Co-funded by the Horizon 2020
Framework Programme of the European Union

Practical Autonomous Cyberhealth for resilient SMEs & Microenterprises

Grant Agreement No. 883335
Innovation Action (IA)

D5.1 Hybrid Threat Intelligence Framework – First Release

Document Identification			
Status	Final	**Due Date**	31/01/2022
Version	1.0	**Submission Date**	24/01/2022

Related WP	WP5	**Document Reference**	1.0
Related Deliverable(s)	D2.1, D3.1, D4.1, D3.2, D4.3, D5.2	**Dissemination Level (*)**	PU
Lead Participant	NEC	**Lead Author**	INF
Contributors	INF, SPH, NEC, POLITO	**Reviewers**	SSE
			NCSRD

Keywords:
Threat Intelligence, Distributed Collection, Data Preprocessing, Multimodal Anomaly Detection, Threat Classification, Alarm Management, Recommendation, Remediation

This document is issued within the frame and for the purpose of the *PALANTIR* project. This project has received funding from the European Union's Horizon2020 Framework Programme under Grant Agreement No. 883335. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Document Information

List of Contributors	
Name	Partner
Dimitris Papadopoulos, Orestis Kompougias, Katerina Mitropoulou	INF
Thanasis Priovolos, Ilias Balampanis	SPH
Davide Sanvito, Roberto Bifulco	NEC
Leonardo Regano, Cataldo Basile	POLITO

Document History			
Version	Date	Change editors	Changes
0.1	06/12/2021	Dimitris Papadopoulos	Initial Table of Contents and assignment of contributions.
0.2	24/12/2021	WP5 partners	First round of contributions.
0.3	11/01/2022	WP5 partners	First full draft ready for internal review.
1.0	18/01/2022	WP5 partners	Final version

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	INF	19/01/2022
Quality manager	INF	19/01/2022
Project Coordinator	DBC	24/01/2022

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AIM	AOL Instant Messenger
API	Application Programming Interface
AS	Anonymization Service
BoW	Bag of Words
CMS	CountMinSketch
CSV	Comma Separated Values
DC	Data Collection
DCP	Data Collection and Data Preprocessing
DDoS	Distributed Denial of Service
DHCP	Dynamic Host Configuration Protocol
DL	Deep Learning
DNS	Domain Name System
DoS	Denial of Service
DP	Data Preprocessing
FN	False Negative
FP	False Positive
FPR	False Positive Rate
FTP	File Transfer Protocol
GAN	Generative Adversarial Network
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDS	Intrusion Detection System
IMAP4	Internet Mail Access Protocol version 4
IMAPS	Internet Mail Access Protocol over SSL
IP	Internet Protocol
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MAD	Multimodal Anomaly Detection
MitM	Man in the Middle
ML	Machine Learning
NNTP	Network News Transfer Protocol
NTP	Network Time Protocol
PALANTIR	Practical Autonomous Cyberhealth for resilient SMEs & Microenterprises
POP3	Post Office Protocol
RADIUS	Remote Authentication Dial-In User Service
ReLU	Rectified Linear Unit
REST	REpresentational State Transfer
RR	Recommendation and Remediation
SC	Security Capability
SCC	Security Capabilities Catalogue

Abbreviation / acronym	Description
SCHI	Security Capabilities Hosting Infrastructure
SCO	Security Capability Orchestrator
SDA	Semantic Data Aggregator
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SSH	Secure Shell
TAR	Trust, Attestation and Recovery
TCAM	Threat Classification and Alarm Management
TCP	Transmission Control Protocol
TF-IDF	Term frequency — Inverse document frequency
TFTP	Trivial File Transfer Protocol
TI	Threat Intelligence
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate
URL	Uniform Resource Locator
WP	Work Package

Executive Summary

This document presents the description of the first release of all the components related to the Hybrid Threat Intelligence Framework and related to the work carried out in the context of Work Package 5. All four tasks have been active through the first year of WP5 activities. An updated version will be presented in month 32 (April 2023) and will be focused on the second and final release of all the components.

After an overview of Threat Intelligence components, together with their differences with respect to the architecture described in deliverable D2.1 (Requirements & High-Level Design), the document describes the design of individual subcomponents, namely the Distributed Collection and Data Pre-processing (DCP), the Multimodal Anomaly Detection (MAD), the Threat Classification and Alarm Management (TCAM) and the Recommendation and Remediation (RR). Then, requirements collected in deliverable D2.1 related explicitly to WP5 components are reported and translated into technical specifications. Finally, before the conclusion, the last section provides in greater depth the implementation details for each one of the subcomponents presented above.

1. Introduction

1.1. Objectives and goals of the deliverable

This deliverable, "D5.1 Hybrid Threat Intelligence Framework – First Release" presents the low-level design and technical implementation of the components involved in the PALANTIR Threat Intelligence framework, within WP5. It highlights the work performed in tasks T5.1 to T5.4, spanning from the creation of scalable data ingestion and preprocessing pipelines to the training and evaluation of analytics-based anomaly detection and threat classification models, as well as to the production of recommended policies that are leveraged to mitigate the detected threats. For each subcomponent that comprises the Threat Intelligence Framework, their technical specs (obtained, iterated, and updated from the core requirements agreed upon in D2.1) and their low-level implementation details for the individual dependencies, libraries, and tools in use or anticipated to be used, are also provided.

The primary audience of this document are all technical consortium members; i.e., those involved in the implementation and technical decision taking, who participate either in WP5 and/or in the other directly or indirectly related WPs (such as WP3 and WP4). This deliverable provides design and implementation hints that can be of use to any external technical reader involved in the field of security analytics, leveraging data collection, aggregation and analysis capabilities to perform security functions that detect, analyse and mitigate cyberthreats.

This deliverable is the first iteration, out of two, within the WP5-related deliverables. The next deliverable (D5.2) is due in month 32 (April 2023), where the final design and implementation of WP5 components is expected along with finalised interactions, inter-component APIs and other interfaces in use to enforce the cross-WP workflows within the PALANTIR platform. The next release will also include the alarm management functionalities that will enable standardised threat intelligence sharing to/from external sources, via the Threat Sharing component described in D4.1. It should be noted that some technical aspects of this first release may be subject to change after January 2022, when the integration of all components will be the major focus, justifying any adaptation required to assure the components' integrability with the rest of the PALANTIR platform.

1.2. Relation with D2.1 and other WPs

Similarly, to the rest of the technical deliverables relevant to the first release of PALANTIR standalone components, D5.1 uses D2.1 as a starting point to dive into lower technical details about the WP5-related activities. This is primarily the case for Section 2, where the original PALANTIR architecture and agreed workflows are incorporated and adapted to the design of each WP5-related subcomponent. Moreover, the requirements indicated in D2.1 are mapped to technical specifications in Section 3, explicitly stating the need for the specific subcomponent impacted by the requirement.

Aside from that, the work presented here has a strong link with WP3-related activities as far as it concerns: a) the acquisition of traffic from monitoring Security Capabilities described in D3.1 that provides the input for the Distributed Collection and Data Preprocessing subcomponent (DCP) of D5.1 and b) the mapping of identified threats and their respective mitigation policies produced by the Recommendation and Remediation (RR) subcomponent of D5.1 to specific available SCs, consequently implemented as SecaaS services (described in D3.1).

Finally, the work of D5.1 is directly related to WP4, especially with the components described in D4.1. More specifically, this deliverable defines the workflows for threat detection that allow for threat monitoring and management through the cybersecurity dashboards of the Portal.

1.3. Structure of the document

The structure of the document in the following sections is explained below:

In **Section 2**, the overall design of the Hybrid Threat Intelligence framework is explained, based on the architecture of the project. Every subcomponent has its design described, including key ideas, kinds of data that is communicated or stored, and foreseen interfaces and communications channels with other PALANTIR components. Afterwards, any internal module is also described in its own subsection, which includes the explicit goal, behaviour and relations in the internal interactions or workflows devised for the subcomponent.

In **Section 3**, the relevant requirements from D2.1 are revisited and mapped to implementation specifications. This effectively refines more generic requirements into specific technical needs to be fulfilled, and is aimed towards a more clear and concise definition of the expected features of each subcomponent.

Section 4 compiles the ensemble of technologies, open-source tools and frameworks used in the technical implementation of each subcomponent and its respective modules. each module and submodule. It also emphasises their added benefits and adequacy to the envisaged setting through a per-module preliminary evaluation.

Section 5 is the conclusion and includes some remarks on what has been achieved and the next steps.

Finally, the provided annexes include the PALANTIR intra-subcomponent interfaces for each of the aforementioned subcomponents, as well as the definition of the data models that describe the main data sources for the current release.

