_TC_REQUEST	This topic is for requesting the new trustworthiness claim providing the following attributes:
	query: A query that includes the claims to be evaluated or reported, as well as the recipient to whom should send the message
		{
		"query": [
		{
		"trusteeID": "pseudonym_456",
		"tchNotifyDestinationTopics": [
		"miranda.node1.taf"
		],
		"requestedClaims": [
		{
		"name": "secure_boot"
		}
		]

		}] }

TCH_TC_RESPONCE	This topic is for receiving the new trustworthiness claim with the following attributes:
	success: In case the ad-hoc request for trustworthiness claims has been produced properly
	error: In case the ad-hoc request for trustworthiness claims has not been produced properly
		{
		"success": "Initialization successful"
		}

3.9.3. Deployment details

To seamlessly evaluate the TAF runtime execution, we containerize it and manage it through the Docker runtime environment. Below the docker-compose.yml configuration file is provided that is used to deploy the TAF as a docker service (lines 1-10) and the necessary Kafka (lines 12-39) and Zookeeper (lines 41-57) services to interact with the TAF service. By running the "docker compose up -d" command, all the services are spawned and we are ready to run the evaluation script and collect our measurements.

```
taf:
    image: ubi/taf:1.0-perf-level-of-isolation
    container_name: mec_taf
    entrypoint: ["sh", "-c", "TAF_CONFIG=/app/res/taf.json /app/out/main"]
    volumes:
      - ./taf/main:/app/out/main:ro
      - ./taf/taf.json:/app/res/taf.json:ro
    depends_on:
      - kafka
    network_mode: host

  kafka:
    image: confluentinc/cp-kafka:7.6.0
    container_name: "kafka"
    hostname: kafka
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_MIRANDA: zookeeper:2181
      KAFKA_LISTENERS:                        DOCKER_INTERNAL://0.0.0.0:9092,
LOCALHOST://0.0.0.0:9091, EXTERNAL://0.0.0.0:9093
      KAFKA_ADVERTISED_LISTENERS:               DOCKER_INTERNAL://kafka:9092,
LOCALHOST://127.0.0.1:9091, EXTERNAL://192.168.2.14:9093
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
DOCKER_INTERNAL:PLAINTEXT,LOCALHOST:PLAINTEXT, EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: DOCKER_INTERNAL
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_LOG_RETENTION_HOURS: 48
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
    ports:
      - 9093:9093
```

```
      - 9091:9091
    volumes:
      - kafka:/var/lib/kafka/data
    depends_on:
      - zookeeper
    logging:
      options:
        max-size: "10m"
        max-file: "10"
    restart: unless-stopped

  zookeeper:
    image: confluentinc/cp-zookeeper:7.6.0
    container_name: "zookeeper"
    hostname: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - 2181:2181
    volumes:
      - zookeeper-data:/var/lib/zookeeper/data
      - zookeeper-logs:/var/lib/zookeeper/log
    logging:
      options:
        max-size: "10m"
        max-file: "10"
    restart: unless-stopped

volumes:
  kafka:
    name: demo_kafka
  zookeeper-data:
    name: demo_zookeeper-data
  zookeeper-logs:
    name: demo_zookeeper-logs
```

3.9.4. Individual component testing

Table 63. Component Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-TAF-01	Connection of TAF with at least one trust source (i.e., MIRANDA Attestation Agent) to validate the correctness of the TAF behaviour based on the expected (reference) trust evolution of the target service graph.	Achieved
TC-TAF-02	Scalability testing - Validate the correctness of TAF process for varying size and complexity of trust models. This primarily includes number of nodes and links comprising a service graph chain or a topology for which a trust assessment needs to be performed.	Achieved

TC-TAF-03	Instantiation of TAF as a trusted application (e.g., SGX enclave) for benchmarking interplay between security and performance.	Achieved

Table 64. Test Case TAF-01

Test Case ID	TC-TAF-01	Component	TAF
Description	Connection of TAF with at least one trust source (i.e., MIRANDA Attestation Agent) to validate the correctness of the TAF behaviour based on the expected (reference) trust evolution of the target service graph.		
Tested by	UBITECH		
Associated Requirements	SR-SPL-17		
Pre-condition(s)	The device should be TCB-enabled and TAF successfully deployed and running.		
Test steps			
1	Initiate TAF and connect TSM with an external attestation source/agent.		
2	Different number and type of evidence to be ingested by the Attestation Agent as part of the integrity check of the host edge device prior to triggering the TLEE and the TDE of the TAF. Examples include secure boot CFA etc.		
Input data	N/A		
Results	Decrease or increase in the trust level based on the binary output of trust source		
KPIs	Target -> at least 1 trust sources with >= 3 evidence | Measured 1 trust source with 3 evidences (i.e., CIV, secure boot, CFI)		
Test Case Result	Achieved		

Table 65. Test Case TAF-02

Test Case ID	TC-TAF-02	Component	TAF
Description	Scalability testing - Validate the correctness of TAF process for varying size and complexity of trust models. This primarily includes number of nodes and links comprising a service graph chain or a topology for which a trust assessment needs to be performed.		
Tested by	UBITECH		
Associated Requirements	SR-SPL-17		
Pre-condition(s)	The device should be TCB-enabled and TAF successfully deployed and running.		
Test steps			

1	Initiate TAF (TMM) with the trust model capturing varying level of topology complexities. This essentially dictates the number of scalar trust values that need to be fused for being checked against the defined trust policies/rules.
2	Successful running of a trust assessment service until a session teardown.
Input data	Trust model
Results	N/A
KPIs	Target -> < 300ms considering the convergence in the community for reaching a trust decision in safety critical systems | Measured -> 280ms
Test Case Result	Achieved

Table 66. Test Case TAF-03

Test Case ID	TC-TAF-03	Component	TAF
Description	Instantiation of TAF as a trusted application (e.g., SGX enclave) for benchmarking interplay between security and performance.		
Tested by	UBITECH		
Associated Requirements	SR-SPL-17		
Pre-condition(s)	The device should be TCB-enabled and TAF successfully deployed and running.		
Test steps			
1	Initiate TAF as a trusted application.		
2	Successful running of a trust assessment process.		
Input data	N/A		
Results	N/A		
KPIs	Target -> < 25%overhead added by the enclavisation | Measured -> 20% overhead		
Test Case Result	Achieved		

3.9.5. Next steps and future development updates

The foreseen future development for the Trust Assessment Framework is:

- To converge the actual benchmarking of TAF as a standalone component and to compare and evaluate it against the MIRANDA Use Cases (e.g., the DAEM UC)
- From a research perspective:
  - to increase the trust sources (e.g., to include a misbehaviour detection component as a trust source) and check the accuracy;
  - to consider a potential federation of TAF for a better perception of trust level throughout the network;

- and to replace rule-based logic with subjective logic (e.g., to elevate TAF from binary-based decision to subjective logic-based decision where we capture the uncertainty dimension).

3.10 Trusted Computer Base

3.10.1. Prototype Description

The Trusted Computing Base (TCB) constitutes the underlying Root of Trust (RoT) of the MIRANDA framework. In a nutshell, such a component is critical to the entire lifecycle management of an entity, supporting secure boot, cryptographic services, access control, secure storage, and attestation. In the context of MIRANDA, each edge device is equipped with a TCB to support runtime attestation for both deployed services and the underlying infrastructure components on which these services operate. The generated attestation reports are utilized by the Trust Assessment Framework. The attestation mechanisms are supported by (a) the Attestation Agent for enabling real-time verification through implicit attestation guided by key-restriction policies and by (b) the eBPF-based Tracer for host-based monitoring based eBPF and kernel-level extensions.

Currently the Attestation Agenet supports Configuration Integrity Verification (CIV) functionality based on varying types of Root of Trust (e.g., SGX Gramin and TrustZone), while the eBPF-based Tracer captures system calls and the execution flow of specific processes and binaries. In a nutshell, CIV ensures the correct state of platform configuration or binaries, protecting against compromise by malicious entities. It involves receiving policies for correct configuration, setting up cryptographic material on newly joined nodes, and proving correctness without disclosing sensitive information. The use of restrained keys ensures that their functionality remains locked until the node proves its correctness by supplying accurate measurements. Once validated, the node can use the key to sign nonces, providing verifiable statements about its state. Verifiers can confirm the authorized state of the Prover by obtaining valid signatures utilizing the agreed-upon key, even in the absence of specific information regarding its exact state. At a high level CIV has two phases[3] a) the Join phase that creates the necessary keys and certificates, set the key restriction usage policies and b) the runtime phase where the verifier challenges (e.g., queries runtime attestation quote) the prover to provide the attestation. The high-level architecture is depicted in Figure 15.

The Trusted Computing Base contributes to fulfilling the following requirement:

- SR-SPL-17 – Trusted system context and data export/import: The system must provide authentication and trust assessment of third parties before sharing internal context and data.

[3] N. Fotos, S. Vasileiadis and T. Giannetsos, "Trust or Bust: Reinforcing Trust-Aware Path Establishment with Implicit Attestation Capabilities," *2025 IEEE International Conference on Cyber Security and Resilience (CSR)*, Chania, Crete, Greece, 2025, pp. 867-874, doi: 10.1109/CSR64739.2025.11130147.

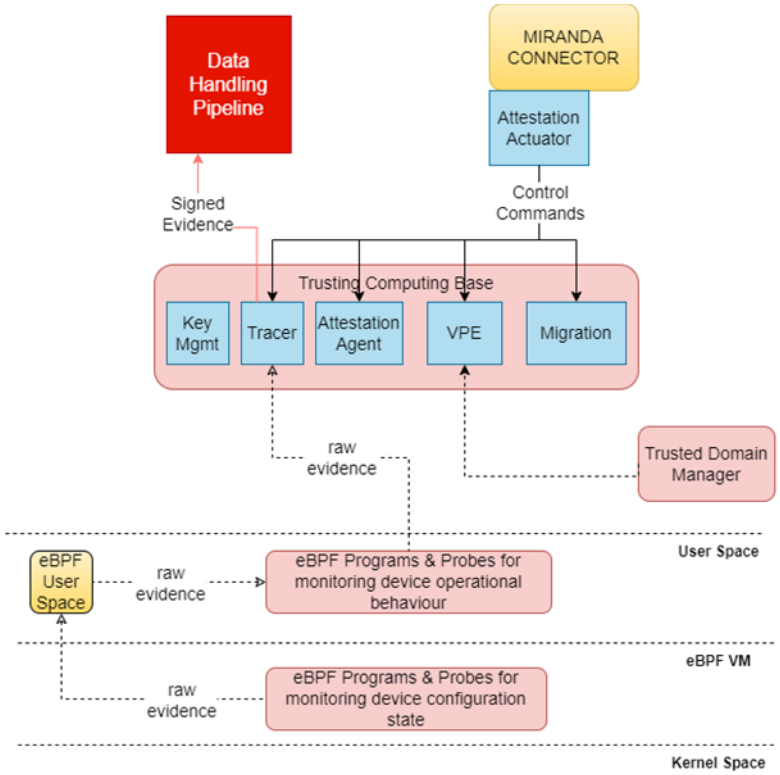

Figure 15. Internal design of the Trusted Computing Base

3.10.2. Communication interfaces

Table 67. TCB Communication interfaces

#	Method	Description	Request Body (input)	Response Body (output)
1	TCP Socket	This API is used for attestation key creation as part of the join phase.	N/A	A unified buffer with fields: 1. Key Name: A 32-byte array representing the attestation key's name. 2. N: A 256-byte array representing the public root id key of the device 3. E: 3 bytes representing the exponent of the private root key of the device
2	TCP Socket	This API is used for verifying the credentials received by the Attestation Agent	A unified buffer with fields: 1. Key Name: A 32-byte array representing the attestation key's name. 2. CIV credential: A 144 byte array representing the authentication ticket created	A unified buffer with fields: 1. Verification Status: A 32-byte array representing the correct or wrong verification

			by the DM and the VPE public key 3. Encrypted Rand CIV: A 256-byte array representing the random encryption key for decrypting the CIV credentials 4. Authentication Digest for CIV: A 32-byte array representing the Hash of the encrypted credential, that the attestation agent will compare against to 5. IV: A 16-byte array representing the initialization vector that was constructed during the symmetric encryption of the credentials	
3	TCP Socket	This API is used for initiating a challenge - response of a random nonce from the Attestation Agent.	N/A	A unified buffer with fields: 1. Nonce: A 32-byte array representing a random generated nonce
4	TCP Socket	This API is used for executing the runtime attestation protocol.	A unified buffer with fields: 1. VPE Signature R: A 32-byte array representing the r parameter of the VPE's signature 2. VPE Signature s: A 32-byte array representing the s parameter of the VPE's signature 3. VPE public key: A 65-byte array representing the Public Key of the VPE 4. Message: A 32-byte array representing the received message to be signed 5. Tracer Signature r: A 32-byte array representing the r parameter of the Tracer's signature 6. Tracer Signature s: A 32-byte array representing the s	A unified buffer with fields: 1. CIV Signature r: A 32 byte array representing the r parameter of the CIV's signature 2. CIV Signature s: A 32 byte array representing the s parameter of the CIV's signature 3. Attestation Public Key: A 65- byte array representing the Attestation Public key

			parameter of the Tracer's signature 7. Ticket r: A 32-byte array representing the r parameter of the authentication ticket 8. Ticket s: A 32-byte array representing the s parameter of the authentication ticket	
5	TCP Socket	This API is used for initiating a challenge - response of a random nonce from the VPE.	N/A	A unified buffer with fields: 1. Nonce: A 32-byte array representing a random generated nonce
6	TCP Socket	This API is used for verifying the VPE credentials	A unified buffer with fields: 1. Key Name: A 32-byte array representing the attestation key's name. 2. N: A 256-byte array representing the public root id key of the device 3. E: 3 bytes representing the exponent of the private root key of the device 4. VPE credential: A 160- byte array representing the wrapped VPE key and the reference values of the high end TCB 5. Encrypted Rand VPE: A 256- byte Array that contains the Encrypted key for encrypting/decrypting the VPE credentials 6. Authentication Digest for VPE: A 32-byte array representing the Hash of the encrypted credential, that the VPE will compare against to 7. IV: A 16-byte array representing the initialization vector that was constructed during the symmetric encryption of the credentials	A unified buffer with fields: 1. Verification Status: A 32-byte array representing the correct or wrong verification

7	TCP Socket	This API is used for initiating the runtime attestation mechanism of the VPE.	A unified buffer with fields: 1. Tracer Signature r: A 32-byte array representing r parameter of the Tracer's signature 2. Tracer Signature s: A 32-byte array representing the s parameter of the Tracer's signature 3. CIV nonce: A 32-byte array representing the challenge sent by Attestation Agent	A unified buffer with fields: 1. VPE Signature r: A 32 byte array representing the r parameter of the VPE's signature 2. VPE Signature s: A 32 byte array representing the s parameter of the VPE's signature
8	TCP Socket	This API is used for creating the credentials for VPE and CIV	A unified buffer with fields: 1. Key Name: A 32-byte array representing the attestation key's name. 2. N: A 256-byte array representing the public root id key of the device 3. E: 3 bytes representing the exponent of the private root key of the device	A unified buffer with fields: 1. VPE credential: A 160- byte array representing the encrypted VPE Key and the reference values of the TCB 2. Encryption VPE credential Key: A 256-byte Array that contains the encrypted key for encrypting/decrypting the VPE credentials 3. Authentication digest VPE:A 32-byte array representing the Hash of the encrypted credential, that the VPE will compare against to 4. IV VPE: A 16-byte array representing the initialization vector that was constructed during the symmetric encryption of the credentials 5. CIV credential: A 144- byte array representing the encrypted VPE public key and the encrypted authentication ticket 6. Encrypted Rand CIV: A 256-byte array that contains the encrypted key for encrypting/decrypting the CIV credentials 7. Authentication Digest for CIV: A 32- byte array representing the Hash of the encrypted credential, that the Attestation Agent will compare against to 8. IV: A 16-byte array representing the initialization vector that was constructed during the symmetric encryption of the credentials

3.10.3. Deployment details

TCB's security and attestation capabilities are built on UBITECH's library called UBITrust v1.0[4]. The library has the following capabilities:

- Attestation Key Management:
  - Creation of self-generated Attestation Keys bound to verified key usage restriction policies for governing the construction of (on-premise) attestation assertions;
  - Support for migratable keys with similar policy restrictions, enhancing flexibility while maintaining high degree of security.
- Credential Management:
  - Support for MakeCredential and ActivateCredential operations, ensuring compatibility with TPM specifications.
- Cryptographic Functionality:
  - Hashing algorithms for secure data integrity.
  - HMACs (Hash-Based Message Authentication Codes) for message authentication.
  - Symmetric encryption using AES-128 for both 16-byte and 32-byte keys.
  - Asymmetric encryption with RSA for robust key exchange and data protection.
  - A secure Key Derivation Function (KDF) for generating cryptographic keys.
  - A trusted Random Number Generator (RNG) for high-entropy randomness.
- Digital Signatures:
  - Implementation of ECDSA for secure, standards-compliant signing operations.

3.10.4. Individual component testing

Table 68. TCB Component Test Cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-TCB-01	This unit test is dedicated for testing the JOIN phase for Configuration Integrity Verification Protocol. CIV scheme leverages TCB to ensure that the configuration of an embedded system can be verified by authorized stakeholders.	Achieved
TC-TCB-02	This unit test is dedicated for testing the Runtime phase for Configuration Integrity Verification Protocol	Achieved

[4] https://github.com/ubitech/ubitrust

Table 69. Test Case TCB-01

Test Case ID	TC- CB-01	Component	Attestation Agent
Description	This unit test is dedicated for testing the JOIN phase for Configuration Integrity Verification Protocol. CIV scheme leverages TCB to ensure that the configuration of an embedded system can be verified by authorized stakeholders.		
Tested by	UBITECH		
Associated Requirements	SR-SPL-17		
Pre-condition(s)	The device should be TCB-enabled.		
Test steps			
1	Execute JOIN phase of CIV		
2	Construction of all necessary attestation-related crypto primitives and key restriction usage policies		
Input data	N/A		
Results	N/A		
KPIs	Target -> < 800ms| Measured -> 650ms		
Test Case Result	Achieved		

Table 70. Test Case TCB-02

Test Case ID	TC-TCB-02	Component	Attestation Agent
Description	This unit test is dedicated for testing the Runtime phase for Configuration Integrity Verification Protocol		
Tested by	UBITECH		
Associated Requirements	SR-SPL-17		
Pre-condition(s)	The device should be TCB-enabled and JOIN Phase performed.		
Test steps			
1	Perform CIV		
2	Successful Verification		
Input data	A random generated nonce		

Results	Sign over the nonce provided as input
KPIs	Target -> < 1sec | Measured -> 960ms
Test Case Result	Achieved

3.10.5. Next steps and future development updates

The foreseen future development for the Trusted Computing Base is:

- Extend the set of attributes that we can measure with the eBPF-based Tracer beyond system calls and to be able to conduct a more dynamic verification of various behavioural aspects of the target device beyond Configuration Integrity Verification (e.g., Control-flow Attestation).
- Finalise the VPE implementation and validation of its security. The verifiable update and enforcement of the key restriction usage policy without the need to change the attestation key.