2. Design

This section details the design choices for the WP5-related Threat Intelligence component and its internal subcomponents and modules. Specifically, the following sub-sections document the design for the Distributed Collection and Data Processing (DCP), the Multimodal Anomaly Detection (MAD), the Threat Classification and Alarm Management (TCAM) and the Recommendation and Remediation (RR) subcomponents.

2.1. Overview of the Threat Intelligence (TI) component

The Threat Intelligence (TI) component complements the protection provided by the Security Capabilities (SCs) part of the Secure Services Ecosystem (described in D3.1) with advanced analytics mechanisms based on Machine Learning (ML) and Deep Learning (DL) and provides automatically generated remediations to address the detected threats.

The following Figure 1 reports the high-level architecture presented in D2.1 and positions the TI component within the whole PALANTIR architecture.

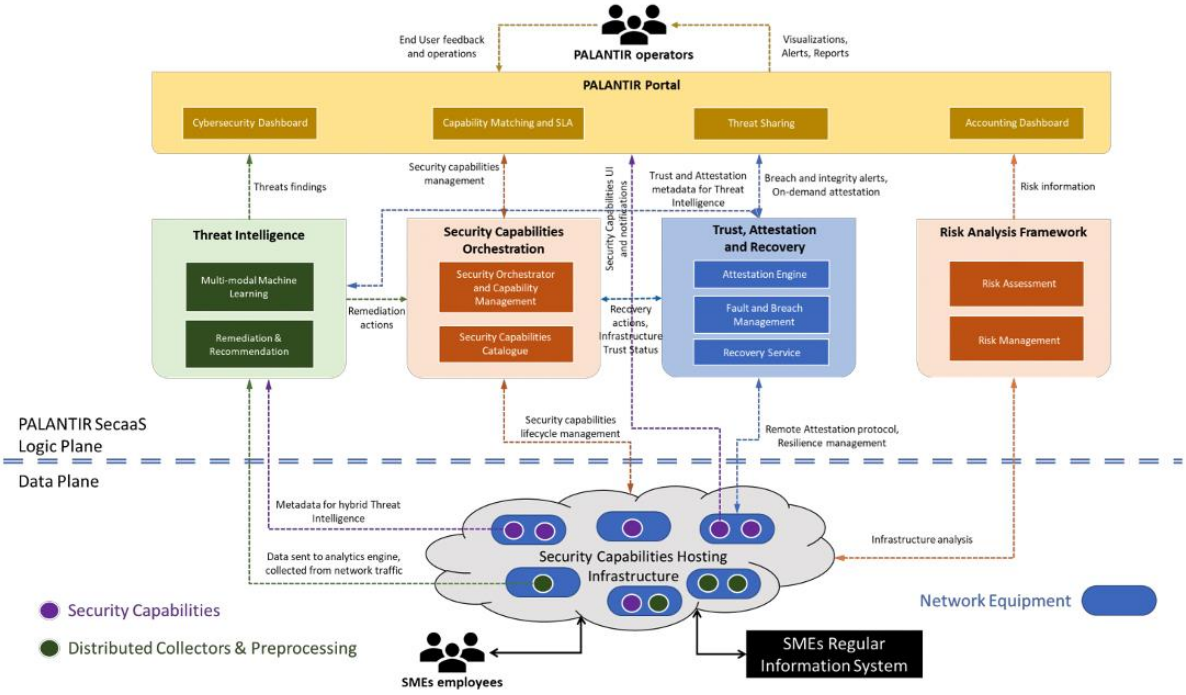

Figure 1: PALANTIR architecture

The TI comprises four subcomponents:

1. Distributed Collection and Data Preprocessing (DCP)
2. Multimodal Anomaly Detection (MAD)
3. Threat Classification and Alarm Management (TCAM)
4. Recommendation and Remediation (RR)

From a high-level perspective, the four subcomponents are arranged in a pipeline as reported in the following diagram (Figure 2).

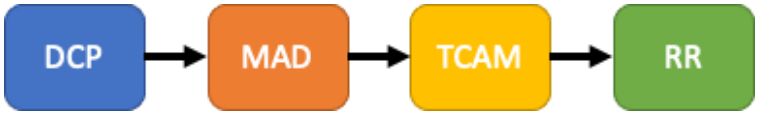

Figure 2: High-level WP5 subcomponents pipeline

The DCP subcomponent is in charge of collecting data from heterogeneous sources from monitoring SCs, running pre-processing and anonymization operations to prepare the data in a format suitable for the ingestion to the subsequent MAD subcomponent.

The MAD subcomponent implements a set of Anomaly Detection methods based on ML and DL taking into account multiple modalities of data from heterogeneous sources. Examples of modalities considered so far include network traffic flow information and system logs.

Once an anomaly is detected, a further step is performed by the subsequent TCAM subcomponent, whose task is the classification of the detected threat either as false positive (FP) or as a specific type of threat. In case the anomaly is not an FP, information is passed to the final RR subcomponent.

The RR subcomponent is responsible for the automatic generation of remediation for the specific type of detected threat by providing a set of policies which can be finally used to configure the appropriate remediation SCs.

The RR subcomponent is, however, not only handling the generation of remediations for threats detected and classified by MAD and TCAM subcomponents, but also for the ones identified by the monitoring SCs part of Secure Services Ecosystem. In this case, an SC should be able to directly trigger the RR subcomponent without involving DCP, MAD and TCAM subcomponents. As depicted in the following Figure 3, the TI component, by adopting a hybrid approach, simultaneously combines the analytics-based methods with more traditional signature-based Intrusion Detection Systems (IDSs) which are deployed as SCs.

Figure 3: High-level representation of hybrid Threat Intelligence according to DoA

In addition to the principal information flow presented above, there are additional per-subcomponent functionalities and intra-component interactions that will be detailed in the following subsections.

Regarding the interactions with other PALANTIR components, the TI is mainly involved in **two cross-component workflows** which are interleaved to the TI-specific pipeline described above: Event Handling and Periodic Attestation.

Figure 4 depicts the Event Handling workflow. The TI receives network traffic data from monitoring SCs through DCP's distributed collectors which are running within the Security Capabilities Hosting Infrastructure (SCHI). Optionally, system logs are also collected from assets protected by the PALANTIR solutions (e.g., a medical server containing sensitive data). The MAD subcomponent reads the input data and performs the Anomaly Detection. In case an anomaly is found, the TCAM subcomponent classifies the specific type of threat and forwards the threat results to the Portal component so that an alert is displayed to the user. At the same time, the RR subcomponent generates recommended policies to mitigate the identified threat. This step requires an interaction with the Security Security Capability Orchestrator (SCO). The RR subcomponent provides a list of mitigation options, which finally enables the user to request the the deployment of relevant SCs through an interaction with

the Service Matching (SM) component that is responsible for the selection of the optimal SC along with the provision of billing information to the user.

The Periodic Attestation workflow is reported in Figure 5. From the TI perspective, part of the steps up to the detection of a data breach are shared with the previous workflow for threat detection. The peculiarity of this type of event is that it requires additional operations which go beyond the ones deployable as SCs and handled by SCO. For this reason, this workflow does not foresee a direct interaction from RR to the Portal. The rest of the actions are instead handled by the Trust, Attestation and Recovery (TAR) component and analysed in greater detail in D4.2.

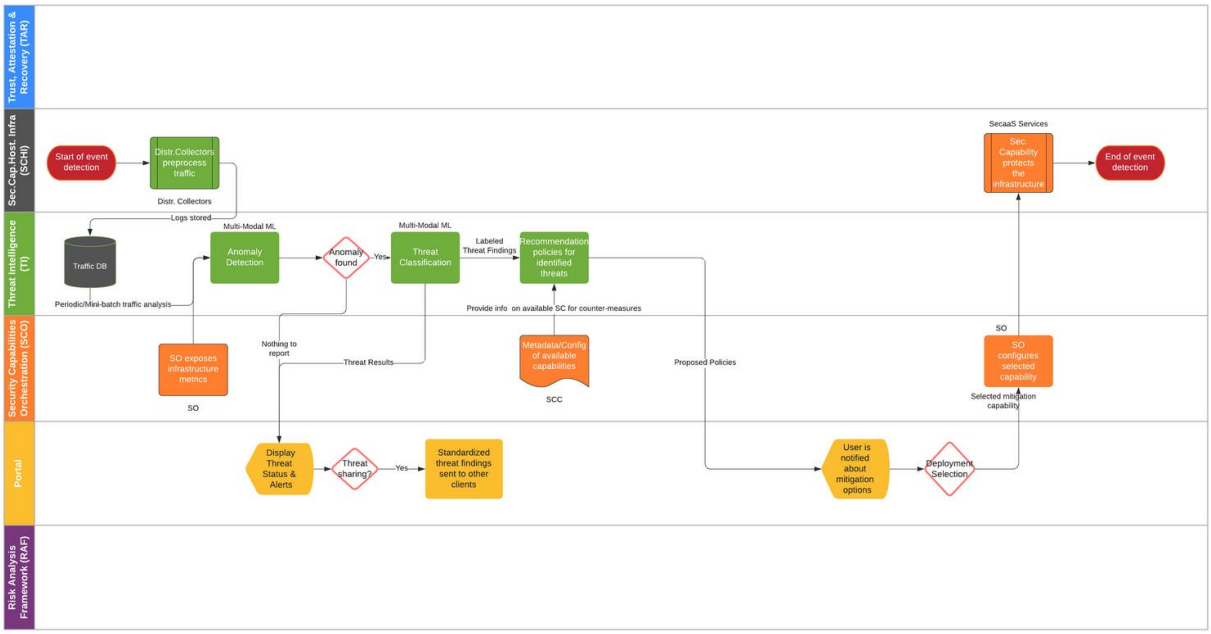

Figure 4: Event Handling workflow

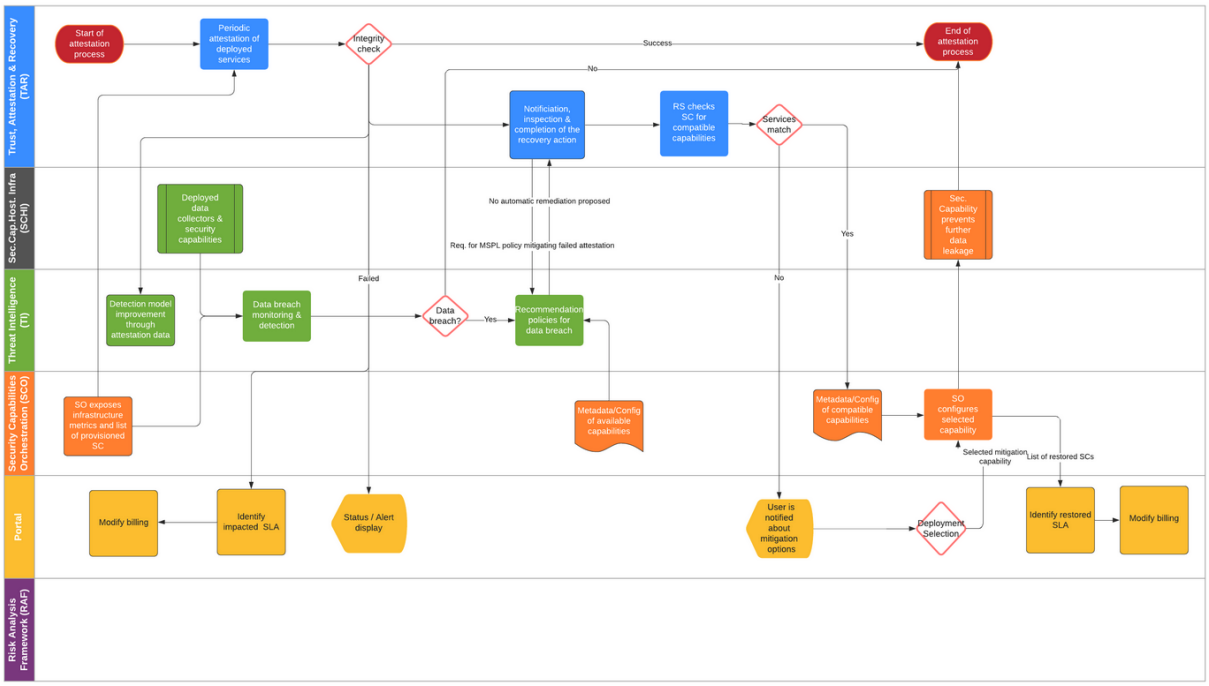

Figure 5: Periodic Attestation workflow

2.2. Differences with D2.1

With respect to the high-level architecture description reported in D2.1, this deliverable provides much more details regarding the design and implementation of the Threat Intelligence component. The reported workflows clarify the information flow within TI across its subcomponents and towards the other PALANTIR components.

The design of the hybrid approach described in D2.1, simultaneously combining the analytics-based methods with more traditional signature-based Intrusion Detection Systems (IDSs), has not been completely addressed yet. The first release of the TI component defines individual analytics modules, but their complete integration, which also requires cross-component interactions, has been postponed for the final release (due in April 2023). An example of their combination might be represented by the generation of meta-alerts that comprise ML-based (WP5) and rule-based (WP3) systems.

Another deviation is related to the interaction of the TI component with the TAR component, something which can be relevant for the detection of data breaches. According to D2.1, the attestation results from the TAR component are also forwarded to the TI. For the detection of these types of anomalies TI already includes a MAD module that can analyse syslog data from the deployed infrastructure. Its detection capabilities could be improved by also considering attestation data (e.g., failed attestation reports), but, given that this is achievable only after the full integration of all the PALANTIR components, the use of this additional information has been postponed to the second release.

Additional deviations from D2.1 and related to specific subcomponents are reported in dedicated per-subcomponent sections.

2.3. Description of Threat Intelligence subcomponent

2.3.1. Distributed Collection and Data Preprocessing (DCP)

The Distributed Collection and Data Preprocessing (DCP) subcomponent is responsible for collecting different types of data, pre-processing them and making them available for the rest of the PALANTIR components. DCP can achieve high throughput and low latency in the collection and preprocessing phases. Both the collection and preprocessing modules are distributed and scalable, in order to meet the infrastructure's requirements about the data volume. All modules of DCP can be deployed either on bare metal (vCPE delivery model) or on a Kubernetes (K8s) cluster.

The DCP subcomponent consists of three main modules: the Data Collection (DC) module, the Data Preprocessing (DP) module and the Anonymization Service module (AS). Also, there is DCP's storage, which uses OpenDistro Elasticsearch. In the current release, the DCP subcomponent supports collection and pre-processing of netflow data and syslog data.

Data Collection module consists of the Collector, the Registry service and the Source and Sink Connectors for Kafka. The Registry Service is responsible for storing all instances of Kafka Connectors, along with their health status (i.e., if a connector is down or not reachable). The Collector is responsible for collecting binary netflow data and dumping them into nfcapd files. Each time a nfcapd file is created, the Collector module must find an available Kafka Source Connector for netflow data to forward the collected nfcapd file. Registry service provides an endpoint, which returns the next available connector, achieving load balancing (round-robin) between all available connectors. The Collector gets the next available connector and sends the collected nfcapd file. The Kafka Source Connector for netflow data converts the received nfcapd file to .csv file and ingests it to Kafka. During ingestion process, each record of the converted .csv file is a separate Kafka message. The last part of the Data Collection module, namely the Kafka Sink Connectors, is responsible for ingesting the pre-processed & anonymized netflow data in DCP's storage.

Data Preprocessing module is responsible for two main operations: **i)** anonymization of the input IP addresses, **ii)** creation of features from the collected netflows that are relevant to other components. The Data Preprocessing module fetches collected netflows from a Kafka topic and writes back in Kafka the preprocessed & anonymized netflows, to be consumed from other components.

Anonymization Service module consists of a REST service and a storage layer. The storage layer uses a Redis database and stores pairs of the original and the obfuscated IP addresses. The developed REST service provides the required endpoints to anonymize and de-anonymize IP addresses. The anonymization function is implemented using the Crypto-PAn algorithm. The de-anonymization function is using Anonymization's storage layer to retrieve a de-anonymized IP address, given an obfuscated IP address.

Figure 6 shows the dockerized architecture of the DCP subcomponent deployed in PALANTIR's testbed and the flow described in the previous paragraph.