3.11 Automatic Detection

3.11.1. DetectMate (AIT)

The automatic detection component contains the anomaly detection tool DetectMate (previously AMiner). The DetectMate implements a set of configurable detectors, that can detect complex anomalies.

3.11.1.1. Prototype Description

The DetectMate consists of a set of detectors and a connector component that reads and writes to Kafka MQ.

The process of the DetectMate is a s follows:

1. Receive configuration from user via API interface
2. Receive parsed logs from Kafka MQ
3. Start training with parsed logs until training limit is reached (defined in config)
4. Start detection phase
5. Write anomalous logs to Kafka MQ

The following requirements are relevant to the DetectMate:

- NON-FR-B-1.35
- NON-FR-B-1.34
- SR-SPL-04
- FR-A-1.5
- FR-A-1.4

3.11.1.2. Communication interfaces

Table 71. Detect Mate Communication interfaces (HTTP)

#	Method	REST Endpoint	Description	Request Body	Response Body

1	GET	/admin/status	Returns status (running state, and current configurations)	None	"{
					""status"": {
					""component_type"": ""string"",
					""component_id"": ""string"",
					""running"": true|false
					},
					""settings"": {
					""key"": ""value""
					},
					""configs"": {
					""key"": ""value""
					}
					}"

2	POST	/admin/start	Starts the data processing engine thread.	None	{"message": "..."}
3	POST	/admin/stop	Stops the data processing engine thread.	None	{"message": "..."}
4	POST	/admin/reconfigure	Dynamically updates service parameters.	{"config": dict, "persist": bool}	{"message": "..."}
5	POST	/admin/shutdown	Gracefully terminates the entire service process.	None	{"message": "..."}

Table 72. Detect Mate Communication interfaces (Kafka)

Topic	Description	Message Sample
input	The input (in JSON format) the log anomaly detector receives from Kafka MQ.	{ "__version__": "string", "parserType": "string", "parserID": "string", "EventID": 0,

		"template": "string",
		"variables": ["string"],
		"parsedLogID": 0,
		"logID": 0,
		"log": "string",
		"logFormatVariables": {
		"key": "value"
		},
		"receivedTimestamp": 0,
		"parsedTimestamp": 0
		}

output	The output (in JSON format) the log anomaly detector sends to Kafka MQ.	{
		"__version__": "string",
		"detectorID": "string",
		"detectorType": "string",
		"alertID": 0,
		"detectionTimestamp": 0,
		"logIDs": [0],
		"score": 0.0,
		"extractedTimestamps": [0],
		"description": "string",
		"receivedTimestamp": 0,
		"alertsObtain": {
		"key": "value"
		}
		}

3.11.1.3. Deployment details

First, clone DetectMateService and navigate into the repository:

```
git clone https://github.com/ait-detectmate/DetectMateLibrary.git
cd DetectMateService
git checkout demo_m
```

We recommend using uv to manage the environment and dependencies.

1. Create and activate a virtual environment with uv:

```
uv venv
source .venv/bin/activate
```

2. Install the project:

```
uv pip install .
```

If you prefer plain pip, you can set things up like this instead:

```
# Create a virtual environment
python -m venv .venv
```

```
# Activate it
source .venv/bin/activate
# Install the project in editable mode with dev dependencies
pip install .
```

To run the service with custom variables, we can define settings. For example, create a file named settings.yaml:

```
component_name: my-first-service
component_type:   core      #   or   use   a   library   component   like
"detectors.RandomDetector"
log_level: INFO
log_dir: ./logs
manager_addr: ipc:///tmp/detectmate.cmd.ipc
engine_addr: ipc:///tmp/detectmate.engine.ipc
```

The config.yaml file that defines the parser for JSON logs looks like this:

```
parsers:

    JsonMatcherParser:
        method_type: matcher_parser
        auto_config: False
        log_format: "<Content>"
        time_format: null
        params:
            remove_spaces: True
            remove_punctuation: True
            lowercase: True
            path_templates: data/miranda_templates.txt

    JsonParser:
        method_type: json_parser
        time_format: null
        auto_config: False
        params:
            timestamp_name: "time"
            content_name: "message"
            content_parser: JsonMatcherParser
```

The JSON parser requires predefined template for the logs which must be stored in data/miranda_templates.txt.

The Dockerfile is given as:

```
FROM python:3.12-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
    git && \
    rm -rf /var/lib/apt/lists/*

COPY --from=ghcr.io/astral-sh/uv:latest /uv /usr/local/bin/uv
#RUN pip install uv

COPY . /app
COPY pyproject.toml .

RUN uv pip install --system -e .
```

```
CMD ["uv", "run", "demo/detectmate_detector.py"]
```

The docker-compose file is given as:

```
services:
  app2:
    build:
      dockerfile: demo/Dockerfile.app2
      context: ..
    environment:
      KAFKA_SERVER: kafka:9092
    depends_on:
      - kafka
  kafka:
    hostname: kafka
    container_name: kafka
    image: apache/kafka-native
    ports:
      - "9092:9092"
    environment:
      # Configure listeners for both docker and host communication
      KAFKA_LISTENERS: 'PLAINTEXT://kafka:9092,CONTROLLER://kafka:9093'
      KAFKA_ADVERTISED_LISTENERS: 'PLAINTEXT://kafka:9092'
                                          KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
'CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT'

      # Settings required for KRaft mode
      KAFKA_NODE_ID: 1
      KAFKA_PROCESS_ROLES: broker,controller
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      KAFKA_CONTROLLER_QUORUM_VOTERS: '1@kafka:9093'

      # Listener to use for broker-to-broker communication
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT

      # Required for a single node cluster
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

To run everything:

```
docker-compose up
```

3.11.1.4. Individual component testing

Table 73. Detect Mate Component Test cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-DM-01	Detection successful	Achieved
TC-DM-02	Deployment and reading and sending to Kafka successful	To be tested

Table 74. Test case DM-01

Test Case ID	TC-DM-01	Component	DetectMate
Description	To test the functionality of the DetectMate we let it detect anomalies on logs.		
Tested by	AIT		
Associated Requirements	FR-A-1.5, FR-A-1.4		
Pre-condition(s)	input data and configuration given		
Test steps			
1	The input logs are ingested by the core function of the parser.		
2	The parsed logs are ingested by the core function of the detector.		
3	The detector trains on 70% of the logs.		
4	The detector detects anomalies on the remaining 30% of the logs.		
5	The anomalies are verified by a human validator.		
Input data	A log file (Mindicity logs), type: JSON		
Results	The DetectMate was successful, ran without errors and detected a significant proportion of the logs with high precision.		
KPIs	Precision of result? Target -> 80% | Measured -> 100%		
Test Case Result	Pass		

Table 75. Test case DM-02

Test Case ID	TC-DM-02	Component	DetectMate
Description	The log parser is deployed with Docker and receives logs from Kafka and sends the parsed logs to Kafka MQ.		
Tested by	AIT		
Associated Requirements	FR-A-1.5, FR-A-1.4, SR-SPL-04		
Pre-condition(s)	input data given		
Test steps			
1	Deployment of parser and detector via Docker		
2	Start of the parser and detector with correct settings and configs.		
3	Simulated Kafka MQ sends data to the parser.		
4	Parser parses data and sends to Kafka MQ		
5	Detector receives data from Kafka MQ.		

6	Detector trains on part of the data until limit reached (defined in config)
7	Detector detects anomalies and sends them to Kafka MQ.
8	Simulated receiver receives the anomalies.
9	Anomalies are validated by human validator.
Input data	Type and format of data
Results	The DetectMate was deployed successfully, ran without errors and detected a significant proportion of the logs with high precision.
KPIs	Precision of result? Target -> 80% | Measured -> 100%
Test Case Result	Pass / Failed

3.11.1.5. Next steps and future development updates

More detectors will be implemented for the DetectMate to cover a wider range of anomaly types with the detectors. An automatic configuration process will be implemented for the detector since the configuration is dependent on the logs and highly complex.

3.11.2. Sequence-Based Anomaly Detection

3.11.2.1. Prototype Description

The **Ensemble Sequence-Based Anomaly Detector** is a real-time streaming module designed to identify deviations in system log patterns. By combining semantic embeddings with a consensus mechanism across three distinct algorithms, the system robustly detects anomalies while minimizing noise.

System Architecture

The system follows a linear, stream-processing pipeline architecture designed for high-throughput log analysis. The architecture is composed of five distinct interconnected modules.

- **Data Ingestion Layer:** A Kafka consumer interface that subscribes to the raw-logs topic to ingest system events in real-time.
- **Transformation Engine:** A composite module consisting of a **Pre-processor** for text sanitization and a **RoBERTa-based Embedding Model**[5] (768-dim output) for vectorization.
- **State Aggregator:** A Sliding Window buffer that holds and averages embeddings (default size: 50 logs) to represent the current state.
- **Ensemble Detection Module:** A parallel processing block containing three independent detectors:
    - **ML Component:** Isolation Forest (unsupervised learning).

[5] https://huggingface.co/Dumi2025/log-anomaly-detection-model-roberta

- **Statistical Components:** Z-score and Interquartile Range (IQR) calculators.

- **Decision & Output Layer:** A logic gate that implements a "2-of-3" voting mechanism and an alerting interface that publishes results to the alerts Kafka topic.

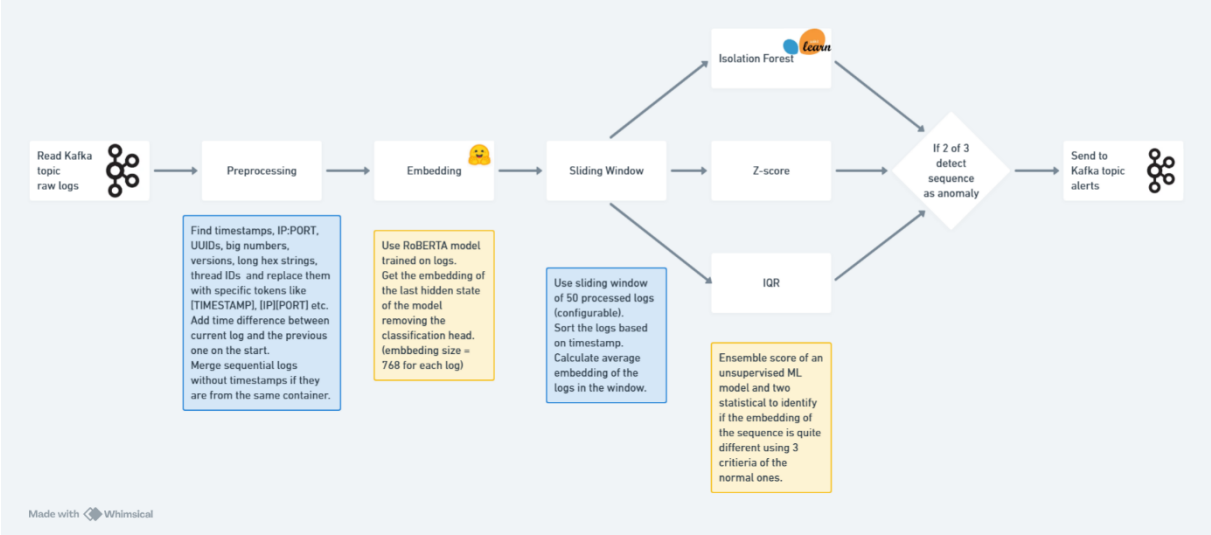

Figure 16. Sequence-Based Anomaly Detection Architecture

Core Functionalities

The system provides specific functional capabilities to ensure robust detection and reduce noise.

- **Log Normalization & Merging:** The system automatically cleanses logs by tokenizing volatile data (IPs, UUIDs, timestamps) and intelligently merges multi-line logs (e.g., stack traces) from the same container into single events.
- **Semantic Representation:** Unlike simple keyword matching, the system extracts the **semantic meaning** of log sequences using the hidden states of a fine-tuned RoBERTa model, capturing subtle context changes.
- **Temporal Context Awareness:** By calculating the average embedding over a sliding window of 50 logs, the functionality ensures that detection is based on the *sequence of events* rather than isolated log lines.
- **Consensus-Based Verification:** The system minimizes false positives through a majority voting function. An anomaly is only flagged if it is confirmed by at least two distinct mathematical approaches (one ML-based, two statistical).
- **Real-Time Delta Tracking:** It calculates and embeds the time difference between consecutive logs, allowing the model to detect anomalies related to event timing and frequency.

The Requirements Mapping:

- FR-A-1.4
- FR-A-1.5
- FR-A-1.13
- FR-A-1.18
- FR-B-1.41
- NFR-B-1.45

3.11.2.2. Communication interfaces

The main communication interface with the other components is using Kafka.

Table 76. Sequence-Based Anomaly Detection communication interfaces

Topic	Description	Message Sample

component.raw-syslogs	System Logs (e.g. Mindicity logs)	{
		"id": "text",
		“log”: Raw log “string”,
		“receivedTimestamp”: “timestamp”
		}

component.alerts	Alerts created from the component	{
		"timestamp": "created timestamp”,
		"window_start": "timestamp",
		"window_end": "timestamp",
		"is_anomaly": bool,
		"ensemble_votes": int,
		"vote_details": json details,
		"isolation_forest_score": float,
		"euclidean_distance": float,
		"num_logs": int,
		"messages": List of raw logs inside the window,
		“processed_messages”: List of processed logs inside the window,
		“log_imestamps”: List of extracted timestamps,
		“ips”: “list of ip and ports”
		}

…	…	…

3.11.2.3. Deployment details

Docker compose file

```
services:
  inference:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: miranda-inference
    volumes:
      - ./src:/app/src
      - ./models:/app/models
      - ./.env:/app/.env
    environment:
      - DEVICE=cpu
      - KAFKA_BOOTSTRAP_SERVERS=${KAFKA_BOOTSTRAP_SERVERS}
```

```
      - KAFKA_INPUT_TOPIC=${KAFKA_INPUT_TOPIC:-miranda_syslog_raw}
      - KAFKA_OUTPUT_TOPIC=${KAFKA_OUTPUT_TOPIC:-miranda_syslog_alerts}
      - KAFKA_CONSUMER_GROUP=${KAFKA_CONSUMER_GROUP:-kafka-mm2}
    command: ["python", "src/inference.py"]
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 4G
        reservations:
          memory: 2G
    profiles:
      - inference

  inference-gpu:
    build:
      context: .
      dockerfile: Dockerfile.gpu
    container_name: miranda-inference-gpu
    volumes:
      - ./src:/app/src
      - ./models:/app/models
      - ./.env:/app/.env
    environment:
      - DEVICE=cuda
      - KAFKA_BOOTSTRAP_SERVERS=${KAFKA_BOOTSTRAP_SERVERS}
      - KAFKA_INPUT_TOPIC=${KAFKA_INPUT_TOPIC:-miranda_syslog_raw}
      - KAFKA_OUTPUT_TOPIC=${KAFKA_OUTPUT_TOPIC:-miranda_syslog_alerts}
      - KAFKA_CONSUMER_GROUP=${KAFKA_CONSUMER_GROUP:-kafka-mm2}
    command: ["python", "src/inference.py"]
    restart: unless-stopped
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: 1
              capabilities: [gpu]
    profiles:
      - inference-gpu

networks:
  default:
    name: miranda-network
```

Dockerfile

```
FROM python:3.12-slim

# Set working directory
WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    git \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements first for better caching
COPY requirements.txt .
```

```
# Install Python dependencies
RUN pip install --upgrade pip setuptools wheel
RUN pip install -r requirements.txt