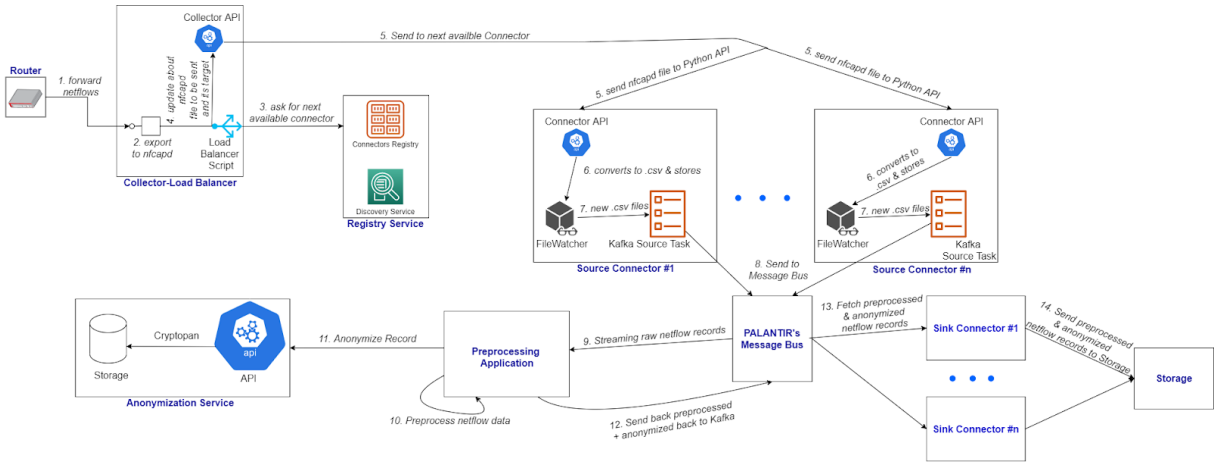

Figure 6: Dockerized Architecture of the DCP component

2.3.1.1. Interfaces with other components and subcomponents

The DCP subcomponent interacts with the deployed SCs in order to collect forwarded data from the infrastructure network devices. It also interfaces with the MAD subcomponent to provide anomynized and preprocessed streaming data for the execution of the anomaly detection algorithms. Finally, it interacts with TCAM, in order to de-anonymize the threat findings before they are further pushed to the RR subcomponent and to the Portal. Given that the DCP constitutes the core data exchange hub of the TI component, a full list of the implemented interfaces for the first release is provided in Annex A.

2.3.1.2. Modules

2.3.1.2.1. Collector

The Collector module is part of the Data Collection and Preprocessing subcomponent and initiates the ingestion chain for collected data. This module receives collected data and forwards them to the data collection module.

In the netflow use case, the Collector module listens to a specific socket, where binary netflows are forwarded from the PALANTIR infrastructure's network devices (i.e., a resource hosting SC instances). Every few seconds (or minutes), the collected netflows are dumped into nfcapd files. The Collector module forwards the created nfcapd files to the Data Collection module to be ingested in PALANTIR's message bus. Prior to forwarding these files, the target endpoint must be identified by the Collector module. Thus, it communicates with the Registry module, which returns the target endpoint for the collected file, and then forwards the created nfcapd file accordingly.

2.3.1.2.2. Registry

The Registry module is the unit of the Data Collection and Preprocessing subcomponent, which is responsible for holding information about all instances of the Data Collection module. The Data Collection module consists of multiple submodules, which are running in a distributed manner. A part of them is responsible for ingesting data to the PALANTIR's message bus while the rest of them are responsible for fetching data from the message bus and sending them to another module.

Whenever a data collection instance is started, it is registered in the Registry module, providing information regarding its name and its endpoint URL. Respectively, if an instance shuts down gracefully without errors, an un-register request must be sent in the Registry module to remove it from list of available instances. Moreover, the Registry module pings all available instances periodically to check their status. If an instance is down, it is removed from the list of available ones. Finally, as mentioned above this module is responsible for informing the Collector module about the target endpoint to which the collected data should be sent, achieving load balancing between the available instances.

2.3.1.2.3. Data collection

The Data collection module consists of multiple submodules, that materialise two different tasks: **i)** ingestion of collected and forwarded data to the message bus, **ii)** fetching of data from the message bus, conversion to appropriate format and forwarding to other modules (i.e., data storage module). Both tasks can be completed by running multiple distributed instances of each submodule, achieving high throughput and low latency.

The **Source connector** submodule is responsible for the former task (ingestion of data to the message bus). It consists of three components: **i)** an API to receive files, **ii)** a FileWatcher service to check for new files and **iii)** a SourceTask process, which ingests collected data to the message bus. In the netflow use case, the received files are nfcapd files, as they are created from the Collector module.

- The developed API provides an endpoint to receive nfcapd files from the Collector module. Every received file is converted to a .csv file and is stored under a pre-defined directory (configured via the source connector's properties). It is also responsible for registering and un-registering the source connector from the Registry module and for periodically updating its status providing its name and its URL endpoint via HTTP requests.
- The FileWatcher service is continuously running in the background checking the directory of the converted .csv files for new entries. It provides an interface, which must be implemented by all listeners who receive messages from it. Each time a new entry is detected it sends a message to all registered listeners, providing the file's name and location.
- The SourceTask process, as a registered listener for FileWatcher events, implements the interface provided by it. Following the observer pattern, when FileWatcher sends a new message, SourceTask receives it. This message contains information about the file that must be ingested in the message bus. SourceTask opens the file, reads it line by line and ingests each line as a different message in the message bus. For the netflow use case, each line describes a netflow record.

The **Sink connector** is the submodule which is responsible for achieving the latter task (fetching data from message bus and forwarding them to other modules). It can also have multiple instances and run in a distributed manner. For the netflow use case, sink connector module instances fetch preprocessed & anonymized netflows from the message bus, convert them to appropriate JSON format and then send them to the Data storage module. A similar process is also followed for the syslog use case.

2.3.1.2.4. Data anonymization

Data Anonymization module is composed of three submodules: **i)** the core application service which is responsible for data anonymization/ de-anonymization, **ii)** a fast storage layer for saving original and anonymized pairs of data, so that de-anonymization can be executed with low latency, **iii)** a web user interface for monitoring and assisting the usage of the storage layer.

In the netflow case the core application service anonymizes and de-anonymizes the IP addresses. Preprocessing module sends the IP addresses that need to be anonymized in the core application service and waits for an obfuscated IP address as response.

2.3.1.2.5. Data preprocessing

Data preprocessing module is responsible for applying all defined preprocessing functions in the raw data. The defined preprocessing functions create new features that are useful to the rest of the PALANTIR components, exploiting existing features of the collected data.

In the netflow case, five preprocessing functions are applied in collected raw netflow data. The first pair of functions is responsible for encoding the protocol *(pr)* and TCP flags *(flg)* columns using one-hot encoding. The second pair of functions calculates the total number of flow's packets and bytes. When they are applied to bidirectional netflows the result of the former equals to the sum of the ingoing packets *(ipkt)* plus the outgoing packets *(opkt)* whereas the result of the latter equals to the sum of the ingoing bytes *(ibyt)* plus the outgoing bytes *(obyt)*. In the case of non-bidirectional netflows the number of outgoing packets *(opkt)* and bytes (obyt) equals to zero, so the results of these two functions are equal to the ingoing packets *(ipkt)* and ingoing bytes *(ibyt)* respectively. The final preprocessing function extracts a new feature from the given netflow data, indicating if the destination port of the netflow is a commonly used port of a known service. Below are the listed services alongside with their commonly used ports (in parenthesis): FTP (20, 21), SSH (22), Telnet (23), SMTP (25), DNS (53), DHCP (67, 68), TFTP (69), HTTP (80), HTTPS (443), POP3 (110), NNTP (119), NTP (132), IMAP4 (143), SNMP (161), LDAP (389), IMAPS (993), RADIUS (1812), AIM (5190).

For the syslog case, a log processing pipeline has been established in order to apply machine learning on the collected system logs from monitored assets (e.g., protected servers), covering the need to convert the textual logs to numerical features. Subsequently, anomaly detection and threat classification algorithms are used to detect and classify logs that are correlated with potentially malicious behaviour.

Term frequency - Inverse document frequency (TF-IDF) algorithm was applied to transform the textual system logs to their vector representations (sparse embeddings). TF-IDF is based on the Bag of Words (BoW) model [1], which contains insights about the less relevant and more relevant words in a document. The importance of a word in the text is of great significance in information retrieval. The process is depicted in Figure 7.

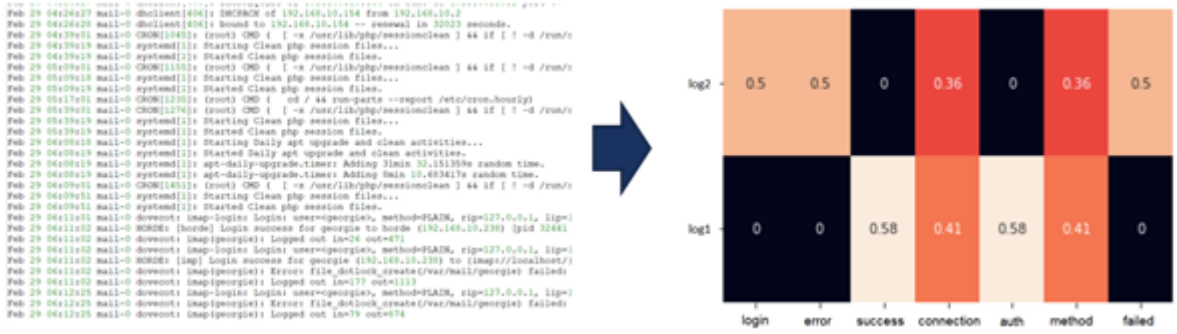

Figure 7: TF-IDF Log Transformation example

2.3.1.2.6. Data storage

The Data Collection and Preprocessing subcomponent includes a storage layer which enables the storing of the collected data. The collected data are converted to JSON format by the Data collection module and are then forwarded to the storage layer.

2.3.1.3. Differences with D2.1

So far there are some limitations compared to the Threat Intelligence paragraph in deliverable D2.1. Distributed collectors have been deployed as described. They can collect forwarded network and syslog data from any device either physical or virtual, if the data are in binary format describing netflow

records. The existing limitation (which will be addressed in the second release) is that only netflow records in binary format can be collected at the moment and there are no collectors for collecting event and logs from these devices. The collected data are preprocessed and anonymized in real time, and they are stored in a distributed file system as described in D2.1. Also, they are in format that can be read from Multimodal Machine Learning module.

2.3.2. Multimodal Anomaly Detection (MAD)

The Multimodal Anomaly Detection (MAD) subcomponent is responsible for running a set of Anomaly Detection modules able to detect abnormal behaviours from heterogeneous sources of data. The MAD subcomponent works in strict contact with the following TCAM module. Indeed, the detected anomalies can potentially contain security threats that will be specifically classified by the next subcomponent.

At the current status, two types of data sources have been considered: network traffic data and system logs. For each data type, multiple Anomaly Detection modules can be run in parallel, each one better suited for particular types of anomalies. This is depicted in the following sequence diagram. It should be noted that the results produced by multiple modules related to the same data modality (e.g., network traffic data) should be aggregated before triggering the threat classification. In other words, the TCAM subcomponent is triggered at most once per network flow, whenever at least one Anomaly Detection module (among the ones targeting network traffic data) marks the flow as anomalous.

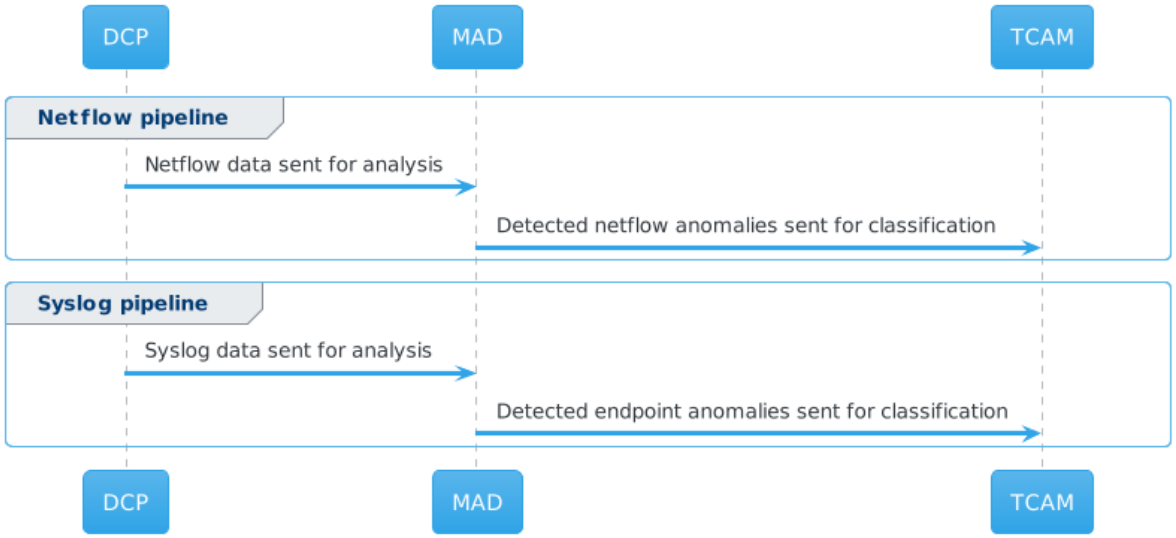

Figure 8: Sequence diagram for the MAD subcomponent

2.3.2.1. Interfaces with other components and subcomponents

The MAD subcomponent mainly interacts with DCP and TCAM subcomponents. The input data for the Anomaly Detection algorithms comes from DCP and, in case an anomaly is detected, output data proceeds further in the processing pipeline and is passed as input to the TCAM.

2.3.2.2. Modules

2.3.2.2.1. MIDAS for Network traffic analytics

MIDAS [2] is an Anomaly Detection algorithm suitable for dynamic graph data, i.e., a graph where the set of nodes and edges changes over time. Network traffic data can be mapped into a graph representation by considering the endpoints of a connection (e.g., the source and destination IP addresses) as graph nodes and the connections themselves as edges linking two nodes. MIDAS aims at detecting *microcluster anomalies* which are defined as unusual behaviour in terms of suddenly arriving

groups of suspiciously similar edges. Such edges are similar in two dimensions: spatially (regarding the nodes they connect in the graph) and temporally (regarding the time frame they appear in).

The input data for MIDAS, which also defines the granularity of its detection, is a single graph edge. Individual network connections are assigned an anomaly score which can be tested against a threshold to mark each connection as normal or anomalous. Further details about the tuning of the threshold are reported in Section 4.2. It should be noted that although edges are individually classified, the anomaly score is computed by taking into account the past history of the traffic or, in other words, the current status of the dynamic graph. Two types of anomalies are well detected by MIDAS: the ones manifesting, with respect to what has been observed in the past, an unusually higher rate of connections among the same pair of network nodes or an unexpectedly higher rate of network connections towards/from a set of nodes from/towards a single node.

In addition to the streaming nature of its approach, MIDAS has two additional features that make it appealing for the TI. First, the set of graph nodes (i.e., the source-destination pairs of the network connections) is not fixed a priori. Second, MIDAS implementation has a constant memory and update time. These two characteristics make it particularly suitable for the real-time requirement of TI.

2.3.2.2.2. Isolation forest for System log analytics

Isolation Forest is an anomaly detection algorithm that exploits the concept of "isolated" observations after applying a random forest of decision trees [3]. The reasoning is simple, anomaly observations are easy to isolate because they will show a significantly shorter path length (Figure 9). Isolation Forest is suitable for diverse dataset types and shows an acceptable memory usage, rendering it a promising technique to apply in anomaly detection for cybersecurity incidents based on large batches of system logs. It is also worth mentioning that the training process can be achieved with normal and anomalous traffic in the same dataset, thus making it valid for production environments.

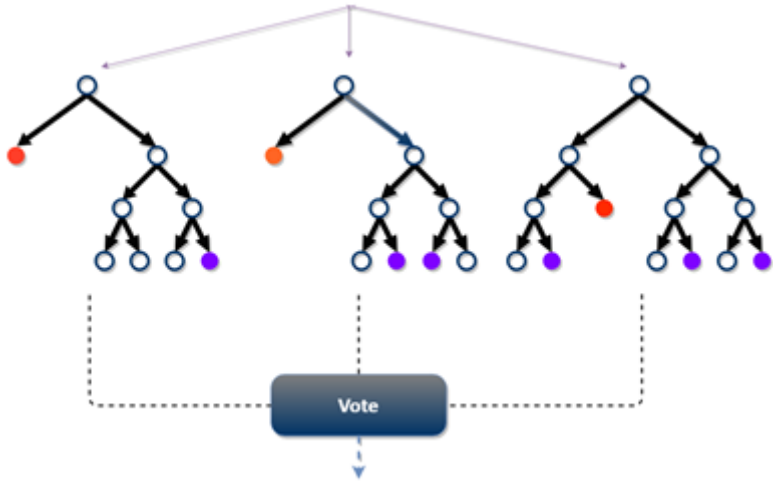

Figure 9: Isolation Forest. Outliers (red) are less frequent than regular observations and require less splits (closer to the root of the tree)

2.3.2.2.3. Deep Autoencoder for Network traffic analytics

The vanilla implementation of Autoencoders is a Neural Network architecture whose purpose is to learn the underlying distribution of data by forcing dimensionality reduction and reconstruction of the original input. Autoencoders receive an input $x \in R^N$, which gets passed through a series of Neural Network layers that produce progressively smaller outputs (Encoder), as shown in Figure 10. This bottleneck performs dimensionality reduction of the input x to a latent vector $z \in R^{L<N}$. The latent vector is then passed through a series of Neural Network layers that produce progressively bigger outputs (Decoder) producing the output x0 with the goal of reproducing the original input x. Autoencoders for non-binary regression are trained using the Mean Squared Error loss.