# Copy source code
COPY src/ ./src/
COPY .env ./.env

# Create directories for models and results
RUN mkdir -p models
COPY models/ ./models/

# Set environment variables
ENV PYTHONUNBUFFERED=1
ENV PYTHONDONTWRITEBYTECODE=1

# Default command
CMD ["python", "src/inference.py"]
```

3.11.2.4. Individual component testing

Table 77. Sequence-Based Anomaly Detection Component test cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-SAD-01	End-to-End Anomaly Detection Pipeline	Achieved
TC-SAD-02	Ensemble Voting and False Positive Reduction	To be tested
TC-SAD-03	Performance and Real-time Processing	To be tested
TC-SAD-04	Detection Accuracy on Attack Patterns	To be tested

Table 78. Test Case SAD-01

Test Case ID	**TC-SAD-01**	**Component**	**Sequence-Based Anomaly Detector**
Description	Validates the complete anomaly detection pipeline from Kafka log consumption through semantic embedding, sliding window aggregation, ensemble detection, and alert publication to Kafka output topic.		
Tested by	SPH		
Associated Requirements	FR-A-1.4, FR-A-1.5, FR-A-1.18, FR-B-1.41		
Pre-condition(s)	Kafka broker running, trained models loaded (Isolation Forest, centroid, thresholds), input topic 'raw-logs' and output topic 'anomaly-alerts' configured, test log dataset available		
Test steps			
1	Start KafkaAnomalyPipeline with test configuration		
2	Publish 50 normal log messages to 'raw-logs' topic		

3	Verify logs consumed and sliding window fills to 50 logs
4	Verify embeddings generated for all logs
5	Verify inference triggered and window classified as normal
6	Publish 50 anomalous logs (e.g. SQL injection attack pattern)
7	Verify anomaly detected by ensemble (2+ detectors agree)
8	Verify alert published to 'anomaly-alerts' topic with complete metadata (scores, votes, messages, timestamps, IPs)
Input data	Mindicity Logs (JSON-LD format)
Results	Pipeline successfully consumed 100 logs from Kafka. Normal sequence: No alert generated (correct classification). Anomalous sequence: Alert generated with votes=3/3 (all detectors agreed). Alert published with Isolation Forest score: -0.42, Z-score: flagged, IQR: flagged.
KPIs	End-to-end latency: Target -> <1 minute
Test Case Result	Pass

3.11.2.5. Next steps and future development updates

- Add concrete summary of logs inside the window to be used easily on the IoC extraction component and UI
- Add human readable explanations for the reason of the alert creation.
- Performance Testing based on accuracy, precision, recall and all test cases.
- Deployment on Kubernetes.

3.12 DeepGuardian

Figure 12 presents DeepGuardian architecture, whose is composed of five main components: (i) Aggregator, (ii) Collector, (iii) Agent, (iv) Dashboard, and (v) Policy Enforcer.

The Aggregator corresponds to the main framework unit and is responsible for coordinating the federated training procedure. It manages and distributes the different ML models used for anomaly detection by the federated agents, storing the various training iterations of these models in a database for analysis, comparison and auditing purposes. Additionally, the Aggregator is responsible for the enrolment of the different services that will be accompanied by the Agent and Collector components.

The Agent component, where the federated agent resides, includes the detection engine, which is responsible for inferring inbound and outbound traffic, based on the network traffic collected by the Collector. It is also the main trainee in the federated training procedure, together with other Agent components deployed in applications of the same type that use the same ML model. Both the Agent and the Collector components may be injected as sidecars alongside any existing container hosting the application to be secured.

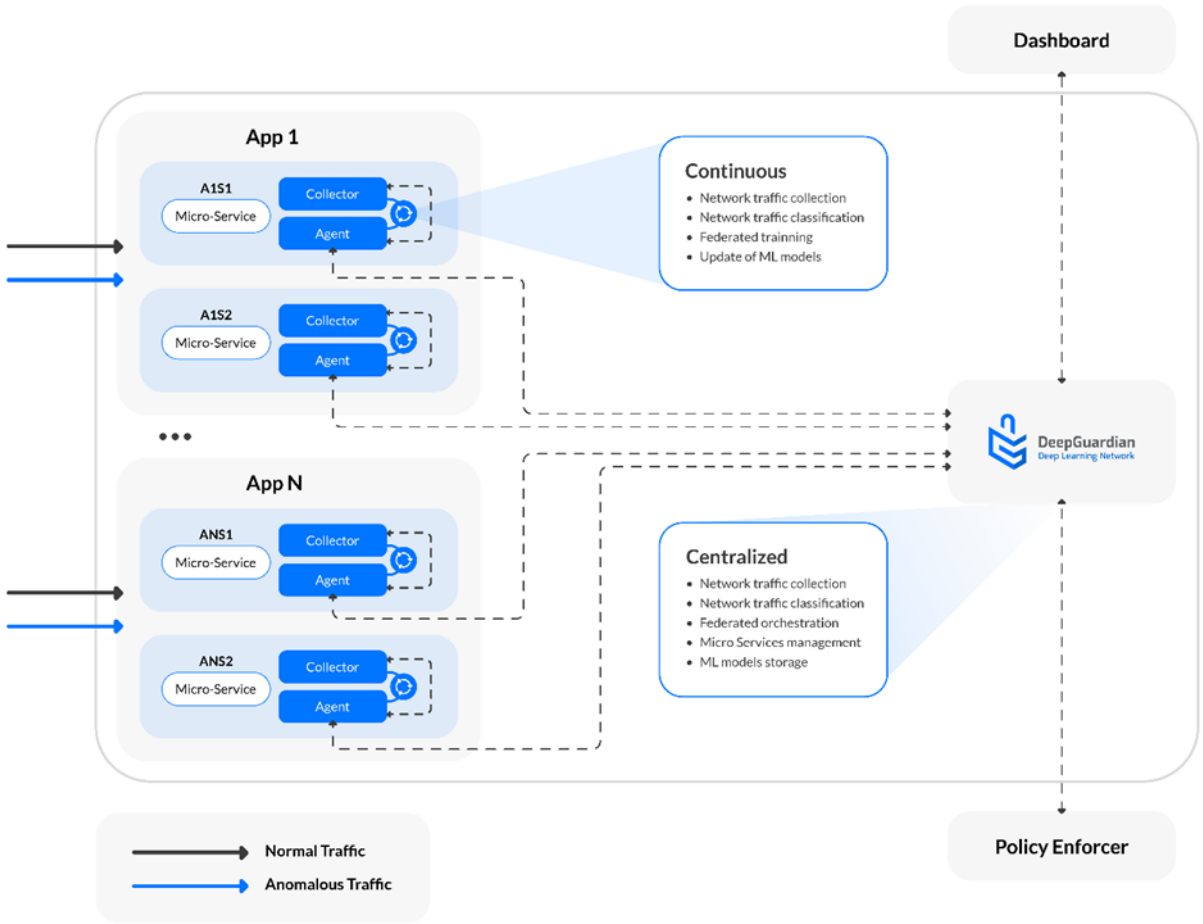

Figure 17. DeepGuardian Architecture

DeepGuardian architecture

The Dashboard enables the user to gather first hand insights over all the applications being protected, by providing per application and overall statistics (i.e., network flows detected and several flows/classification metrics: number of normal and attack flows, percentage of flows per Source IP, Inbound/Outbound Packets per minute, Packets and Bytes transferred per time window, etc) and by enabling the human-in-the-loop functionalities, providing the user the ability to override the system classification, which is later considered upon the next federated training round.

The Policy Enforcer mechanism is an optional module of the framework and enables the enforcement of network policies aiming to mitigate potential threats detected by the detection engine.

DeepGuardian is designed to handle multiple applications across multiple environments, allowing for a centralised Aggregator that manages all DeepGuardian components.

DeepGuardian detection engine is prepared to work in two different modes: anomaly detection or attack classification. For the first, an autoencoder approach is used, leveraging the reconstruction error produced by this ML method together with a dynamic threshold to distinguish between normal and malicious traffic. For the second, a XGBoost module is used to provide multi-class classification over the network traffic.

Classification as a Service (CaaS)

Considering the ever-increased need for functionalities as services, a dedicated module has been devolved to attend this requirement: the CaaS. The major difference between this service to the core DeepGuardian functionalities is the ability to receive network traffic via an asynchronous communication channel, provide the respective inference and statistics related to the classification process. Figure presents the CaaS workflow.

The above workflow has been validated with two data batches from LogsTails. The first included data collected under normal operating conditions, whereas the second encompassed data collected while the system was under attack. It was possible to observe the classification reports being published in the corresponding topic, as well as the linked statistics.

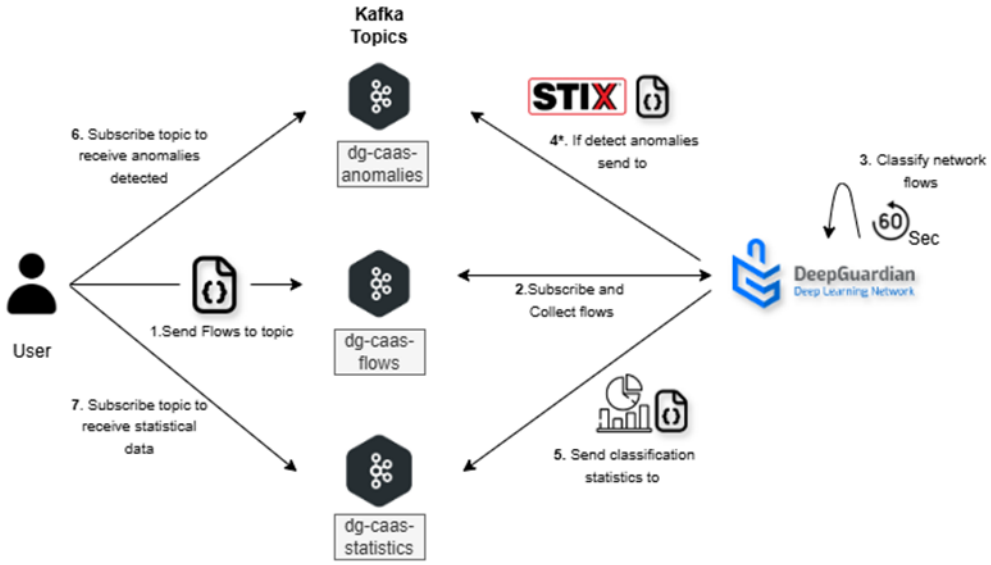

Figure 18. DeepGuardian - CaaS workflow

Mapping with MIRANDA requirements

DeepGuardian maps to the following MIRANDA requirements:

- FR-A-1.4 – Continuous detection
  - DeepGuardian continuously monitors inbound and outbound network traffic, enabling real-time detection of anomalies and attacks through ML-based analysis.
- FR-A-1.5 – Detection capabilities
  - The framework supports anomaly-based and multi-class classification capabilities, leveraging unsupervised and supervised ML approaches.
- FR-A-1.18 – Semi-automated response with policies
  - The Policy Enforcer module enables enforcement of network policies in response to detected threats, allowing proactive mitigation actions based on predefined rules.
- FR-A-1.26 – Actionable Threat Intelligence
  - Detected anomalies are exported in STIX2 format, providing contextualized and actionable threat intelligence in near real time.
- FR-B-1.2 – Interface to security functions

- DeepGuardian components are prepared to interface with other security agents or appliances via standardize REST APIs or via asynchronous communication channels.

- FR-B-1.4 – Interface to external parties
  - The Classification-as-a-Service (CaaS) module exposes asynchronous interfaces, supporting federation and interaction with external systems.
- SR-SPL-04 – Portability and replicability
  - All components can be rapidly deployed using Helm charts, enabling reproducible and portable deployments with minimal manual effort.
- SR-SPL-10 – Secure data collection
  - Clients' data is collected by the Collector container and locally processed by the Agent. Thus, never shared with any outside instance or entity, avoiding the risk of data breach. Furthermore, for the CaaS module, where the data is shared via asynchronous communication channels, dully encryption layers will be employed to ensure the data security and privacy.

3.12.1.1. Communication interfaces

Table 79. DeepGuardian Communication Interfaces

Topic	Description	Message Sample
dg-caas-flows	Topic used to feed the CaaS with network flows to be classified.	{ "src_ip": "192.168.1.10", "dst_ip": "8.8.8.8", "src_port": 54321, "dst_port": 443, "protocol": 6, "packets": 100, "bytes": 15000, "duration": 5.2, "tcp_flags": 18, "tos": 0, "vlan_in": 0, "vlan_out": 0 }
dg-caas-anomalies	Topic used to report the detected anomalies in STIX2 format by the CaaS.	{ "type": "bundle", "id": "bundle--<uuid>", "objects": [{ "type": "observed-data", "id": "observed-data--<uuid>", "created": "2025-12-04T10:30:00.000Z",

		"first_observed": "2025-12-04T10:30:00.000Z",
		"last_observed": "2025-12-04T10:30:05.000Z",
		"number_observed": 1,
		"object_refs": ["network-traffic--<uuid>"]
		},
		{
		"type": "network-traffic",
		"id": "network-traffic--<uuid>",
		"src_ip": "192.168.1.10",
		"dst_ip": "8.8.8.8",
		"src_port": 54321,
		"dst_port": 443,
		"protocols": ["tcp"]
		}
		]
		}

dg-caas-statistics	Topic used to report statistics related with the classification process.	{
		"cluster_name": "prod-cluster",
		"namespace": "onesource",
		"deployment_name": "agent-01",
		"timestamp": "2025-12-04T10:30:00.000Z",
		"total_flows": 15000,
		"normal_flows": 14850,
		"malicious_flows": 150,
		"malicious_percentage": 1.0,
		"classification_round": 42,
		"accuracy": 0.98,
		"precision": 0.95,
		"recall": 0.92,
		"f1_score": 0.935
		}

3.12.1.2. Deployment details

All DeepGuardian components can be quickly deployed using helm charts.

3.12.1.3. Individual component testing

Table 80. DeepGuardian Component Test cases

Component: DeepGuardian - CaaS		
Test Case ID	**Description**	**Result**
TC-Comp-01	Asynchronous submission of network flows and classification via CaaS	Pass
TC-Comp-02	Publication of actionable threat intelligence and statistics by the CaaS	Pass

Table 81. Test case DP-01

Test Case ID	**TC-DG-01**	**Component**	**DeepGuardian - CaaS**
Description	Validate asynchronous submission of network flow data to the CaaS and correct anomaly/attack classification results.		
Tested by	ONE		
Associated Requirements	FR-A-1.4, FR-A-1.5, FR-A-1.26, FR-B-1.4		
Pre-condition(s)	▪ CaaS module deployed and reachable ▪ ML detection/classification models loaded ▪ Message broker or asynchronous channel configured		
Test steps			
1	Submit normal network flow data to the CaaS input topic		
2	Submit malicious/anomalous network flow data		
3	Trigger inference on the received data		
4	Consume classification results from the CaaS output topic		
Input data	Network flow records in JSON format (e.g., src/dst IP, ports, protocol, packets, bytes)		
Results	CaaS publishes classification results identifying normal and malicious traffic, along with related inference metadata.		
KPIs	Target -> Accuracy: 98%, F1-Score: 95% | Measured -> Accuracy: 99%, F1-Score: 95%		
Test Case Result	Pass		

Table 82. Test case DG-02

Test Case ID	**TC-DG-02**	**Component**	**DeepGuardian - CaaS**
Description	Validate generation and publication of actionable threat intelligence and statistics by the CaaS.		

Tested by	ONE
Associated Requirements	FR-A-1.26, FR-B-1.4, FR-B-1.27
Pre-condition(s)	▪ CaaS module deployed and reachable ▪ ML detection/classification models loaded ▪ Message broker or asynchronous channel configured ▪ STIX2 output format enabled
Test steps	
1	Feed a batch of network flows containing malicious activity into the CaaS
2	Perform classification and anomaly detection
3	Observe the detected anomalies being reported in STIX2 format
4	Observe the classification statistics (accuracy, number of malicious flows, etc.) being reported
Input data	Network flow batches in JSON format
Results	Threat intelligence events published in STIX2 format and classification statistics made available to subscribers
KPIs	N/A
Test Case Result	Pass

3.12.1.4. Next steps and future development updates

DeepGuardian - CaaS

- CaaS validation with data from MIRANDA partners
- CaaS validation in the Sandbox environment
- CaaS integration with MIRANDA UCs

3.13 Automatic Response

3.13.1. AI-based Response (SPH)

3.13.1.1. Prototype Description

This module is used for connection of all alerts generated from detection models with the attack response action. It translates enriched security alerts into concrete, actionable security policies. The AI-Based Response Module sits downstream in the MIRANDA detection-response pipeline:

Detection Layer:

- Sequence-Based Anomaly Detection
- DetectMate
- DeepGuardian

- Netflow / LoRA Anomaly Detector

Enrichment Layer:

- IoC (Indicators of Compromise) Enrichment Module: Consumes alerts from all detection modules, enriches with threat intelligence (known malicious IPs, domains, file hashes), correlates with external threat feeds, adds context from network topology, produces comprehensive enriched alerts

Response Layer (This Module):

- AI-Based Response Module: Consumes enriched alerts from IoC module, generates security policies based on threat context, outputs actionable countermeasures.

Enforcement Layer:

- VEREFOO: Run firewall security policies.
- Container Orchestrator
- Access Control Systems

Primary Input: Reads from Kafka data stream of IoC enrichment module

The IoC enrichment module provides alerts that include:

- Original Detection Data: Alert type (anomaly, signature match, policy violation), confidence/severity scores, affected resources (IPs, containers, services), timestamps and event sequences
- Threat Intelligence Enrichment: Known malicious indicators (IPs in threat feeds, blacklisted domains, malware signatures), attack classification (DDoS, SQL injection, data exfiltration), threat actor attribution (if available), historical attack patterns
- Network Context: Network topology information (affected network segments, asset criticality, trust zones), communication patterns (internal/external, protocol types), asset relationships

Processing Model

The module supports two complementary AI/ML approaches:

Option 1 - ML Classifier using Feature Extraction: Traditional machine learning approach with engineered features. Extracts structured features from enriched alerts (threat scores, IoC counts, network position). Fast inference suitable for high-volume scenarios. Trained on historical attack-response pairs.

Option 2 - AI LLM-based Model: Large Language Model approach for contextual reasoning about complex attack scenarios. Analyzes enriched alerts in natural language format. Generates human-readable policy explanations. Suitable for novel or complex attack patterns requiring contextual understanding.

Both models may be used in tandem: ML classifier for rapid triage and severity assessment, LLM for complex cases requiring detailed policy generation.

Primary Output: Security policies on firewall (for the VEREFOO input)

VEREFOO-formatted firewall rules including:

- IP-based blocking rules (block malicious IPs identified in IoC data)
- Network segmentation policies (isolate compromised segments)

- Port and protocol restrictions (limit attack surface)
- Rate limiting rules (mitigate DDoS and brute force)

Secondary Outputs: Other security policies (basic, easily implemented)

- Access Control Policies: Temporary account suspension, session termination, privilege revocation
- Container Orchestration Policies: Container restart/isolation, pod eviction, namespace restrictions
- Monitoring Adjustments: Increased logging verbosity for affected resources, alert threshold modifications, forensic data collection triggers
- Communication Policies: Email/Slack notifications to security teams, incident ticket creation, escalation workflows

Requirements Mapping:

- FR-A-1.18: The system must be able to autonomously and proactively respond to security warning and alerts according to pre-defined policies.
- FR-A-1.19: The system may be able to autonomously learn the best response strategy from past activity and decisions and apply it to similar conditions
- FR-A-1.22: The system may be able to trigger management operation in response to security warnings or alerts
- FR-A-1.23: The system must trigger autonomous response at least on the following events: warning or alert from cybersecurity appliance; warning or alert from internal detection process; changes in chain topology or composition; user-defined conditions (thresholds on monitored data, risk level, trust values)
- FR-A-1.24: The system should evaluate the following conditions when taking (semi-)autonomous decisions: service topology, threat landscape, monitoring data (logs, network traces, system measures), warning/alarms from cyber-security appliances and processes,
- FR-A-1.25: The system may automatically derive response policies from (actionable) threat intelligence

3.13.1.2. Communication interfaces

The designed API that will be implemented will contains the following REST Endpoints as shown in the table below:

Table 83. AI-based Response Communication interfaces (HTTP)

#	Method	REST Endpoint	Description	Request Body	Response Body
1	POST	/analyze-alert	Submit IoC enriched alert to return suggested security policy	Alert after IoC enrichment	Suggested Security policies
2	POST	/analyze-alerts	Submit batch of IoC enriched alerts to return suggested security policy	Batch of alerts after IoC enrichment	Suggested Security policies

3	GET	/ policies/{alert_id}	Retrieve generated policies for specific alert		Suggested Security policies
4	POST	/response-feedback	Submit policy effectiveness feedback after end-user validation	Accept/Decline suggested security policy	

Table 84. AI-based Response Communication interfaces (Kafka)

Topic	Description	Message Sample
ioc-extraction	Read data of IoC extraction module	

Security-policies	Publish results on this Kafka topic	{
		AlertID: “text”,
		SecurityPolicy: “text”,
		ConfidenceScore: float
		ModelID: “text”,
		Timestamp: timestamp creation
		}

3.13.1.3. Deployment details

The component is currently on designing phase TRL 2, so the deployment to Kubernetes is on the next steps.

3.13.1.4. Individual component testing

The proposed test cases for this module are included in the following table.

Table 85. AI-based Response Component Test cases

Component: Test Cases		
Test Case ID	Description	Result
TC-AIR-01	IoC Alert Consumption from Kafka	To be tested
TC-AIR-02	Malicious IP Detection and Firewall Policy Generation	To be tested
TC-AIR-03	VEREFOO Policy Format Validation	To be tested
TC-AIR-04	AI Model Performance Metrics	To be tested
TC-AIR-05	Policy Generation Performance and Latency	To be tested
TC-AIR-06	REST API Endpoint Functionality	To be tested

TC-AIR-07	End-to-End Integration: IoC Enrichment to Firewall Deployment	To be tested

3.13.1.5. Next steps and future development updates

- Baseline model for integration purposes
- Experiments of what type of AI Model will be used.
- Performance of the selected model with MIRANDA data.
- Dockerize the code
- Deployment on Kubernetes cluster

3.14 Threat Hunting

3.14.1. Prototype Description

The *threat hunting* tool oversees the prediction of the (lateral) movements of an attacker inside a target network, allowing the administrators to anticipate their moves and take pre-emptive action against the damage.

This component leverages the power and flexibility of GNNs (Graph Neural Networks) to predict a multi-step attack.

The preliminary training workflow for this component is shown in Figure 19.

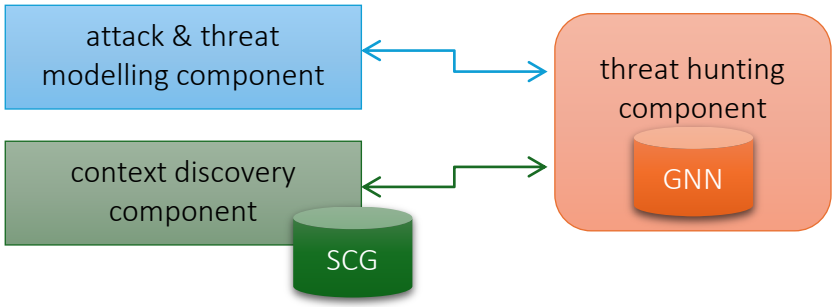

Figure 19. Training interactions for the threat hunting tool.

During the GNN training phase, the threat hunting component uses the SCG extracted by the context discovery component, paired with all potential attack paths discovered by the attack & threat modelling component, to train the internal neural network. Internally, the GNN learns how an attack can traverse the network (modelled by the SCG) and performs any possible combination of multi-step attacks. This training phase is performed across a variety of SCGs to ensure the GNN is sufficiently adaptable to any network configuration with any attack path. The training is computationally intensive, but it is performed off-line.

Once a suitable trained GNN is available, the threat hunting component can be deployed in a target network to start analysing it, as shown in Figure 20Figure 20.

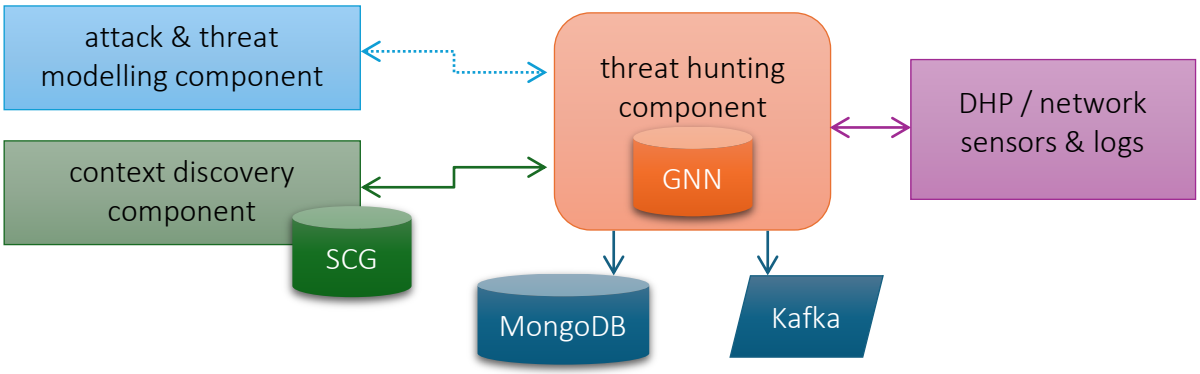

Figure 20. Inference interactions for the threat hunting tool.

When the threat hunting component is running on a network, it actively listens for any network changes (thanks to the context discovery service), since the network shape determines how an attacker can mount an attack. In addition, it constantly monitors data from various traffic and log sensors deployed across the network (e.g., via the data handling pipeline) to assess the network's overall status. The sensor data is used to detect anomalies or single attack steps that an attacker is currently mounting in the network; the SCG and this information are then fed into the GNNs, which output zero (no attack underway) or one or more potential attack paths the attacker might try to exploit. The information about the potential attacker's movements is then published to Kafka or inserted into MongoDB collection for later review and as a historical log.

The threat hunting component fulfils the following requirements:

- FR-A-1.4 – Continuous detection: The threat hunting component constantly monitors the status of the network and reacts when a potential multi-stage attack is underway.
  FR-A-1.5 – Detection capabilities: The threat hunting component builds upon a variety of other detection tools and will offer a hight accuracy with real-time responsiveness.

3.14.2. Communication interfaces

The threat hunting component offers a REST we API (via the Flask[6] Python component) to manage its lifecycle. The table below lists such methods.

Table 86. Threat Hunting Communication Interfaces

#	Method	REST Endpoint	Description	Request Body	Response Body
1	GET	`/thunting/status`	Retrieve the status of the threat hunting service		`{ "status": "text", "uptime": "time" }`

[6] https://flask.palletsprojects.com/

2	POST	/thunting/start	Start the threat hunting process		{ "status": "text" }
3	POST	/thunting/refresh	Request an update of the SCG		{ "status": "text" }
4	POST	/thunting/stop	Stop the threat hunting process		{ "status": "text" }

In addition, when a suspected attack is underway, the component publishes the predicted attack paths in a MongoDB collection and on a Kafa topic. The output is a JSON object with a structure similar to the following example:

```
{
      "timestamp": "time",     # Timestamp of the detection
      "attacks": [       # List of predicted attack paths
            [{"service" : "text", "vulnerability": "text"},
                  {"service" : "text", "vulnerability": "text"},
                  …
            ],
            …
      ]
}
```

3.14.3. Deployment details

The threat hunting tool is a pure Python application and can be deployed in a variety of ways.

Currently, the main way to use and deploy the component is via Poetry[7]. The following commands can be used to install it and configure automatically all its dependencies:

```
cd thunting
poetry install   # Create the virtual environment and install dependencies
```

Once installed, the threat hunting service can be launched using the command:

```
Poetry run thunting.py
```

This command reads a configuration with a format similar to the following example:

```
ctxd:            # Configuration for the context discovery service
  host: "127.0.0.1"
  port: 27017
  db_name: "miranda"
  collection: "ctxd"
  user: "miranda"
  pass: "xxxx"
attacks:         # Configuration for the attacks & threat modelling service
  url: "http://127.0.0.1:1234"
publishers:      # Configuration for publishing the data
  mongodb:
    host: "127.0.0.1"
```

[7] https://python-poetry.org/

```
    port: 27017
    db_name: "miranda"
    collection: "thunting"
    user: "miranda"
    pass: "xxxx"
  kafka:
    host: "127.0.0.1"
    port: 29092
    topic: "demo"
```

In the future, the threat hunting tool will be also made available as a Docker container.

3.14.4. Individual component testing

Table 87. Threat Hunting Component Test cases

Component: Test Cases		
Test Case ID	**Description**	**Result**
TC-ThreatHunting-01	Test the GNN training convergence	To be tested
TC-ThreatHunting-02	Test the accuracy of the predictions	To be tested
TC-ThreatHunting-03	Test the speed of the predictions	To be tested

Table 88. Test case TH-01

Test Case ID	**TC-TH-01**	**Component**	**Threat Hunting**
Description	Check that the internal threat hunting neural network (GNN) is learning from the used datasets		
Tested by	CNR (component owner)		
Associated Requirements	FR-A-1.5		
Pre-condition(s)	Training set available (built using multiple different SCGs and attack paths found using the attack & threat modelling tool)		
Test steps			
1	Train the GNN		
2	Check the GNN convergence (e.g. using the learning curves)		
3	Adjust the GNN hyperparameters and retrain, if necessary		
Input data	Dataset containing one or more SCGs annotated with multiple attack paths		
Results	Learning curves		
KPIs	Target: The overall trend of the learning curves must be decreasing towards a zero loss		
Test Case Result	Pass / Failed		

Table 89. Test case TH-02

Test Case ID	TC-ThreatHunting-02	Component	Threat Hunting
Description	Evaluate the accuracy of the predictions		
Tested by	CNR (component owner)		
Associated Requirements	FR-A-1.5		
Pre-condition(s)	Test set available (built using multiple different SCGs and attack paths found using the attack & threat modelling tool) A trained GNN		
Test steps			
1	Test the GNN on the test set		
2	Compute a variety of standard machine-learning metrics and plots (e.g. accuracy, AUC, recall, etc)		
Input data	Dataset containing one or more SCGs annotated with multiple attack paths		
Results	Evaluation metrics and plots		
KPIs	Target: The vast majority of the evaluation metrics should be at least 70% (most of the statistics are percentages).		
Test Case Result	Pass / Failed		

Table 90. Test case TH-03

Test Case ID	TC-ThreatHunting-03	Component	Threat Hunting
Description	Check that the threat hunting service can detect an attacker movement in a network with near real-time speed		
Tested by	CNR (component owner)		
Associated Requirements	FR-A-1.4		
Pre-condition(s)	A trained GNN A testbed network Tools to emulate an attack		
Test steps			
1	Emulate some attacks on a testbed network where the threat hunting tool and its required services are running		
2	Compute the detection speed of the attacks		
Input data	Test bed with some emulated attacks		
Results	Detection rate of the attacks and their detection times		
KPIs	Target: The detection time of the multi-stage attacks should be less than 500 ms.		

Test Case Result	Pass / Failed

3.14.5. Next steps and future development updates

The threat hunting component is currently under active development and it is expected to be fully functional towards the later part of the project.

Ongoing activities focus on designing and constructing datasets required to train and test the internal Graph Neural Network (GNN).

In the future, a Docker-based containerisation version will also be available to further ease deployment.

At later stages of the development, the component is expected also to incorporate continuous learning functionalities, enabling the progressive update of the models as new data become available. These capabilities are currently under investigation and will be addressed in subsequent implementation phases.

3.15 Actionable CTI

3.15.1. Indicators of Compromise Extraction

3.15.1.1. Prototype Description

Overview:

The Indicators of Compromise Extraction Component, developed for the purposes of the MIRANDA project, is designed to extract IoCs from alerts generated by DetectMate and Space Hellas' anomaly detectors, as well as from alerts produced by the DeepGuardian detector. It provides contextual information and additional enrichments (when applicable) derived from ingested CTI sources and from the MISP Threat Sharing Platform (via the SPH Threat Sharing API). The component outputs the extracted Indicators of Compromise and associated enrichments in JSON format (ECS-compatible) and in a STIX-compatible format containing the IoCs in the appropriate STIX object types.

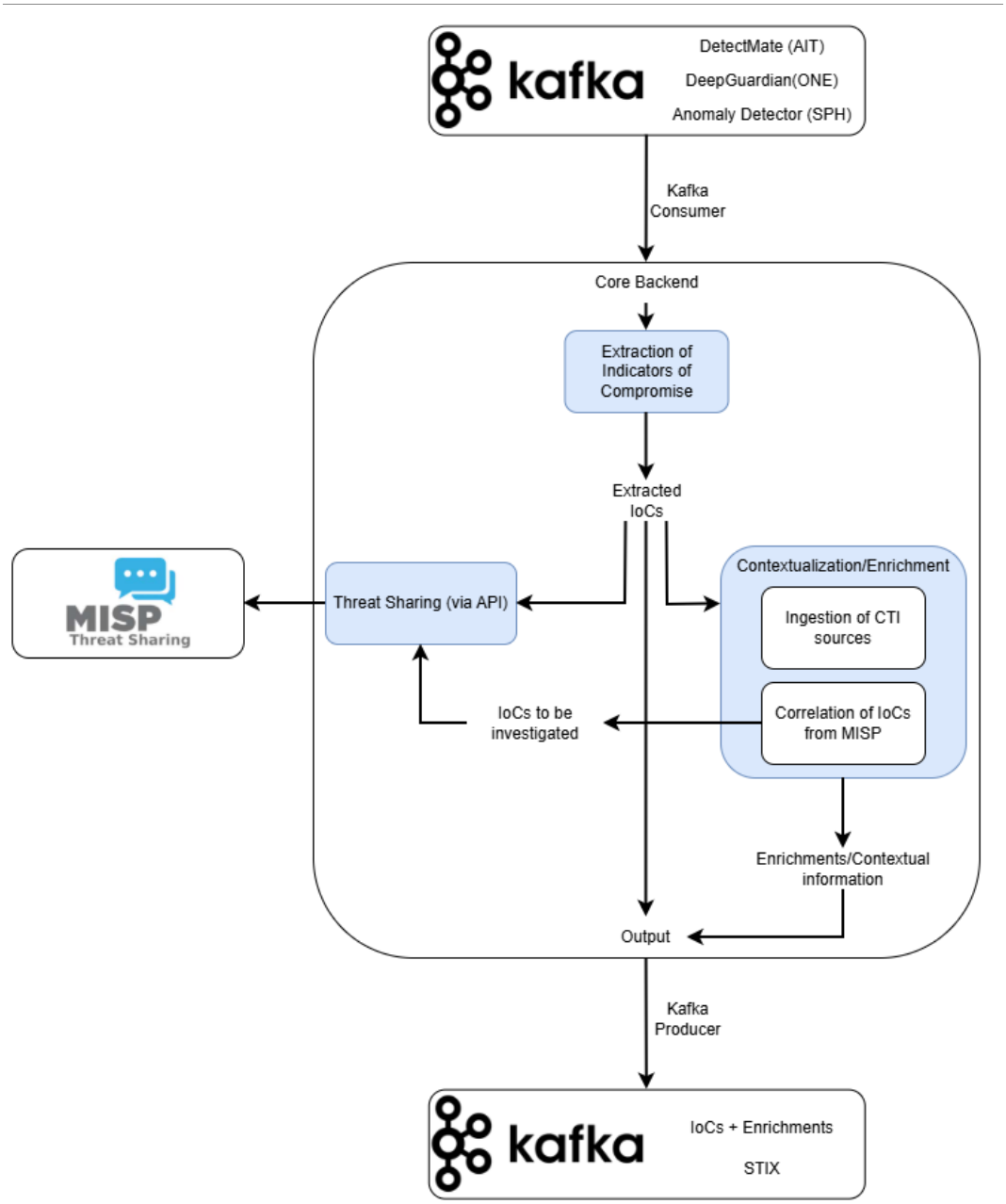

Figure 21. 3.14.1. Indicators of Compromise Extraction Architecture

Design and workflow:

The IoC extraction component is a containerised, service-oriented application that operates without end-user interaction and focuses on the automation of IoC extraction, contextualization, and sharing processes. It is designed to process multiple alerts concurrently to improve performance and to pipeline data processing across all processing stages, enabling near-real-time processing. The system consists solely of a backend instance and communicates with external sources via Kafka consumers and producers, as well as through the Threat Sharing API developed by SPH.

The component is developed in Python and uses the Confluent Kafka Python library to implement and configure interactions with Kafka. It also leverages the PyMISP library to ensure

compatibility with the Threat Sharing API and/or direct integration with the MISP API. Additionally, the STIX2 library is used to format the extracted IoCs and to generate STIX-compliant output. During pre-flight operations, the component ingests CTI sources to support enrichment and contextualization. The component is containerized and runs as a Docker service, with all required library dependencies installed during the Docker image build process. Figure 21 illustrates the high-level workflow design.

Requirements:

- FR-A-1.26 - Actionable Threat Intelligence: The system must generate threat intelligence in an actionable form, namely distilled, contextual and real-time data.
- FR-A-1.32 - Adaptive parsing: The system must provide autonomous parsing capabilities that extract structured data from text logs suitable for machine processing
- FR-B-1.29 - Data manipulation: The system should be able to perform manipulation and enrichment operations (e.g.: encrypt and sign data; add timestamps; calculate elapsed times; add geotagging; split complex elements into elementary data; translate or transform fields according to replacement patterns; direct/inverse DNS lookup; aggregate related events into a single record; parse command-separated values into individual fields; parse string representations into their numeric values; extract unstructured event data into fields using delimiters)
- FR-B-1.38 - Execution environment for MIRANDA Connector and Controller: The system must provide a Python >= 3.10

3.15.1.2. Communication interfaces

The IoC extraction and contextualization component does not expose any user-facing interfaces. It communicates exclusively via Kafka consumers and producers for data streaming and through APIs toward MISP and does not expose any API endpoints for external interaction.

Table 91. Indicators of Compromise Extraction Communication Interfaces

Topic	Description	Message Sample
threat_intel	Output of component that include the extracted IoCs and the enrichments and contextual information	{ "Original-Alert": { "type": "bundle", "id": "bundle--94de5219-eb02-4953-bfe5-b7d6ef0863b6", "testbed": "ONE", "objects": [{ "type": "anomalies", "spec_version": "2.1", "id": "anomalies--0e8cf689-a5db-4ee2-b8fd-4bd604a075a9", "created": "2024-11-11T12:41:34.232502Z",

"modified": "2024-11-11T12:41:34.232502Z",
"value": "0.7682093512385718",
"value_type": "reconstruction_error",
"flow_id": "185",
"source_ip": "10.1.1.210",
"source_port": "1883",
"destination_ip": "10.42.2.25",
"destination_port": "37915",
"flow_data": "[185, '10.1.1.185', 1883, '10.42.2.116', 37915, 6, '2024-11-11 12:41:28.603', 0.0, 1, 0, 66, 0, 66, 66, 66.0, 0.0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0, 0.0, 0, 0.0, 0, 0.0, 0.0, 0, 0, 0, 0.0, 0.0, 0, 0, 0, 0, 0, 0, 52, 0, 0.0, 0, 66, 66, 66.0, 0.0, 0.0, 0, 0, 0, 0, 1, 0, 0, 0, 0.0, 66.0, 66.0, 0.0, 0.0, 0.0, 0.0, 0, 0, 0, 1, 66, 0, 0, 12, 0, 0, 66, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1]",
"timestamp": "2024-11-11 12:41:28.603",
"label": "anomaly"
},
{
"type": "model",
"spec_version": "2.1",
"id": "model--3ee09226-d9da-47fe-8ec7-24585b0d1de3",
"created": "2024-11-11T12:41:34.293635Z",
"modified": "2024-11-11T12:41:34.293635Z",
"model_id": "autoencoder_v0.0",
"model_name": "autoencoder_model",
"learning_type": "unsupervised",
"flow_features": "['id', 'src_ip', 'src_port', 'dst_ip', 'dst_port', 'protocol', 'timestamp', 'udps.flow_duration', 'src2dst_packets', 'dst2src_packets', 'udps.totlen_fwd_pkts', 'udps.totlen_bwd_pkts', 'src2dst_max_ps', 'src2dst_min_ps', 'src2dst_mean_ps', 'src2dst_stddev_ps', 'dst2src_max_ps', 'dst2src_min_ps', 'dst2src_mean_ps', 'dst2src_stddev_ps', 'udps.flow_byts_s', 'udps.flow_pkts_s', 'udps.flow_iat_max',

		'udps.flow_iat_mean', 'udps.flow_iat_min', 'udps.flow_iat_std', 'udps.fwd_iat_tot', 'src2dst_mean_piat_ms', 'src2dst_stddev_piat_ms', 'src2dst_max_piat_ms', 'src2dst_min_piat_ms', 'udps.bwd_iat_tot', 'dst2src_mean_piat_ms', 'dst2src_stddev_piat_ms', 'dst2src_max_piat_ms', 'dst2src_min_piat_ms', 'src2dst_psh_packets', 'dst2src_psh_packets', 'src2dst_urg_packets', 'dst2src_urg_packets', 'udps.fwd_header_len', 'udps.bwd_header_len', 'udps.fwd_pkts_s', 'udps.bwd_pkts_s', 'udps.pkt_len_min', 'udps.pkt_len_max', 'udps.pkt_len_mean', 'udps.pkt_len_std', 'udps.pkt_len_var', 'udps.fin_flag_cnt', 'udps.syn_flag_cnt', 'udps.rst_flag_cnt', 'udps.psh_flag_cnt', 'udps.ack_flag_cnt', 'udps.urg_flag_cnt', 'udps.cwe_flag_cnt', 'udps.ece_flag_cnt', 'udps.down_up_ratio', 'udps.pkt_size_avg', 'udps.fwd_seg_size_avg', 'udps.bwd_seg_size_avg', 'udps.fwd_byts_b_avg', 'udps.fwd_pkts_b_avg', 'udps.fwd_blk_rate_avg', 'udps.bwd_byts_b_avg', 'udps.bwd_pkts_b_avg', 'udps.bwd_blk_rate_avg', 'udps.subflow_fwd_pkts', 'udps.subflow_fwd_byts', 'udps.subflow_bwd_pkts', 'udps.subflow_bwd_byts', 'udps.init_fwd_win_byts', 'udps.init_bwd_win_byts', 'udps.fwd_act_data_pkts', 'udps.fwd_seg_size_min', 'udps.active_mean', 'udps.active_std', 'udps.active_max', 'udps.active_min', 'udps.idle_mean', 'udps.idle_std', 'udps.idle_max', 'udps.idle_min', 'bidirectional_packets']", "threshold": "0.7682089320012652", "binary_class_accuracy": "87.00", "multi_class_accuracy": "0.0", "comparison_metric": "mse", "framework": "tensorflow", "framework_version": "v2.14.0", "collection_tool": "nfstream", "collection_tool_version": "v6.5.3", "training_timestamp": "20241111124134" }, { "type": "relationship", "spec_version": "2.1",

"id": "relationship--8f2ddd37-9b42-4ed3-ae03-a72d067d5608",
"created": "2024-11-11T12:41:34.294327Z",
"modified": "2024-11-11T12:41:34.294327Z",
"relationship_type": "indicates",
"source_ref": "model--3ee09226-d9da-47fe-8ec7-24585b0d1de3",
"target_ref": "anomalies--0e8cf689-a5db-4ee2-b8fd-4bd604a075a9"
}
]
},
"Indicators-Of-Compromise": {
"network": {
"source": {
"ip": "10.1.1.210",
"port": "1883"
},
"destination": {
"ip": "10.42.2.25",
"port": "37915"
},
"protocol": {
"id": "6",
"name": "TCP",
"description": "Transmission Control"
}
},
"eventID": "432fffc6-37ad-483a-8d38-d4347a0c38e1",
"CTI_context": {
"Mitre": {},
"AIT": {},
"CISA": {
"TTP": []
},
"enrichment": {},
"eventID": "432fffc6-37ad-483a-8d38-d4347a0c38e1"