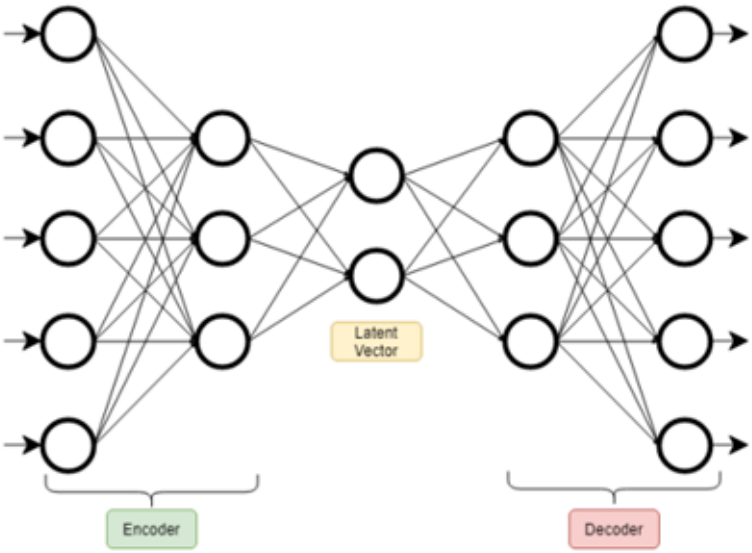

Figure 10: Autoencoder architecture. The forward pass of data is from left to right. The input is first encoded into a latent vector and then decoded, producing the reconstruction of the input

In order to detect anomalous behaviour, Autoencoders are trained with normal/benign traffic only and are expected to produce outputs with a high loss when fed with anomalous data. This happens because anomalies do not belong to the distribution of normal behaviour that is learned by Autoencoders.

2.3.2.2.4. GANomaly for System log and Network traffic analytics

The GANomaly architecture proposed comprises a Generative Adversarial Network (GAN) [4], [5] based on the aforementioned Autoencoder architecture, purposefully built for outlier detection purposes. A GAN is comprised of two neural networks contesting with each other in a zero-sum game, depicted in Figure 11.

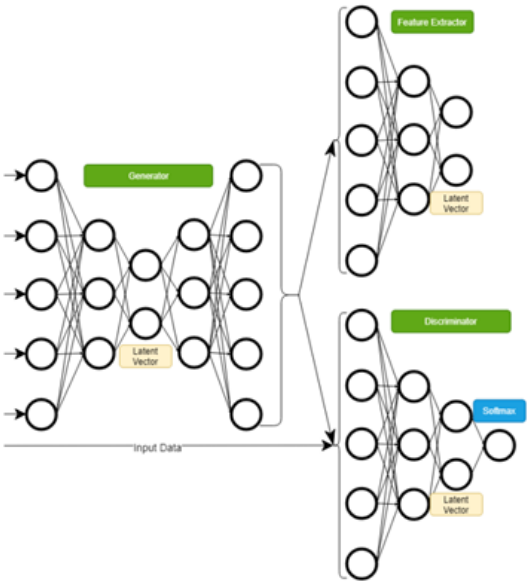

Figure 11: GANomaly architecture. A Generative Adversarial Network that relies on 3 autoencoders, the Generator, the Feature Extractor and the Discriminator

The Generator network aims to learn the underlying data distribution and produces samples from the learnt distribution. The Discriminator network identifies data as either real or generated by the Generator. After training, the Generator can be used to generate new samples or perform various other tasks. GANomaly is a GAN that uses an Autoencoder as the Generator. Two extra encoders are also employed, one being the Discriminator and the other being a feature extractor that re-encodes the reconstructed input. The Generator's loss is a weighted sum of three loss functions which all aim to help the Generator learn the underlying data distribution so that any outliers stand out from the other data points.

On inference mode, the prediction relies on deciding whether each data sample is an outlier or not, given its anomaly score A. This score is calculated by taking the difference between the feature extractor's latent vector and that of the Generator, which we scale to [0,1]. A high anomaly score means a high confidence of the sample being malicious, while a low score means that the sample is predicted as being normal. The decision boundary can be defined on a per-use case basis depending on the precision-recall trade-off that is acceptable.

2.3.2.3. Differences with D2.1

As anticipated in Section 1.2, the main deviation relevant to the MAD subcomponent is related to the combination of analytics-based methods with more traditional signature-based Intrusion Detection Systems (IDSs). This combination requires interactions with other PALANTIR components (specifically with WP3 components) and will be addressed in the second release.

2.3.3. Threat Classification and Alarm Management (TCAM)

The outlier flows (flows that signify malicious/suspicious behaviour) detected by the MAD module, are pushed to the TCAM module. The latter is responsible for classifying them either as false positives or as threats, and providing a corresponding confidence score for each predicted label.

In a similar way to the MAD subcomponent, TCAM is responsible for assigning threat labels on two distinct data modalities: network traffic data and system logs. For each data type, the corresponding TCAM modules are implemented as independently trained models which can be run in parallel.

In this first release, two Random Forest machine learning models were trained to support the core threat detection functionalities for the aforementioned data modalities. It should be noted that, although their algorithmic design is similar in both cases, each model receives different data as input (i.e., nfcapd files, system logs) and is relevant for complementary attack scenarios (i.e., network-based threats, endpoint-based threats). The operations handled by TCAM are illustrated in the following sequence diagram.

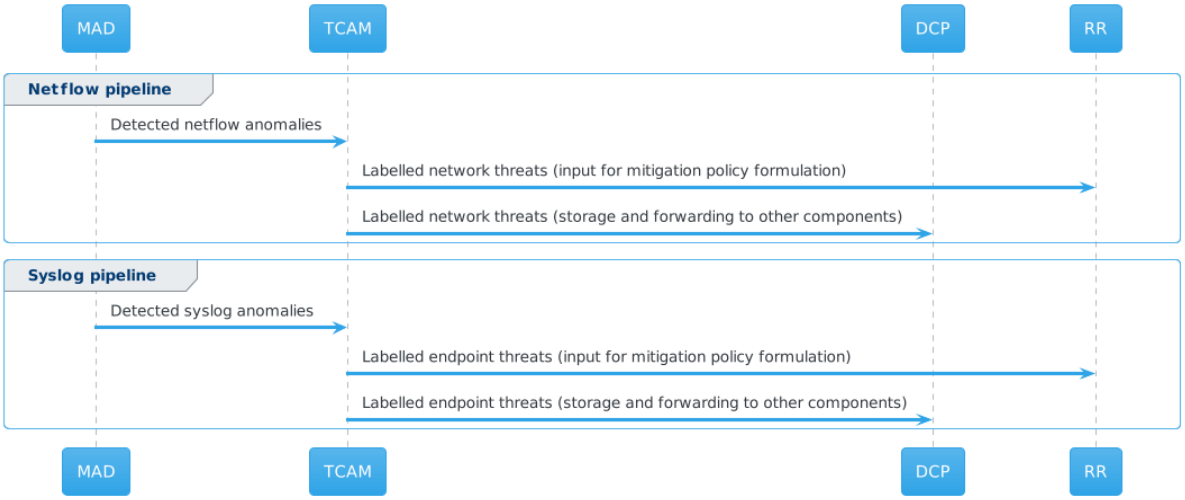

Figure 12: Sequence diagram for the TCAM subcomponent

2.3.3.1. Interfaces with other components and subcomponents

The TCAM subcomponent interacts with the DCP, MAD and RR subcomponents. The output of the Anomaly Detection algorithms from MAD (detected anomalies) is provided as input to the TCAM modules. TCAM then assigns a threat label to each identified anomaly along with a classification confidence score and passes this information to the RR subcomponent which is tasked with proposing remediation policies relevant for the threat at hand, and to the DCP subcomponent which handles storage and forwarding to other PALANTIR components (e.g., Portal).

2.3.3.2. Modules

2.3.3.2.1. Random Forest

Random Forest is an ensemble, supervised machine learning method used for classification. It constructs a multitude of decision trees at training time and outputs the mode of the classes (the most repeated value) of the individual trees as the final class [6]. Essentially, each tree's prediction is counted as a vote for one class and the final label is predicted to be the class which receives the most votes (majority vote) (Figure 13). The algorithm applies the general technique of bootstrap aggregation (or bagging) to tree learners, leading to a better performance model by decreasing the variance, without increasing the bias. Random forest is considered one of the best-performing ML algorithms, mainly because of its ability to remove decision trees' habit of overfitting the training set (being too much dependent on the training set and not performing so well in the testing set) and of its excellent classification accuracy compared to current algorithms [7]. In the case of network traffic classification, the datasets are usually unbalanced since the majority class (normal traffic) is usually orders of magnitude higher than the minority classes (attack flows). Therefore, classifiers are overwhelmed by the dominating class and tend to ignore the flows related to malicious activity. Random forest is of no exception, thus techniques like cost-sensitive learning and oversampling of the minority class are leveraged to tackle this issue.

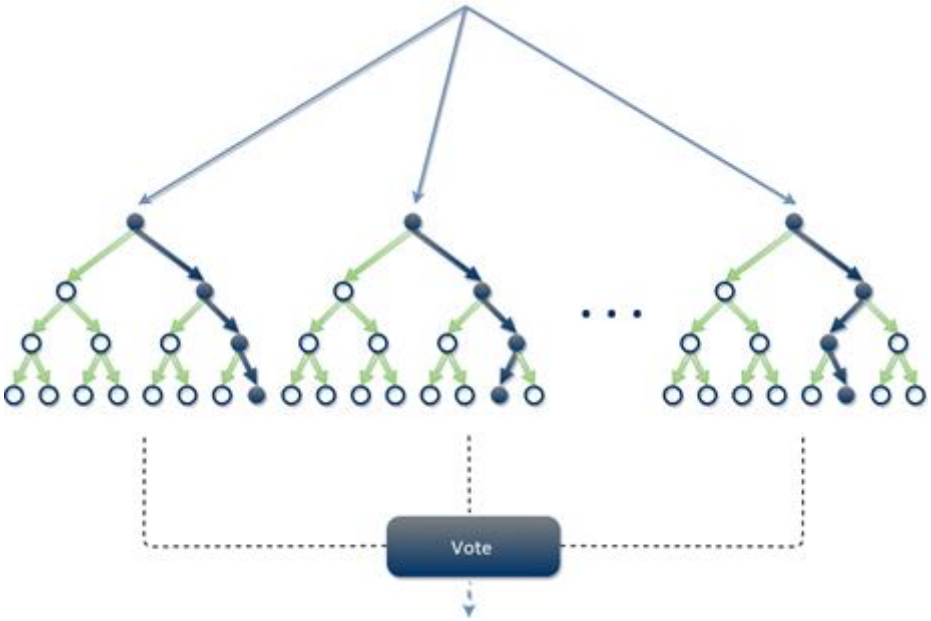

Figure 13: Random Forest architecture. It constructs a multitude of decision trees at training and outputs the mode of the classes of the individual trees

2.3.3.3. Differences with D2.1

Two main deviations relevant to the TCAM subcomponent are identified below:

- According to D2.1, only network-based information is provided as input to the classification algorithms of the Hybrid Threat Intelligence component. However, during the early

implementation activities of the project, it became apparent that specific types of threats (specifically those related to endpoint events such as data breaches) are typically not detectable using this modality. To this end, WP5 partners decided to further extend the detection capabilities of the TI by introducing an additional data modality (syslog data), resulting in the implementation of the corresponding classification algorithm that enables real-time monitoring of endpoint events by consuming system logs.

- Given that the focus of T5.3 partners for this first release was given on the training of threat classification models for the aforementioned modalities and the integration activities with the rest of the TI subcomponents, the threat sharing functionalities mentioned in D2.1 ("Threat Intelligence data is shared using STIX format") was pushed to the final release of the TCAM subcomponent.

2.3.4. Recommendation and Remediation (RR)

The Recommendation and Remediation tool is in charge of:

- recommending the actions to take in order to mitigate the increased threats that other PALANTIR tools have identified. The information about the risks to mitigate is reported in a Threat Intelligence Report from the pipelined analytics components (i.e., TCAM threat findings). The solutions that are identified by the RR tool named *remediation recipe*, which are sequences of actions that are explained in an abstract format.
- deploying the recipe selected by the PALANTIR user, which produces the set of changes to perform to the landscape (e.g., adding new security controls, such as SC configrations) and the changes to the configuration of the security controls in the landscape (including the ones that the RR tool proposes to add).

All the remediation recipes are characterised by:

- a set of *labels* that indicate the threat scenario for which they have been developed.
- a set of *enabling constraints* that allow understanding all the constraints for their applicability. For instance, they report all the information necessary for their correct deployment and the security capabilities they need to be enforced (e.g., a layer7 filter), which may not be available in the network.
- the set of remediation *deployment instructions*, written in an abstract language, that programmatically state all the steps that need to be performed to remediate the identified risks.

Figure 14 reports an example recipe that can be fed to the Recommendation and Remediation module. In particular, this recipe is able to remediate an ongoing attack on a specific host in the network by inserting in the path between the attacker and the target host a control on a specific payload. For example, this can be useful if the impacted host has become part of a botnet, and the Command and Control messages between them exhibit a specific string that can be filtered to disrupt the communication between the bot and the botnet master.

The language describes different concepts, which are reported here using different colours:

- the operations that are available as they are exposed either from the PALANTIR framework or any of its components, in green, e.g.:
  - adding security controls,
  - modifying the configuration of specific security controls,
  - modification to the network layout or flows,
- language-specific concepts, in red, introduced to satisfy the language required features, e.g.:
  - results from past computations,
  - placeholders for predefined concepts,
- inputs from the threat intelligence findings from TCAM, in orange.

It should be noted that the actions are tailored to the actual landscape of the target network. In this case, if a security control with payload filtering capability is already present in the path from the impacted host to the attacker, the existing control is reconfigured to filter the target payload; otherwise, new

security control is placed in front of the impacted host, and is properly configured with the needed filtering rules.

```
list_paths from impacted_host_ip to 'attacker'
        iterate_on path_list
                find_node of type 'l7filter' in iteration_element
                if not present
                        add_firewall behind impacted_host_ip in iteration_element with level 7
                        add_filtering_rule predefined_rules to new_node
                else
                        add_filtering_rule 'filter_payload_X' to found_node
                endif
        end iteration
```

Figure 14: Example recipe of the Recommendation and Remediation module

The target landscape is described using a Landscape Description Language. This is a graph-based representation of the network layout, which describes both nodes (and their attributes, e.g., capabilities) and edges. This representation is prone to be imported with graph libraries available for the main programming languages (e.g., iGraph on Python). Currently, it is represented as a simple text file that follows the specification defined during a past EC-funded project (SECURED [8]). However, the final format for the landscape description has not been decided. Furthermore, we are also investigating the possibility of avoiding using network graphs if the network flows will be expressive enough for our needs.

This component will implement the following workflow, which can be divided into two phases:

- recommendation of recipes (see Figure 15);
- deployment of the selected recipe (see Figure 16).

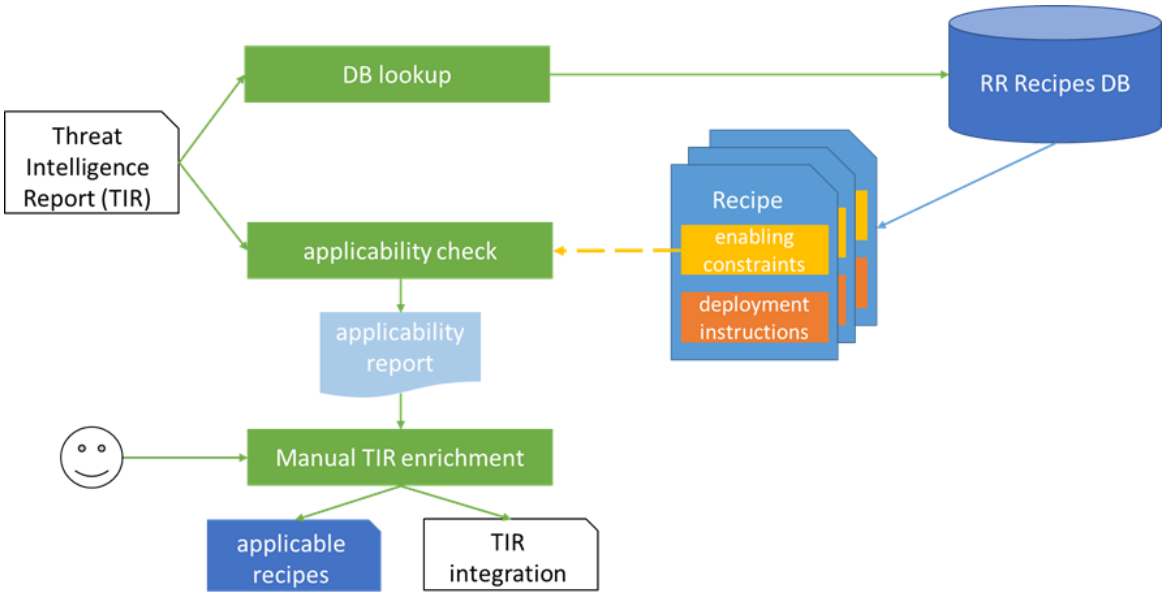

Figure 15: The RR tool workflow, phase 1: recommendation

When the RR tool receives the notification of a risk to mitigate in the form of a Threat Intelligence Report (TIR), it uses the information in the TIR to look up into a database where all the remediation recipes are stored (DB lookup). All the remediation recipes are labelled according to a standard set of

categories. For instance, the recipe to mitigate the risks from a malware infection is labelled as 'malware' and further refined with additional more specific data, as 'botnet' or 'ransomware.' If needed, custom recipes against specific strains of malware may be added to increase the efficacy of the proposed mitigations. The TIR includes the same set of labels, allowing very fast filtering of the recipes.