```
                }
        },
        "MISP": {
                "1883": [
                        {
                                "Event": {
                                        "id": "171",
                                        "info": "Event ID: 87b90298-ec85-407e-99ec-4b11b26050e4",
                                        "org_id": "1",
                                        "orgc_id": "1",
                                        "uuid": "31bf794e-8bbf-4c31-98fb-80c8bb4e1f3a",
                                        "user_id": "1",
                                        "threat_level_id": "4",
                                        "distribution": "1",
                                        "analysis": "0",
                                        "date": "2026-01-14",
                                        "timestamp": "1768380763",
        "publish_timestamp": "0",
                                        "Org": {
                                                "id": "1",
                                                "name": "ADMIN",
                                                "uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
                                        },
                                        "Orgc": {
                                                "id": "1",
                                                "name": "ADMIN",
                                                "uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
                                        },
                                        "ThreatLevel": ""
                                },
                                "category": "Network activity",
                                "type": "port"
                        },
```

```
{
    "Event": {
        "id": "198",
        "info": "Event ID: 74b9daf3-37a0-4e1d-8b4f-5e673a0745c4",
        "org_id": "1",
        "orgc_id": "1",
        "uuid": "7c921722-93db-40a7-8e05-c9e93a9463db",
        "user_id": "1",
        "threat_level_id": "4",
        "distribution": "1",
        "analysis": "0",
        "date": "2026-01-14",
        "timestamp": "1768396337",
        "publish_timestamp": "0",
        "Org": {
            "id": "1",
            "name": "ADMIN",
            "uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
        },
        "Orgc": {
            "id": "1",
            "name": "ADMIN",
            "uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
        },
        "ThreatLevel": ""
    },
    "category": "Network activity",
    "type": "port"
},
{
    "Event": {
        "id": "200",
```

"info": "Event ID: cf3105e7-0932-49f2-ae32-0410dbf5fc6f",
"org_id": "1",
"orgc_id": "1",
"uuid": "b35b5a73-631c-46b7-a8fe-6436aa6ac641",
"user_id": "1",
"threat_level_id": "4",
"distribution": "1",
"analysis": "0",
"date": "2026-01-30",
"timestamp": "1769774955",
"publish_timestamp": "0",
"Org": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"Orgc": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"ThreatLevel": ""
},
"category": "Network activity",
"type": "port"
}
],
"37915": [
{
"Event": {
"id": "171",
"info": "Event ID: 87b90298-ec85-407e-99ec-4b11b26050e4",

"org_id": "1",
"orgc_id": "1",
"uuid": "31bf794e-8bbf-4c31-98fb-80c8bb4e1f3a",
"user_id": "1",
"threat_level_id": "4",
"distribution": "1",
"analysis": "0",
"date": "2026-01-14",
"timestamp": "1768380763",
"publish_timestamp": "0",
"Org": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"Orgc": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"ThreatLevel": ""
},
"category": "Network activity",
"type": "port"
},
{
"Event": {
"id": "198",
"info": "Event ID: 74b9daf3-37a0-4e1d-8b4f-5e673a0745c4",
"org_id": "1",
"orgc_id": "1",
"uuid": "7c921722-93db-40a7-8e05-c9e93a9463db",

"user_id": "1",
"threat_level_id": "4",
"distribution": "1",
"analysis": "0",
"date": "2026-01-14",
"timestamp": "1768396337",
"publish_timestamp": "0",
"Org": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"Orgc": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"ThreatLevel": ""
},
"category": "Network activity",
"type": "port"
},
{
"Event": {
"id": "200",
"info": "Event ID: cf3105e7-0932-49f2-ae32-0410dbf5fc6f",
"org_id": "1",
"orgc_id": "1",
"uuid": "b35b5a73-631c-46b7-a8fe-6436aa6ac641",
"user_id": "1",
"threat_level_id": "4",
"distribution": "1",

"analysis": "0",
"date": "2026-01-30",
"timestamp": "1769774955",
"publish_timestamp": "0",
"Org": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"Orgc": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"ThreatLevel": ""
},
"category": "Network activity",
"type": "port"
}
],
"10.1.1.210": [
{
"Event": {
"id": "171",
"info": "Event ID: 87b90298-ec85-407e-99ec-4b11b26050e4",
"org_id": "1",
"orgc_id": "1",
"uuid": "31bf794e-8bbf-4c31-98fb-80c8bb4e1f3a",
"user_id": "1",
"threat_level_id": "4",
"distribution": "1",
"analysis": "0",

"date": "2026-01-14",
"timestamp": "1768380763",
"publish_timestamp": "0",
"Org": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"Orgc": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"ThreatLevel": ""
},
"category": "Network activity",
"type": "ip-src"
},
{
"Event": {
"id": "198",
"info": "Event ID: 74b9daf3-37a0-4e1d-8b4f-5e673a0745c4",
"org_id": "1",
"orgc_id": "1",
"uuid": "7c921722-93db-40a7-8e05-c9e93a9463db",
"user_id": "1",
"threat_level_id": "4",
"distribution": "1",
"analysis": "0",
"date": "2026-01-14",
"timestamp": "1768396337",

"publish_timestamp": "0",
"Org": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"Orgc": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"ThreatLevel": ""
},
"category": "Network activity",
"type": "ip-src"
},
{
"Event": {
"id": "200",
"info": "Event ID: cf3105e7-0932-49f2-ae32-0410dbf5fc6f",
"org_id": "1",
"orgc_id": "1",
"uuid": "b35b5a73-631c-46b7-a8fe-6436aa6ac641",
"user_id": "1",
"threat_level_id": "4",
"distribution": "1",
"analysis": "0",
"date": "2026-01-30",
"timestamp": "1769774955",

"publish_timestamp": "0",
"Org": {
"id": "1",

"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"Orgc": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"ThreatLevel": ""
},
"category": "Network activity",
"type": "ip-src"
}
],
"10.42.2.25": [
{
"Event": {
"id": "171",
"info": "Event ID: 87b90298-ec85-407e-99ec-4b11b26050e4",
"org_id": "1",
"orgc_id": "1",
"uuid": "31bf794e-8bbf-4c31-98fb-80c8bb4e1f3a",
"user_id": "1",
"threat_level_id": "4",
"distribution": "1",
"analysis": "0",
"date": "2026-01-14",
"timestamp": "1768380763",
"publish_timestamp": "0",
"Org": {
"id": "1",
"name": "ADMIN",

"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"Orgc": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"ThreatLevel": ""
},
"category": "Network activity",
"type": "ip-dst"
},
{
"Event": {
"id": "198",
"info": "Event ID: 74b9daf3-37a0-4e1d-8b4f-5e673a0745c4",
"org_id": "1",
"orgc_id": "1",
"uuid": "7c921722-93db-40a7-8e05-c9e93a9463db",
"user_id": "1",
"threat_level_id": "4",
"distribution": "1",
"analysis": "0",
"date": "2026-01-14",
"timestamp": "1768396337",
"publish_timestamp": "0",
"Org": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"Orgc": {

"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"ThreatLevel": ""
},
"category": "Network activity",
"type": "ip-dst"
},
{
"Event": {
"id": "200",
"info": "Event ID: cf3105e7-0932-49f2-ae32-0410dbf5fc6f",
"org_id": "1",
"orgc_id": "1",
"uuid": "b35b5a73-631c-46b7-a8fe-6436aa6ac641",
"user_id": "1",
"threat_level_id": "4",
"distribution": "1",
"analysis": "0",
"date": "2026-01-30",
"timestamp": "1769774955",
"publish_timestamp": "0",
"Org": {
"id": "1",
"name": "ADMIN",
"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"Orgc": {
"id": "1",
"name": "ADMIN",

"uuid": "4a4b27d9-021a-4952-a53b-c6419f7d0655"
},
"ThreatLevel": ""
},
"category": "Network activity",
"type": "ip-dst"
}
]
},
"CTI_context": {
"Mitre": {},
"AIT": {},
"CISA": {
"TTP": []
},
"enrichment": {},
"eventID": "432fffc6-37ad-483a-8d38-d4347a0c38e1"
},
"STIX": {
"network.source.ip": {
"type": "indicator",
"spec_version": "2.1",
"id": "indicator--3058ba37-640d-4b88-b45c-d8775bc849f5",
"created": "2026-01-30T12:09:19.222508Z",
"modified": "2026-01-30T12:09:19.222516Z",
"name": "{'indicator_name': 'Malicious ip address', 'indicator_patternV4': 'ipv4-addr:value', 'indicator_patternV6': 'ipv6-addr:value'}",
"pattern": "[ipv4-addr:value = '10.1.1.210']",
"pattern_type": "stix",
"pattern_version": "2.1",
"valid_from": "2026-01-30T12:09:19.222517Z"
},
"network.destination.ip": {
"type": "indicator",

"spec_version": "2.1",
"id": "indicator--3bf11502-4636-44ee-b252-89c9e738301d",
"created": "2026-01-30T12:09:19.223619Z",
"modified": "2026-01-30T12:09:19.223622Z",
"name": "{'indicator_name': 'Malicious ip address', 'indicator_patternV4': 'ipv4-addr:value', 'indicator_patternV6': 'ipv6-addr:value'}",
"pattern": "[ipv4-addr:value = '10.42.2.25']",
"pattern_type": "stix",
"pattern_version": "2.1",
"valid_from": "2026-01-30T12:09:19.223623Z"
},
"relationships": [
{
"type": "indicator",
"spec_version": "2.1",
"id": "indicator--15214ddf-0b99-452a-a729-83dc24c69378",
"created": "2026-01-30T12:09:19.224208Z",
"modified": "2026-01-30T12:09:19.224208Z",
"name": "Suspicious IP traffic",
"description": "Traffic from 10.1.1.210 on port 1883 towards 10.42.2.25 on port 37915",
"pattern": "[network-traffic:src_ref.value = '10.1.1.210' AND network-traffic:dst_ref.value = '10.42.2.25' AND network-traffic:protocols[*] = 'TCP']",
"pattern_type": "stix",
"pattern_version": "2.1",
"valid_from": "2026-01-30T12:09:19.224201Z"
}
],
"eventID": "432fffc6-37ad-483a-8d38-d4347a0c38e1"
}
}

stix	Stix compliant output of the component	{
		"network.source.ip": {
		"type": "indicator",
		"spec_version": "2.1",
		"id": "indicator--2ed224a3-c049-47eb-bcc4-0ec7625d6b32",
		"created": "2026-01-30T12:12:56.696035Z",
		"modified": "2026-01-30T12:12:56.696042Z",
		"name": "{'indicator_name': 'Malicious ip address', 'indicator_patternV4': 'ipv4-addr:value', 'indicator_patternV6': 'ipv6-addr:value'}",
		"pattern": "[ipv4-addr:value = '10.1.1.210']",
		"pattern_type": "stix",
		"pattern_version": "2.1",
		"valid_from": "2026-01-30T12:12:56.696043Z"
		},
		"network.destination.ip": {
		"type": "indicator",
		"spec_version": "2.1",
		"id": "indicator--718fd706-882b-4114-abfa-b78187993cac",
		"created": "2026-01-30T12:12:56.697392Z",
		"modified": "2026-01-30T12:12:56.697395Z",
		"name": "{'indicator_name': 'Malicious ip address', 'indicator_patternV4': 'ipv4-addr:value', 'indicator_patternV6': 'ipv6-addr:value'}",
		"pattern": "[ipv4-addr:value = '10.42.2.25']",
		"pattern_type": "stix",
		"pattern_version": "2.1",
		"valid_from": "2026-01-30T12:12:56.697395Z"
		},
		"relationships": [
		{
		"type": "indicator",
		"spec_version": "2.1",
		"id": "indicator--7a005bdc-e177-4b85-81ad-31ffca7707ab",
		"created": "2026-01-30T12:12:56.698334Z",
		"modified": "2026-01-30T12:12:56.698334Z",

		"name": "Suspicious IP traffic", "description": "Traffic from 10.1.1.210 on port 1883 towards 10.42.2.25 on port 37915", "pattern": "[network-traffic:src_ref.value = '10.1.1.210' AND network-traffic:dst_ref.value = '10.42.2.25' AND network-traffic:protocols[*] = 'TCP']", "pattern_type": "stix", "pattern_version": "2.1", "valid_from": "2026-01-30T12:12:56.698324Z" }], "eventID": "d7de68a9-2dd0-42ac-a546-a50424100271" }

3.15.1.3. Deployment details

The deployment of the IoC extraction component requires the Dockerfile that copies the application's code installs the requirements

```
# Use an official Python runtime as a parent image
FROM python:3.11.2

# Set the working directory in the container
WORKDIR /app

# Copy the requirements.txt file into the container at /app
COPY ./requirements.txt /app/requirements.txt

# Install dependencies from the requirements.txt
RUN pip install --upgrade pip && pip install -r /app/requirements.txt

# Copy the .env file into the container
#COPY ./.env /app/.env

# Copy the application code into the container
COPY miranda /app/miranda

# Command to run the API script (ctiv2.py)
CMD ["python3", "/app/miranda","--ext","--misp","--stix","--enr"]
```

Additionally, we need to build the environmental variables

```
###################################BUILD#####################################
KAFKA_ADDRESS='<KAFKA ADDRESS>
MISP_BASE_URL=<MISP ADRESS>
##enrichments
…
##stix
…
##inputs
```

Also there is the need for making the configuration in the docker-compose.yml file and set the appropriate flags in the command:

```
  miranda_service:
    image: miranda_image_refactored
    tty: true
    restart: always
    container_name: miranda_component
    working_dir: /app
    env_file: .env
    networks:
      - misp-docker_default
      - netflow_parser_netflow_net
    volumes:
      - ./inputs:/app/inputs
      - ./sources:/app/sources
      - ./enrichments:/app/enrichments
      - ./stix_templates:/app/stix_templates
    command: ["python3", "-m", "miranda", "--ext", "--misp", "--stix", "--
enr"]
```

3.15.1.4. Individual component testing

Table 92. IoC Extraction Component Test Cases

IoC Extraction Component: Test Cases		
IoC-EXT-01	Test the kafka and MISP configurations	Achieved
IoC-EXT-02	Test kafka communication	Achieved
IoC-EXT-03	Test IoC extraction	Achieved
IoC-EXT-04	Test Enrichment	Achieved
IoC-EXT-05	Test MISP functionality	Achieved
IoC-EXT-06	Test Preparation	Achieved
IoC-EXT-07	Test Processing	Achieved
IoC-EXT--08	Test Build STIX	Achieved

Table 93. Test case ioC-EXT-01

Test Case ID	IoC-EXT-01	Component	IoC-Extraction
Description	Verify component functionality after the kafka and MISP configurations		
Tested by	LOGSTAIL		
Associated Requirements	-		
Pre-condition(s)	The kafka and MISP instances are accessible from the container.		
Test steps			
1	Mock or set the correct addresses and auth keys		

2	Create a fake producer
Input data	JSON object
Results	Ensure response in the kafka.
KPIs	Target → Ensure kafka and MISP connection establishment
Test Case Result	Pass

Table 94. Test case ioC-EXT-02

Test Case ID	IoC-EXT-02	Component	IoC-Extraction
Description	Verify component communication with the kafka instance		
Tested by	LOGSTAIL		
Associated Requirements	-		
Pre-condition(s)	The kafka instance is accessible from the container.		
Test steps			
1	Create a fake producer		
Input data	JSON object		
Results	Ensure response in the kafka.		
KPIs	Target → Ensure kafka connection establishment		
Test Case Result	Pass		

Table 95. Test case ioC-EXT-03

Test Case ID	IoC-EXT-03	Component	IoC-Extraction
Description	Verify correct extraction of Indicators of compromise from the DetectMate alerts		
Tested by	LOGSTAIL		
Associated Requirements	FR-A-1.26 - Actionable Threat Intelligence FR-A-1.32 - Adaptive parsing		
Pre-condition(s)	The application has finished preflight processes and is running		
Test steps			
1	Mock a DetectMate alert		
2	Test the Processing of the mocked alert		
Input data	JSON object		
Results	Validate the result of the extraction processes		

KPIs	Target → all the IoCs are extracted
Test Case Result	Pass

Table 96. Test case ioC-EXT-04

Test Case ID	IoC-EXT-04	Component	IoC-Extraction
Description	Verify correct enrichment of the extracted IoCs		
Tested by	LOGSTAIL		
Associated Requirements	FR-A-1.26 - Actionable Threat Intelligence FR-B-1.29 - Data manipulation		
Pre-condition(s)	The CTI sources are mounted as volumes		
Test steps			
1	Mock an executable		
2	Find enrichments from the component's enrichment functionality		
Input data	Mocked executable name		
Results	Validate the result of the enrichment processes		
KPIs	Target → Ensure the enrichments are suitable for this exe		
Test Case Result	Pass		

Table 97. Test case ioC-EXT-05

Test Case ID	IoC-EXT-05	Component	IoC-Extraction
Description	Test MISP's post and search functionality		
Tested by	LOGSTAIL		
Associated Requirements	-		
Pre-condition(s)	The MISP instance is accessible from the container		
Test steps			
1	Mock MISP type atttributes		
2	Create a MISP client		
3	Post the attribute to MISP Platform		
4	Search the attribute in the MISP platform		
Input data	Mocked MISP attribute		
Results	Validate the attribute is posted on the platform		

	Validate the result from the search is fetched from the platform
KPIs	Target → Ensure the post and search from the platform function as expected.
Test Case Result	Pass

Table 98. Test case ioC-EXT-06

Test Case ID	IoC-EXT-06	Component	IoC-Extraction
Description	Test Preflight checks		
Tested by	LOGSTAIL		
Associated Requirements	FR-B-1.29 - Data manipulation		
Pre-condition(s)	The CTI sources are mounted as volumes on the container		
Test steps			
1	Set the paths of the CTI sources		
2	The component ingests the CTI feeds.		
Input data	CTI sources paths		
Results	Validation that the content of CTI feeds is correct and that all of them are found by the component		
KPIs	Target → Validate that the CTI feeds are found and ingested		
Test Case Result	Pass		

Table 99. Test case ioC-EXT-07

Test Case ID	IoC-EXT-07	Component	IoC-Extraction
Description	Test complete alert processing		
Tested by	LOGSTAIL		
Associated Requirements	FR-A-1.26 - Actionable Threat Intelligence FR-A-1.32 - Adaptive parsing FR-B-1.29 - Data manipulation		
Pre-condition(s)	The component has completed the Preflight checks, has established connection with Kafka and can interact with the topics, and also the MISP is up and running.		
Test steps			
1	Mock an alert		
2	Call the main processing function of the component		
Input data	Mocked Alert in JSON format		

Results	Validation that the IoCs are correct the enrichments and the contextualization is correct, that the IoCs are uploaded to MISP and that the final output is uploaded to kafka as a JSON and STIX object
KPIs	Target → Validate that the IoCs are extracted the enrichment is correct based on the IoCs the processing is shared on MISP and on the appropriate kafka topics
Test Case Result	Pass

Table 100. Test case ioC-EXT-08

Test Case ID	IoC-EXT-08	Component	IoC-Extraction
Description	Test STIX ingestion and formatting		
Tested by	LOGSTAIL		
Associated Requirements	FR-A-1.32 - Adaptive parsing		
Pre-condition(s)	The component has completed the Preflight checks, has established connection with Kafka and can interact with the topics, and also the MISP is up and running.		
Test steps			
1	Mock an alert that is in STIX format		
2	Call the main STIX processing function of the component		
3	Assert that the IoCs are extracted correctly		
4	Mock IoCs		
5	Create a STIX object from the mocked IoCs		
Input data	Mocked Alert in JSON format, mocked IoCs in JSON format		
Results	Validation that the IoCs extracted by the STIX object are correct Validation that the STIX object that is created is correct		
KPIs	Target → Validate that the IoCs and STIX object describe and wrap the ingested STIX alert correctly.		
Test Case Result	Pass		

3.15.1.5. Next steps and future development updates

- Optimise the analysis of the IoCs of the alerts from the detectors to make it more efficient.
- Reduce false positives in the enrichment phase.
- Improve overall system performance and reliability.
- Improve the output schema to achieve better readability.

3.15.2. Taranis-AI

3.15.2.1. Prototype Description

Taranis AI is an open-source Open-Source Intelligence (OSINT) platform designed to help organisations collect, process, analyse, and publish intelligence from publicly available sources. It combines data ingestion with automated enrichment, structured reporting, and flexible output formats to support evidence-based decision-making and reporting workflows.

Taranis AI is implemented as a containerised, service-oriented application that separates user interaction, control logic, and computational workloads. The architecture (see Figure a) is designed to support scalable OSINT collection and processing, asynchronous execution of enrichment tasks, and multi-user operation in a controlled environment. The system is composed of a core backend service, a web frontend, multiple worker services, and supporting infrastructure components for messaging, persistence, and real-time updates.

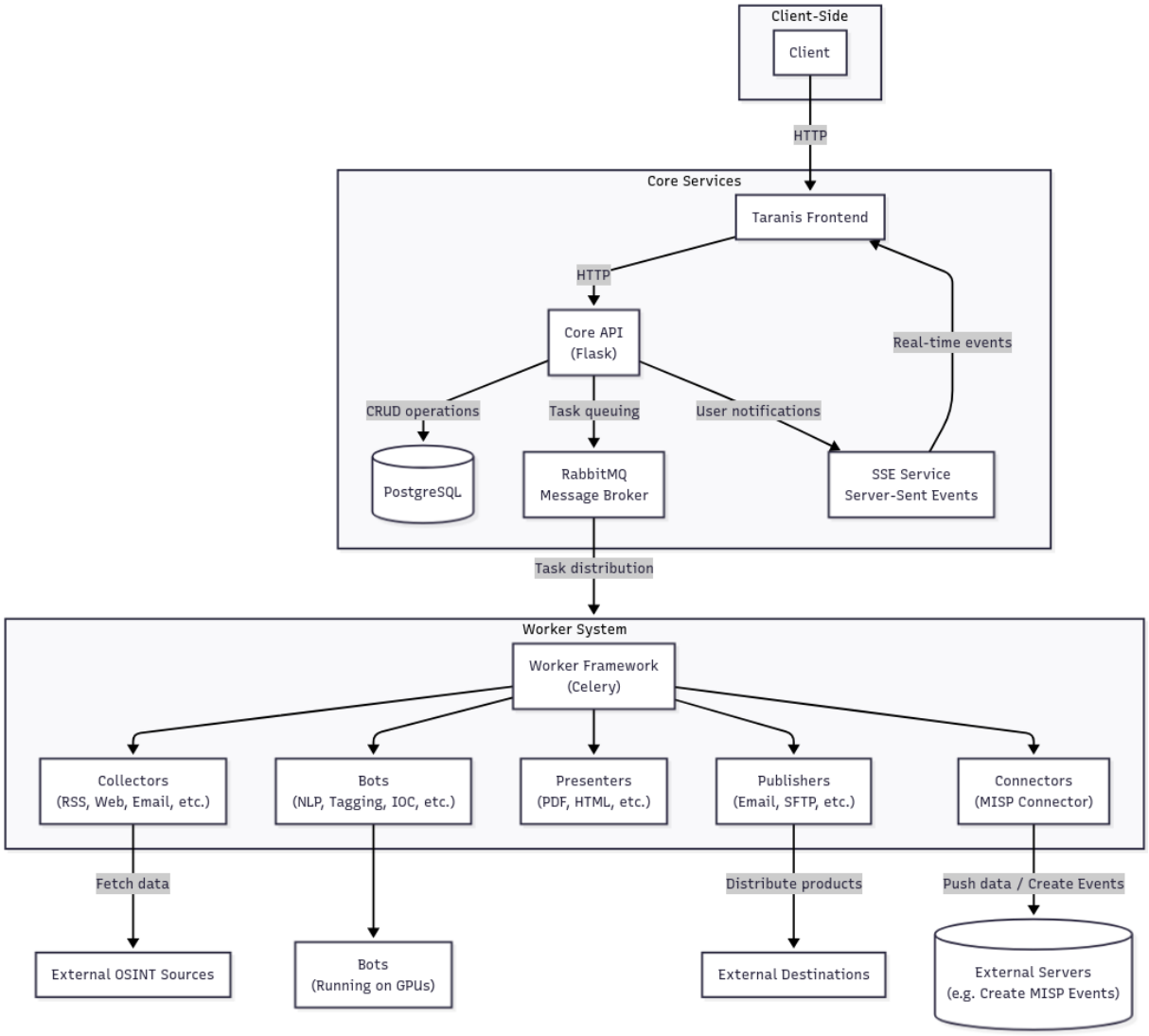

Figure 22. Architecture of Taranis AI.

Collection & Intake

Taranis AI supports the definition and management of multiple OSINT sources for automated collection. Users can create, import, export, and schedule sources, specifying the appropriate collector type and feed location for each. Collected data is stored as news items ready for review and processing.

Automated Processing & Enrichment

Collected items are enriched using a configurable set of bots. These include:

1. Wordlist Bot – Tags items according to curated word lists
2. IOC Bot – Extracts indicators of compromise
3. NLP Tagging Bot – Adds entity and semantic tags via natural language processing
4. Story Bot – Clusters related items into cohesive stories
5. Summary Bot – Generates summaries of story content

Bots can be reordered and activated depending on workflow needs.

Assess & Analyse

Analysts use the Assess interface to filter, search, and evaluate news items and stories. Items can be curated, merged, flagged, and added to structured reports. In the Analyze section, report items are created, edited, and organised according to defined report types. 4. Reporting & Publishing

Reporting & Publishing

Taranis AI supports the creation of output products from structured reports. Products can be published in multiple formats including HTML, JSON, PDF, and plain text. This allows automated, flexible dissemination of intelligence products.

Administration & Customisation

- The platform includes admin-level controls for:
- User, organisation, and role management
- Configuration of report types and attribute groups
- OSINT source, bot, and workflow settings
- Access to the OpenAPI interface for integrations
- These features support secure, multi-user deployments.

Requirements:

- FR-A-1.26 – Actionable Threat Intelligence: The system must generate threat intelligence in an actionable form, namely distilled, contextual and real-time data.
- FR-B-1.4 – Interface to external parties: The system must support federation with external providers.
- FR-B-1.27 – Data collection: The system must collect security related data from both security functions and digital services.
- SR-SPL-07 - Identity management for access to security functions: The system may integrate the authentication mechanisms with existing backends (LDAP, RADIUS) and SSO frameworks
- NON-FR-B-1.44 – Time to deploy containers: The time to deploy containers must not create a significant overhead for the Kubernetes orchestrator.

3.15.2.2. Communication interfaces

Some of the communication interfaces to control collection of sources, data enrichment or check API health are described below:

Table 101. Taranis AI communications interfaces

#	Method	REST Endpoint	Description	Request Body	Response Body
POST	/api/config/osint-sources/collect	Schedule (as soon as possible) the collection of all OSINT sources present.	{ “Authorization”: Bearer <valid-bearer> }	{ "message": "Refresh for source 10 scheduled" }	POST
POST	/api/config/bots/<string:bot_id>/execute	Schedule (as soon as possible) the bot with the ID provided	{ “Authorization”: Bearer <valid-bearer> }	{ "id": "<bot-id>", "message": "Executing Bot <bot-id> scheduled" }	POST
GET	/api/worker/storie	Get stories that are collected	{ “Authorization”: Bearer <valid-api-key> }	{ "attributes": {}, "comments": "", "created": "", "description": "", "dislikes": 0, "id": "", "important": false, "last_change": "", "likes": 0,	GET

				"news_items": [
				{
				"author": "",
				"collected": "",
				"content": "",
				"hash": "",
				"id": "",
				"language": "",
				"last_change": "",
				"link": "",
				"osint_source_id”: "",
				"published": "",
				"review": "",
				“source": "",
				"story_id": "",
				"title": "",
				"updated": ""
				}
				],
				"read": false,
				
				"relevance": 0,
				
				"search_vector": "",
				
				"summary": "",
				"tags": {},
				"title": "",	

				"updated": ""
				}{
				"attributes": {},
				"comments": "",
				"created": "",
				"description": "",
				"dislikes": 0,
				"id": "",
				"important": false,
				"last_change": "",
				"likes": 0,
				"news_items": [
				{
				"author": "",
				"collected": "",
				"content": "",
				"hash": "",
				"id": "",
				"language": "",
				"last_change": "",
				"link": "",
				"osint_source_id": "",
				"published": "",
				"review": "",
				"source": "",	

				"story_id": "", "title": "", "updated": "" }], "read": false, "relevance": 0, "search_vector": "", "summary": "", "tags": {}, "title": "", "updated": "" }	
POST	/api/config/osint-sources/collect	Schedule (as soon as possible) the collection of all OSINT sources present.	{ “Authorization”: Bearer <valid-bearer> }	{ "message": "Refresh for source 10 scheduled" }	POST

Table 102. Taranis AI communications interfaces (MQTT)

Topic	Description	Message Sample
MISP Connector Output	Output format of a story shared to MISP using MISP Connector (Output: MISP Event)	{ "Event": { <event_attributes> "Object": { "name": "taranis-news-item", ... }, { "name": "taranis-story", ... },], "EventReport": [{ "id": "155",

		"uuid": "<uuid>", "event_id": "172", "name": "<title>", "content": "# Story description\n\n\n# Story comment", "distribution": "5", "sharing_group_id": "0", "timestamp": "1769173307", "deleted": False, }], "CryptographicKey": [], } }

3.15.2.3. Deployment details

Taranis AI provides deployment instructions using Docker Compose and Kubernetes. Configuration is managed via environment variables and env files, and recommended system specifications scale with active enrichment workflows.

The deployment is also described here: https://github.com/taranis-ai/taranis-ai/blob/30b668ad3e01829b674bab4b95956fbdd9847c10/docker/README.md

Step by step guide:

- Clone repository with

```
git clone --depth 1  https://github.com/taranis-ai/taranis-ai
```

- Go to docker directory

```
cd taranis-ai/docker/
```

- Copy env.sample to .env (see also Figure c showing the default confuguration parameters)

```
cp env.sample .env
```

- Start the application with

```
docker compose up -d
```

Dockerfile:

```
COMPOSE_PROJECT_NAME=taranis
TARANIS_TAG=stable
DOCKER_IMAGE_NAMESPACE=ghcr.io/taranis-ai

TARANIS_PORT=8080
GRANIAN_PORT=8080

# Default passwords. CHANGE THESE FOR PRODUCTION!
POSTGRES_PASSWORD=supersecret
JWT_SECRET_KEY=supersecret
API_KEY=supersecret
```

```
BOT_API_KEY=supersecret
RABBITMQ_PASSWORD=supersecret
```

The necessary images can also be built from source if necessary with:

```
cd Taranis AI
docker build -t taranis-core . -f ./docker/Containerfile.core
docker build -t taranis-ingress . -f ./docker/Containerfile.ingress
docker build -t taranis-worker . -f ./docker/Containerfile.worker
docker build -t taranis-frontend . -f ./docker/Containerfile.frontend
docker build -t taranis-frontend . -f ./docker/Containerfile.rabbitmq
```

There are several Containerfiles and each of them builds a different component of the system. These Containerfiles exist (see code listing below):

- Containerfile.worker
- Containerfile.core
- Containerfile.ingress
- Containerfile.frontend
- Containerfile.rabbitmq

Containerfile.core:

```
FROM python:3.13-slim AS builder

COPY --from=ghcr.io/astral-sh/uv:latest /uv /bin/uv
WORKDIR /app/

RUN apt-get update && apt-get install --no-install-recommends -y \
    libpq-dev \
    git \
    curl \
    openssl \
    build-essential \
    python3-dev

COPY ./src/core/. /app/
COPY ./.git /.git

ENV UV_COMPILE_BYTECODE=1

RUN uv venv && \
    export PATH="/app/.venv/bin:$PATH" && \
    uv sync --frozen

FROM python:3.13-slim

LABEL description="Taranis AI Python Flask JSON RPC API"
WORKDIR /app/

RUN groupadd user && useradd --home-dir /app -g user user && chown -R
user:user /app
RUN apt-get update && apt-get install --no-install-recommends -y \
    libpq-dev \
    curl \
    openssl
RUN install -d -o user -g user /app/data

COPY --from=builder --chown=user:user /app/.venv /app/.venv
COPY --chown=user:user ./src/core/. /app/
```

```
USER user

ENV PYTHONOPTIMIZE=1
ENV PATH="/app/.venv/bin:$PATH"
ENV PYTHONPATH=/app
ARG git_info
ENV GIT_INFO=${git_info:-'{}'}
RUN echo BUILD_DATE=$(date --iso-8601=minutes) > .env
ENV DATA_FOLDER=/app/data
ENV GRANIAN_THREADS=2
ENV GRANIAN_WORKERS=2
ENV GRANIAN_BLOCKING_THREADS=4
ENV GRANIAN_HOST=0.0.0.0
ENV GRANIAN_PORT=8080


VOLUME ["/app/data"]
EXPOSE 8080

CMD ["taranis-ai"]
```

Containerfile.frontend

```
FROM python:3.13-slim AS builder

COPY --from=ghcr.io/astral-sh/uv:latest /uv /bin/uv
COPY --from=denoland/deno:bin-2.5.6 /deno /usr/local/bin/deno

WORKDIR /app/

RUN apt-get update && apt-get install --no-install-recommends -y \
    git \
    curl \
    7zip \
    openssl \
    build-essential \
    python3-dev

COPY ./src/frontend/. /app/
RUN rm -f /app/frontend/static/assets/openapi3_1.yaml
COPY                                    ./src/core/core/static/openapi3_1.yaml
/app/frontend/static/assets/openapi3_1.yaml

ENV UV_COMPILE_BYTECODE=1

RUN /app/build_tailwindcss.sh && \
    uv venv && \
    export PATH="/app/.venv/bin:$PATH" && \
    uv sync --frozen

FROM python:3.13-slim

LABEL description="Taranis AI Python Flask HTMX Frontend"
WORKDIR /app/

RUN groupadd user && useradd --home-dir /app -g user user && chown -R
user:user /app
RUN install -d -o user -g user /app/data

COPY --from=builder --chown=user:user /app/.venv /app/.venv
```

```
COPY --chown=user:user ./src/frontend/. /app/
COPY      --from=builder      --chown=user:user      /app/frontend/static/
/app/frontend/static/

USER user

ENV PYTHONOPTIMIZE=1
ENV PATH="/app/.venv/bin:$PATH"
ENV PYTHONPATH=/app
ARG git_info
ENV GIT_INFO=${git_info:-'{}'}
RUN echo BUILD_DATE=$(date --iso-8601=minutes) > .env
ENV DATA_FOLDER=/app/data
ENV GRANIAN_THREADS=2
ENV GRANIAN_WORKERS=2
ENV GRANIAN_INTERFACE=wsgi
ENV GRANIAN_BLOCKING_THREADS=4
ENV GRANIAN_HOST=0.0.0.0
ENV GRANIAN_PORT=8080


VOLUME ["/app/data"]
EXPOSE 8080

CMD ["granian", "app"]
```

Containerfile.worker

```
FROM python:3.13-slim AS builder

COPY --from=ghcr.io/astral-sh/uv:latest /uv /bin/uv
WORKDIR /app/

RUN apt-get update && apt-get upgrade -y && \
    apt-get install --no-install-recommends -y \
      build-essential \
      python3-dev \
      libglib2.0-0 \
      libpango-1.0-0 \
      libpangoft2-1.0-0 \
      git

COPY ./src/worker/. /app/
COPY ./.git /.git

ENV UV_COMPILE_BYTECODE=1

RUN uv venv && \
    export PATH="/app/.venv/bin:$PATH" && \
    uv sync --frozen

FROM python:3.13-slim

LABEL description="Taranis AI Python Celery Worker"

WORKDIR /app

ENV PYTHONOPTIMIZE=1
ENV PYTHONPATH=/app
ENV PATH="/app/.venv/bin:$PATH"
COPY --from=builder /bin/uv /bin/uv
```

```
RUN apt-get update && \
    apt-get install --no-install-recommends -y \
      libpango-1.0-0 \
      libpangoft2-1.0-0 && \
      uv tool run playwright install-deps chromium

RUN groupadd user && useradd --home-dir /app -g user user && chown -R
user:user /app
USER user
COPY --from=builder --chown=user:user /app/ /app/
RUN playwright install --only-shell chromium

ARG git_info
ENV GIT_INFO=${git_info:-'{}'}
RUN echo BUILD_DATE=$(date --iso-8601=minutes) > .env

ENTRYPOINT [ "celery" ]

CMD ["--app", "worker", "worker"]
```

Containerfile.ingress

```
FROM nginxinc/nginx-unprivileged:mainline

ENV TARANIS_CORE_UPSTREAM=core:8080
ENV TARANIS_FRONTEND_UPSTREAM=frontend:8080
ENV TARANIS_SSE_UPSTREAM=sse:8088
ENV TARANIS_BASE_PATH=/

COPY                              ./src/ingress/extras/default.conf.template
/etc/nginx/default.conf.template
COPY    ./src/ingress/extras/10-base_uri_envsubst_entrypoint.sh    /docker-
entrypoint.d/
```

Containerfile.rabbitmq

```
FROM docker.io/library/rabbitmq:4-management-alpine

RUN echo "consumer_timeout = 36000000" >> /etc/rabbitmq/rabbitmq.conf
```

Taranis AI is distributed as a set of container images. Each image is versioned using image tags, which define which application version is deployed. Selecting the appropriate image tag is important for stability, reproducibility, and update strategy.

Taranis AI images are published with the following tag types:

- Versioned tags
  - Versioned tags (e.g. 1.1.7) refer to a specific, immutable release of the software.
  - They provide full reproducibility and are recommended for production deployments where controlled upgrades are required.
- stable tag
  - The stable tag points to the latest tested and officially supported release.
  - It is updated only when a new stable version is published. This tag is suitable for users who want up-to-date features while maintaining a reasonable level of stability.
- latest tag

- The latest tag tracks the most recent build. It may include new features or changes that have not yet undergone full validation.
- This tag is intended for development, testing, or evaluation environments and is not recommended for production use.

3.15.2.4. Individual component testing

Table 103. Taranis AI Component test case

Component: Test Cases		
TC-Taranis-01	Test RSS Collector correctness	Achieved
TC-Taranis-02	Test Simple Web Collector correctness	Achieved
TC-Taranis-03	Test MISP Collector correctness	Achieved
TC-Taranis-04	Test Assess correctness	Achieved
TC-Taranis-05	Test Taranis AI Analyze	Achieved

Table 104. Test case Taranis-01

Test Case ID	TC-Taranis-01	Component	RSS Collector
Description	This test case verifies the correct functionality of the RSS Collector component, including feed retrieval, conditional requests, HTTP header configuration, digest splitting, and wordlist-based filtering.		
Tested by	AIT		
Associated Requirements	FR-A-1.26, FR-B-1.4, FR-B-1.27		
Pre-condition(s)	RSS Collector component is available. Mocked RSS feeds and HTTP responses are configured.		
Test steps			
1	Configure an RSS source with valid parameters		
2	Trigger RSS collection		
3	Verify handling of normal, empty, and unchanged feeds		
4	Verify application of additional HTTP headers and digest splitting		
5	Apply include and exclude wordlists and validate filtering results		
Input data	RSS feed URLs Collector parameters (configuration, headers, digest splitting flags, wordlists) Data format: structured configuration objects and lists of news items		
Results	RSS feeds are collected correctly. Conditional requests and filtering behave as expected.		

KPIs	Target → 100% correct processing of supported RSS scenarios Measured → 100%
Test Case Result	Pass

Table 105. Test case Taranis-02

Test Case ID	TC-Taranis-02	Component	Simple Web Collector
Description	This test case verifies the correct functionality of the Simple Web Collector component, including web content retrieval, extraction, XPath filtering, digest splitting, and HTTP header configuration.		
Tested by	AIT		
Associated Requirements	FR-A-1.26, **FR-B-1.4, FR-B-1.27**		
Pre-condition(s)	Simple Web Collector component is available Mocked web pages and HTTP responses are configured		
Test steps			
1	Configure a web source with a valid URL		
2	Trigger web collection		
3	Verify correct extraction of content and metadata		
4	Apply XPath expressions and digest splitting		
5	Verify application of additional HTTP headers		
Input data	Web page URLs XPath expressions Collector parameters (headers, digest splitting)		
Results	Web content is retrieved and parsed correctly XPath filtering and digest splitting behave as configured		
KPIs	Target → 100% correct extraction and processing Measured → 100%		
Test Case Result	Pass		

Table 106. Test case Taranis-03

Test Case ID	TC-Taranis-03	Component	MISP Collector
Description	This test case verifies the correct functionality of the MISP Collector component, including parameter validation, authentication, data retrieval, and error handling.		
Tested by	AIT		
Associated Requirements	FR-A-1.26, **FR-B-1.4, FR-B-1.27**		

Pre-condition(s)	MISP Collector component is available Mocked MISP API responses are configured
Test steps	
1	Configure a MISP source with required parameters
2	Trigger MISP collection
3	Verify successful data retrieval for valid responses
4	Verify behaviour for empty or missing data
5	Verify handling of malformed responses and invalid parameters
Input data	MISP API URL API key
Results	Valid MISP data is collected Error conditions are detected and handled correctly
KPIs	Target → 100% correct handling of supported MISP scenarios Measured → 100%
Test Case Result	Pass

Table 107. Test case Taranis-04

Test Case ID	TC-Taranis-04	Component	Taranis AI Assess
Description	Verify correct operation of story-related functions in the Assess API, including retrieval, grouping, ungrouping, tagging, and correct handling of story and news-item state.		
Tested by	AIT		
Associated Requirements	FR-A-1.26, **FR-B-1.4**		
Pre-condition(s)	Assess API running Authenticated user available Test stories and news items exist		
Test steps			
1	Retrieve stories and individual story details via Assess API		
2	Verify persistence and attributes of worker-created stories		
3	Group stories and validate merged content and state		
4	Ungroup stories and news items and validate resulting stories		
5	Set and retrieve story tags and validate state consistency		
Input data	Story and news item UUIDs Tag payloads (valid and invalid)		
Results	Story operations return expected responses		

	Grouping and ungrouping modify stories correctly Story and news-item states are consistent
KPIs	Target → 100% correct story operations Measured → 100%
Test Case Result	Pass

Table 108. Test case Taranis-05

Test Case ID	TC-Taranis-05	Component	Taranis AI Analyze
Description	Verify correct operation of the Analyze API for report items, including create, list, update, clone, manage associated stories, and delete.		
Tested by	AIT		
Associated Requirements	FR-A-1.26, **FR-B-1.4**		
Pre-condition(s)	Analyze API running Authenticated user available		
Test steps			
1	Create a report		
2	List report items and verify the count of reports		
3	Update the report item		
4	Clone the report item		
5	Manage report stories		
Input data	Report item JSON Update payload (JSON) Stories payload (list of UUIDs)		
Results	Report item CRUD operations return expected responses. Cloned report has a different ID than the original. Story association endpoints respond successfully and reflect updates. Deletion returns success message indicating the deleted report title.		
KPIs	Target → 100% correct Analyze API report-item operations Measured → 100%		
Test Case Result	Pass		

3.15.2.5. Next steps and future development updates

- MISP integration will be enhanced to better support CTI analysts' workflows and to improve collaborative capabilities. Conflict resolution mechanisms and handling of shared information will be further improved.

- **Application caching** will be improved to enhance overall usability and performance.
- **Version history** for reports and stories will be introduced.
- **Audit logging** will be added to improve traceability.
- In addition, further AI-driven features are planned, such as on-demand summarisation of reports.

3.15.3. CTI Integration

3.15.3.1. Prototype Description

OpenCTI and MISP are open-source platforms used for cyber threat intelligence management and sharing.

OpenCTI is designed to structure, enrich, and analyse threat intelligence data. It supports the modelling of threat actors, campaigns, malware, and indicators using the STIX standard. This approach provides a structured representation of threats and their relationships, supporting analysis and decision-making at strategic and operational levels.

MISP is designed for the collection and exchange of indicators of compromise. It enables the sharing of IP addresses, domains, file hashes, and attack patterns among trusted communities. The platform is widely adopted by CERTs, SOCs, and public institutions for operational threat information sharing.

Used together, the two platforms support complementary objectives. MISP enables rapid dissemination of actionable indicators, while OpenCTI provides context, correlation, and long-term analysis. This combination supports improved situational awareness and more effective incident response.

The architecture combines OWASP, OpenCTI, and MISP into a single, consistent intelligence pipeline. External knowledge from OWASP is collected through a scheduled monitoring component. This component retrieves the latest security risks, vulnerability classifications, and trends and pushes them into OpenCTI as structured entities. This provides a stable knowledge base aligned with widely accepted application security standards.

In parallel, external threat intelligence feeds are ingested directly into MISP. These feeds provide high volume operational indicators and are processed according to local configuration, including tagging, filtering, and sharing rules. MISP acts as the primary intake and dissemination platform for operational data.

Data flows bidirectionally between MISP and OpenCTI through dedicated connectors. Indicators and events ingested into MISP from feeds are transferred to OpenCTI for correlation, enrichment, and analysis.

This unified structure ensures traceability from source to dissemination. It separates rapid sharing from analytical processing while maintaining consistency across platforms. The result is improved situational awareness, reduced duplication of effort, and more effective threat intelligence sharing across stakeholders.

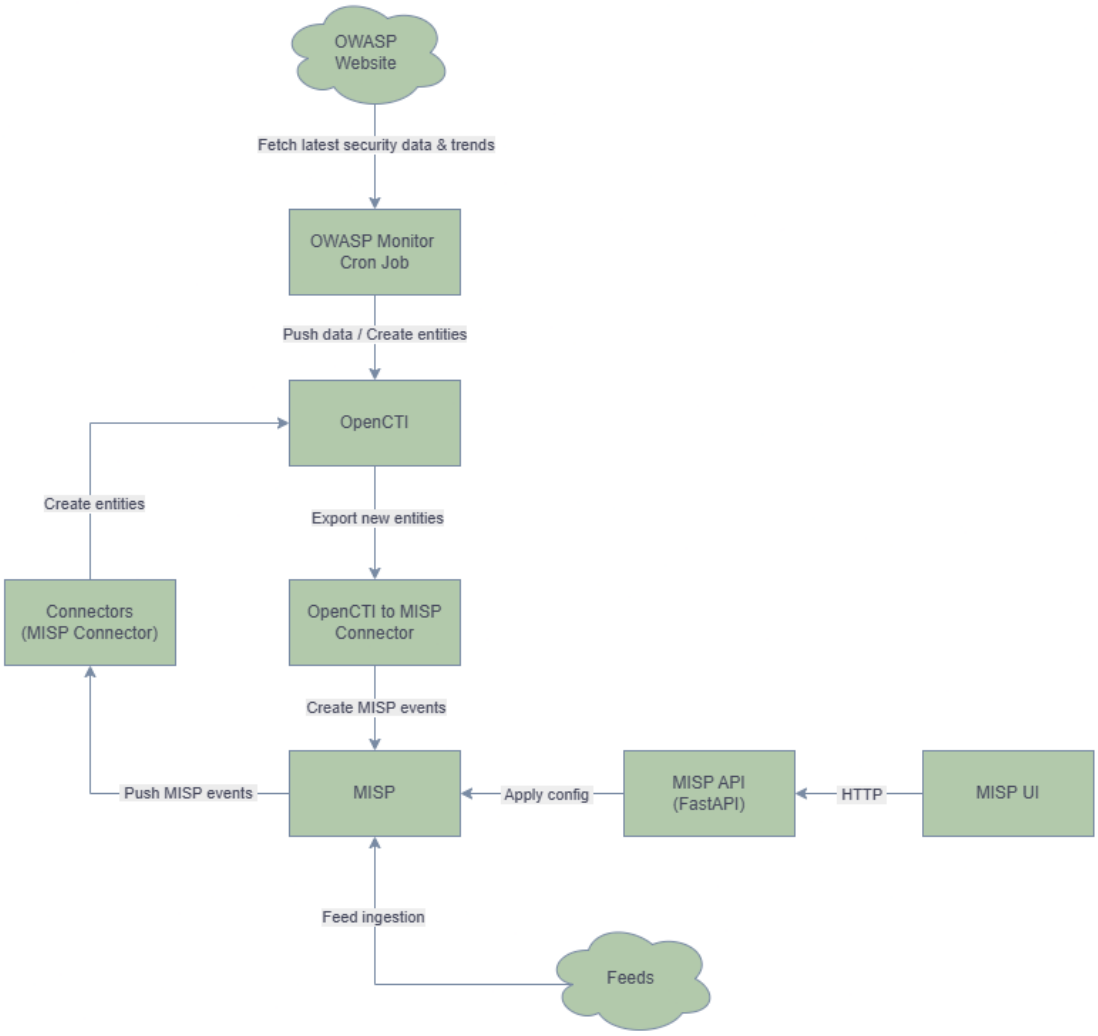

Figure 23. Architecture of CTI Integration

Requirements

- FR-A-1.26 – Actionable Threat Intelligence: The system must generate threat intelligence in an actionable form, namely distilled, contextual and real-time data.
- FR-B-1.4 – Interface to external parties: The system must support federation with external providers.
- FR-B-1.25 - Commands and controls to security functions: The system should support at least the following commands: GET capabilities; GET current configuration; START operation; STOP operation; POST updated configuration; PUT new configuration
- FR-B-1.38 - Execution environment for MIRANDA Connector and Controller: The system must provide Python >= 3.10
- NON-FR-B-1.44 – Time to deploy containers: The time to deploy containers must not create a significant overhead for the Kubernetes orchestrator.
- FR-B-1.46 - System context and data export/import: The system must be able to export/import its internal context and data to/from third parties.

3.15.3.2. Communication Interfaces

This following table documents the REST API endpoints available for MISP configuration and event management, including creation, upload, and download of events.

Table 109. CTI Communication Interfaces

Method	REST Endpoint	Description	Request Body	Response Body
GET	/feeds	List all feeds (wrapper for misp.direct_call("feeds/index")).	None	[{...}] list of feeds (raw MISP output).
POST	/feeds	Create a new feed (wrapper for feeds/add).	JSON (example): { "name": "My Feed", "url": "https://example.com/feed.json", "provider": "ACME", "distribution": 0, "enabled": false }	JSON with created feed info. On error: HTTP 400 with { "detail": "..." }.
POST	/feeds/{feed_id}/enable	Enable a feed by ID (wrapper for feeds/enable/{id}).	None	JSON (example): { "name": "Feed enabled.", "message": "Feed enabled.", "url": "/feeds/enable/2"}
POST	/feeds/{feed_id}/disable	Disable a feed by ID (wrapper for feeds/disable/{id}).	None	JSON (example): { "name": "Feed disabled.", "message": "Feed disabled.", "url": "/feeds/disable/2"}
POST	/feeds/{feed_id}/fetch	Trigger ingestion from a feed (wrapper for feeds/fetchFromFeed/{id}).	None	JSON (example): { "message": "Pull initiated", "note": "Feed fetch is running in the background. You can check job status with /jobs or view events later." } On error: HTTP 400 with { "detail": "..." }.
GET	/events	List events with pagination and optional filters (wrapper for events/index).	None	JSON list from MISP.

GET	/events/{event_id}	Get one event by ID (wrapper for events/view/{id}).	None	JSON event payload (raw MISP response).
POST	/events	Create a new event (wrapper for events/add).	JSON (example): { "info": "Test event", "date": "2026-01-28", "threat_level_id": 2, "analysis": 0, "distribution": 0, "tags": ["tlp:white"] }	JSON with created event. On error: HTTP 400 with { "detail": "..." }.
POST	/events/upload-json	Upload a JSON file and create an event from it.	file (.json)	JSON (example): { "message": "Event created", "event": {...} } On error: HTTP 400/500 with { "detail": "..." }.
GET	/events/{event_id}/export?format=json|stix|stix2	Export one event as JSON or STIX via MISP restSearch (for STIX). Returns a downloadable file.	None	File download stream. For json: application/json with event_{id}.json. For stix2: JSON with event_{id}.stix.json. For stix (1.x): code still returns JSON media type, but filename ends with .stix.xml.
POST	/events/import/stix	Import STIX file into MISP via upload_stix.	file (.json)	Success JSON (example): { "status":"success", "id":"123", "info":"...", "org":"...", "date":"2026-01-28" } Error JSON (example): { "status":"error_name", "message":"..." }

3.15.3.3. Deployment details

Kubernetes is a container orchestration platform used to deploy, operate, and maintain complex applications. In this setup, the platform is defined through a structured set of manifest files organised under the miranda_kube_manifests/ directory. Each subdirectory represents a logical subsystem, such as misp/, misp_api/, opencti/, and the integration components between them. Configuration, storage, networking, and workloads are defined as separate Kubernetes resources, following established cloud-native design principles.

Two core components, MISP and OpenCTI, are already published and publicly available as container images. These images are maintained by their respective projects and can be pulled directly from public container registries, without requiring custom image builds. In addition,

three project-specific components were developed: misp-api, owasp-to-opencti, and opencti-to-misp. The misp-api service manages API interactions used to configure MISP, including feeds, events, and operational settings. The owasp-to-opencti component runs as a background connector that monitors the OWASP website and detects updates in published reports, creating corresponding entities in OpenCTI. The opencti-to-misp connector transfers selected entities from OpenCTI back into MISP, including entities originating from OWASP.

All custom components are delivered as Docker containers, with Dockerfiles and docker-compose definitions used during development. To make these services available in Kubernetes, container images must first be built and pushed to a container registry. Kubernetes pulls these images from the registry during pod creation, ensuring consistent deployment across environments and explicit version control. The following commands describe the standard image build and publication process used for the custom components:

1. docker build -t owasp-to-opencti:v1.0 .
2. sudo docker login https://registry
3. sudo docker tag owasp-to-opencti:v1.0 registry/{name_of_registry}/owasp-to-opencti:v1.0
4. sudo docker push registry/{name_of_registry}/owasp-to-opencti:v1.0

The same workflow is applied for misp-api and opencti-to-misp, using the corresponding image names and versions. The resulting image references are then used directly in the Kubernetes manifests to ensure that the tested image versions are deployed.

The Kubernetes deployment follows a strict and predefined order. First, ConfigMaps and Secrets are applied, including misp-configMap.yml, misp-api-configMap.yml, opencti-configMap.yml, opencti-secrets.yml, miranda-config.yml, and miranda-secrets.yml. These resources define configuration values, credentials, and runtime parameters required at container startup. Next, volume definitions are applied, such as misp-volumes.yml and opencti-to-misp-volumes.yml, ensuring persistent storage is available before any stateful workload starts.

Once configuration and storage are in place, Services are deployed using files such as misp-service.yml, misp-mysql-service.yml, misp-redis-service.yml, opencti-service.yml, elasticsearch-service.yml, and the related connector and worker service files. These Services provide stable internal endpoints and DNS names for reliable inter-component communication. Finally, StatefulSets are applied, including misp-stateful.yml, misp-mysql-stateful.yml, misp-redis-stateful.yml, opencti-stateful.yml, elasticsearch-stateful.yml, rabbitmq-stateful.yml, and the connector StatefulSets. This deployment sequence ensures deterministic startup, persistent data handling, and reliable operation across all components.

3.15.3.4. Next steps and future development updates

- Audit logging will be introduced to support traceability across the intelligence workflow.
- Improve overall system efficiency and response times.

4. CONCLUSIONS

This deliverable confirms that MIRANDA has evolved from a set of independently engineered prototypes into an increasingly cohesive, integrated cyber-intelligence platform that can be deployed and exercised as a unified system. The integration work is grounded in a container-first approach and a shared DevSecOps/CI-CD foundation, enabling repeatable builds, consistent runtime environments, and systematic quality and security checks throughout development and testing. This is a critical step required to move from research artefacts toward operationally usable capabilities.

The common CI/CD toolchain and standardized containerised deployment model establish a controlled path from source code to validated, deployable artefacts. Automated pipeline stages covering build, testing, and security checks create consistent quality gates and shorten feedback loops when defects or vulnerabilities are introduced. Containerisation and Kubernetes-based orchestration provide reproducibility across environments and simplify multi-component integration, while shared development and testing infrastructure and controlled access policies support integration under comparable conditions and with consistent operational assumptions.

Importantly, the platform's integration approach supports incremental delivery and that is the individual components can be improved, re-released, and re-deployed without forcing a platform-wide "big bang" update. This is essential for a complex consortium environment where different components naturally mature at different speeds and where integration must remain continuous, measurable, and safe.

The integrated platform spans the major stages of a cyber-defence workflow: collecting and normalising data, building and updating a representation of system context, enriching observations with cyber threat intelligence, applying detection and predictive analytics, supporting analyst understanding through visualization, and preparing the ground for automated response and enforcement. Data handling provides the structured representations required by downstream analytics by parsing semi-structured logs into structured events and fields suitable for downstream processing.

The platform's design prioritises modularity and decoupling because the components communicate through stable interfaces (often REST-based where appropriate) and through shared messaging and storage mechanisms (e.g., Kafka for event distribution and MongoDB where persistent storage and historical review are required). This design supports progressive integration and enables targeted scaling of high-load functions without destabilising the entire platform.

A core platform achievement is maintaining a machine-readable understanding of the operational environment. Context Discovery constructs a representation of assets, services, and relationships that serves as a common "system picture" for multiple downstream modules. This context representation allows telemetry and alerts to be interpreted in relation to service chains and dependencies, enabling operators to understand not only what happened, but where it happened and why it matters.

Attack & Threat Modelling builds upon this context to associate vulnerabilities and threats with services and relationships and to support reasoning about attack conditions and potential

propagation. In combination, context and modelling shift the platform beyond isolated signal processing: they enable threat analysis that is grounded in the real structure of the system and therefore more actionable for both analysts and automation.

MIRANDA integrates multiple intelligence sources to extend situational awareness beyond local telemetry. The Threat Feed Connector provides a modular, REST-based mechanism for retrieving and normalising external threat feeds and exposing them through a consistent API for consumption by other platform components. The connector's modular architecture and deployment flexibility (local, Docker, Kubernetes) create a practical pathway for expanding supported sources and scaling the service as intelligence volumes grow and new sources are added.

Actionable CTI capabilities deepen the intelligence layer. Indicators of Compromise (IoC) extraction structures indicators into standard representations suitable for correlation and sharing and is designed to integrate with external CTI sources and with a threat sharing interface. In addition, the OpenCTI–MISP integration provides a consistent intelligence pipeline in which vulnerability knowledge and indicators can be collected, exchanged, correlated, enriched, and analysed. This integrated approach supports traceability, correlation, and more effective intelligence sharing across stakeholders and tools.

OSINT enrichment complements CTI flows. Taranis-AI contributes a containerised, service-oriented pipeline that collects open-source information and enriches it through configurable bots (e.g., term tagging, indicator extraction, NLP-based tagging, clustering, and summarisation). This capability supports evidence-based reporting workflows and helps analysts manage high-volume information streams by converting raw items into structured, searchable, and summarised content.

Automatic Detection (including DetectMate) operationalises anomaly detection over parsed logs and publishes suspicious observations for downstream consumption. The detection approach supports configuration through APIs and is designed to integrate with the platform's message bus, providing a scalable pattern for continuous monitoring. The roadmap to add more detectors reflects an important direction: as the platform is exercised across varied environments, detection capabilities must broaden to cover different log schemas and operational environments.

Beyond detection, MIRANDA introduces capabilities for hypothesis-driven analysis and proactive reasoning. The Prediction component supports generating synthetic traffic and predicting traffic under previously unobserved conditions, including generation of synthetic attack traffic based on historical benign data. This capability can be leveraged to augment training and evaluation datasets, to stress-test analytics under controlled conditions, and to explore "what-if" scenarios in support of defensive planning and controlled evaluation.

Threat Hunting strengthens the proactive posture through Graph Neural Network techniques aimed at predicting multi-step attack progression. By combining contextual inputs and modelling knowledge with observed signals, it seeks to anticipate attacker moves and enable pre-emptive action. Although training workflows may remain computationally intensive and occur offline, the presence of defined inference workflows and publication mechanisms provides a clear path to deeper integration as the component matures.

The integrated platform treats assurance and trust as platform primitives. The Trusted Computing Base provides cryptographic and attestation-related capabilities and introduces

kernel-level monitoring support (including eBPF-based tracing) to underpin integrity claims and verification of critical attributes. Such capabilities are increasingly necessary for security platforms that must demonstrate not only detection performance but also the integrity of the data and processes that drive decisions.

The Trust Assessment Framework complements this with intra-domain and inter-domain trust models and decision logic. By aggregating evidence from multiple trust sources and evaluating assessed trust levels against required trust thresholds, it supports trust-aware sharing and cross-domain collaboration. The roadmap to expand trust sources, benchmark against use cases, explore federation, and evolve the decision logic toward uncertainty-aware reasoning provides a credible trajectory for improving robustness and real-world scenarios, including cross-domain use cases.

Usability and safety are addressed through visualization and controlled experimentation. GUI extensions provide interactive exploration of service context graphs and associated threat information while relying on internal platform interfaces, keeping access control centralised and minimising external exposure. Sandboxing enables isolated Kubernetes namespaces with network isolation policies and dedicated identities, providing a controlled environment to deploy workloads and execute security testing safely. These capabilities are important enablers for safe, iterative validation of response policies and enforcement strategies.

The deliverable provides a transparent picture of maturity across components. Several modules are already containerised with deployment guidance and defined test cases, while others remain in active development or early design phases. The AI-based response capability illustrates this: it clarifies a target detection–enrichment–response–enforcement pipeline and defines intended interfaces, but remains at an early readiness level and requires implementation, evaluation on MIRANDA data, containerisation, and Kubernetes deployment before it can be integrated safely and responsibly.

Across the platform, the most important remaining work is therefore not simply adding features, but closing integration gaps and hardening operational behaviour and there for we aligning schemas and interfaces, ensuring predictable performance under load, improving observability and traceability, and validating interoperability across components in realistic use-case settings.

The next integration cycles should focus on cross-cutting priorities that are repeatedly identified across component roadmaps:

- Hardening and robustness: improve overall system efficiency and response times; make parsing, enrichment, and analytics pipelines more resilient; reduce false positives where enrichment or detection over-triggers; and strengthen reliability for continuous operation.
- Completing integrations and interfaces: finalise and validate APIs; replace mocked inputs with live streams in visualization layers; and improve interoperability so component outputs are consumable without ad-hoc adaptation.
- Scaling and storage improvements: strengthen threat feed storage and caching (including planned moves away from file-backed approaches toward scalable datastores); validate orchestration behaviour at larger scale; and improve performance for highly dynamic contexts and analytics updates.

- Identity, access control, and cross-domain readiness: converge on consistent identity management and authorisation models (including RBAC/ABAC where planned) to support federation while maintaining centralised governance.
- Traceability and auditability: introduce audit logging and workflow traceability across key intelligence and enrichment chains to support accountability, governance, and forensic reconstruction.

A key strategic direction is moving from detection and enrichment toward controlled, policy-governed automated response. MIRANDA already contains critical building blocks for this trajectory—context awareness, threat reasoning, trust evaluation, actionable intelligence enrichment, and standard command/control patterns (including OpenC2 and connector mechanisms). Advancing toward closed-loop automation should be pursued as measurable integration increments: establish baseline response models for integration, evaluate performance on MIRANDA data, containerise and deploy within the common runtime environment, and validate behaviour through sandboxed and governed execution paths before enabling broader autonomy.

Overall, the integrated MIRANDA platform is positioned as a credible, standards-aware, and extensible cyber-intelligence environment. It provides integrated capabilities for context discovery, threat modelling, intelligence ingestion (CTI and OSINT), data handling, anomaly detection, prediction and proactive threat hunting, trust and assurance, visualization, and safe experimentation. With continued focus on hardening, interoperability, traceability, scalable operations, and governed automation, MIRANDA is well-placed to progress toward operational cyber-defence workflows that can be validated and adopted more broadly.

5. REFERENCES

[1] Open Command and Control (OpenC2) – Architecture Specification Version 1.0. OASIS Committee Specification 01, 30 September 2022. URL: https://docs.oasis-open.org/openc2/oc2arch/v1.0/cs01/oc2arch-v1.0-cs01.pdf.

[2] Open Command and Control (OpenC2) - Language Specification Version 1.0. OASIS, Sep. 2019. URL: https://docs.oasis-open.org/openc2/oc2ls/v1.0/cs02/oc2ls-v1.0-cs02.pdf.

[3] Specification for Transfer of OpenC2 Messages via HTTPS Version 1.1. OASIS Committee Specification 01, 30 November 2021. URL: https://docs.oasis-open.org/openc2/open-impl-https/v1.1/cs01/open-impl-https-v1.1-cs01.pdf.

[4] Specification for Transfer of OpenC2 Messages via MQTT Version 1.0. OASIS Committee Specification Draft 03, 17 February 2021. URL: https://docs.oasis-open.org/openc2/transf-mqtt/v1.0/cs01/transf-mqtt-v1.0-cs01.pdf.

[5] Open Command and Control (OpenC2) – Profile for Stateless Packet Filtering Version 1.0. OASIS Committee Specification 01, 11 July 2019. URL: https://docs.oasis-open.org/openc2/oc2slpf/v1.0/cs01/oc2slpf-v1.0-cs01.pdf.

[6] Silvio Tanzarella, Matteo Repetto. Context Discovery for Digital Service Chain with OpenC2. 7th International Workshop on Cyber-Security Threats, Trust and Privacy Management in Software-defined and Virtualized Infrastructures (SecSoft), June 27th, 2025, Budapest, Hungary, pp. 579-584. DOI: 10.1109/NetSoft64993.2025.11080629

[7] M. Repetto. Otupy: A flexible, portable, and extensible framework for remote control of security functions. Computers & Security, Volume 158, November 2025, Article no. 104597. DOI: 10.1016/j.cose.2025.104597

[8] Stefano Catenaro. Homogeneous control of stateless firewalls with OpenC2. Master's Degree Thesis, Politecnico di Torino, 2025. Available: https://webthesis.biblio.polito.it/38598/1/tesi.pdf.

[9]

[10]

This project has received funding from the European Union's Horizon Europe Research and Innovation programme under grant agreement No. 101168144. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Cybersecurity Competence Centre. Neither the European Union nor the granting authority can be held responsible for them.