The recipes produced by the DB lookup phase are all the recipes that can mitigate a specific set of threats. Nonetheless, it is not ensured that these recipes could be deployed in the current threat scenario affecting the target landscape at this stage. For this purpose, an additional step is performed, named Applicability Check, where all the enabling constraints are evaluated. The applicability report lists all the directly applicable recipes. All the recipes that have not been evaluated as applicable report the constraints that were not satisfied. Therefore, a user analysing the report can provide additional information that can make additional recipes applicable (e.g., missing IP/URL information or missing information about honey pot networks). The additional data provided by the users are saved in a special data structure named TIR integration (e.g. within the RR subcomponent). In this way, these data are neither forgotten nor merged with official information coming from the tool. It is a future task to understand how this information can be integrated into future versions of the PALANTIR framework.

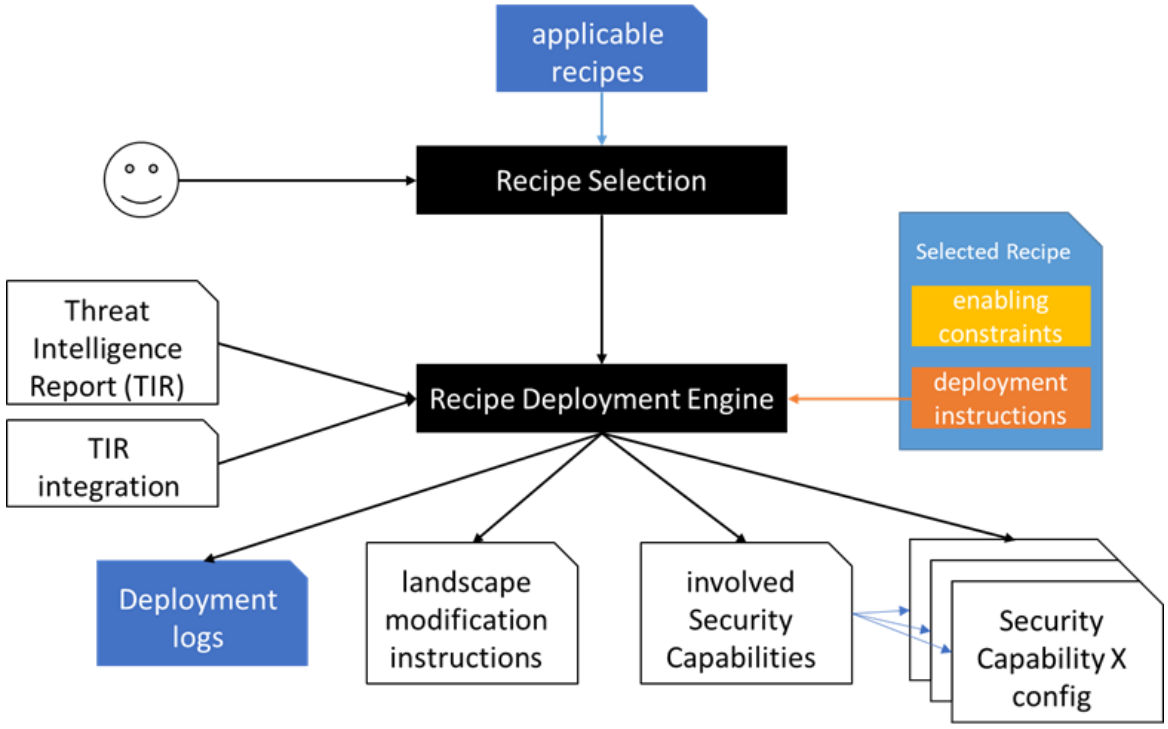

Figure 16: The RR tool workflow, phase 2: deployment

The next phase of the RR tool workflow starts when the user decides the remediation recipe to enforce. At this point, the Recipe Deployment Engine reads and starts interpreting the deployment instructions. In the first phase, all the generic concepts are made concrete with the TIR and the TIR Integration information. Taking as an example the recipe in Figure 14, the generic concepts "*impacted_host_ip*" to "*attacker*" are substituted with their actual IP addresses, as provided by the TIR (following a de-anonymization process from DCP), in order to list all the possible paths between the impacted host and the attacker.

A remediation recipe interpreter executes the deployment instructions with concrete information and generates as output:

- (optionally, if the networked scenario where the remediation takes place needs it) a set of suggested changes to the landscape. It should be noted that while some of these changes may be performed automatically based on the functionalities of the existing SCs, others will require access to the network/infrastructure from the client's side . Examples of these changes are:

- moving network nodes to a different position in the network
- removing nodes from the network
- changing the network connectivity (either by connecting a node to a different network or by redefining the flows by changing the routing information)
- adding nodes (e.g., security capabilities)

- a list of all the security capabilities involved by the remediation together with the changes to the configurations of all the involved security capabilities, either already present in the landscape or proposed by the RR tool and evaluated by the SM component in terms of feasibility and cost. These configurations are provided with an abstract language (medium-level policy language) that configures standard security features.

2.3.4.1. Interfaces with other components and subcomponents

The RR tool interacts (or will interact in future iterations) with the following PALANTIR components:

- TCAM, which will provide this subcomponent with the information discovered by the threat intelligence;
- Security Capability Orchestrator (SCO), which will process the instructions provided as output by this subcomponent to actually remediate the threat scenario identified by the TCAM.
- Service Matching (SM) for proposing a deployment plan per identified recipe based on the current status of the infrastructure.
- Recovery Service (RS), in cases that involve infrastructure-related mitigation actions (e.g., node attestation scenarios).

2.3.4.2. Modules

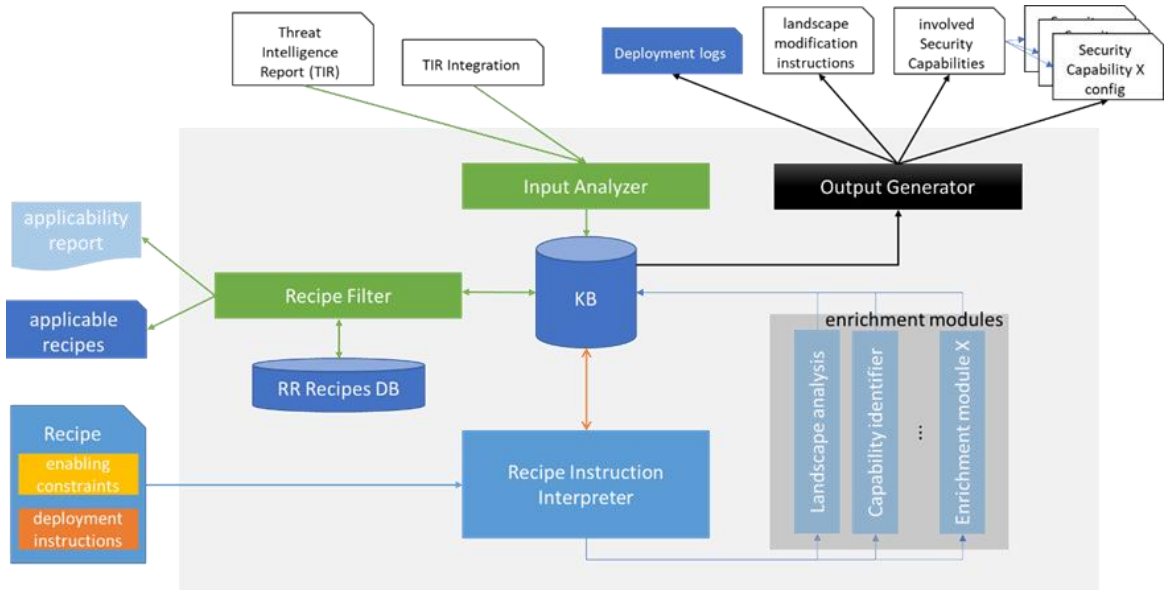

Figure 17: Architecture of the Recommendation and Remediation tool

This section provides insights into the Recommendation and Remediation tool (see Figure 17), particularly detailing the sub-modules constituting the tool and the communications intercurring amongst them.

2.3.4.2.1. Input Analyzer

The Input Analyzer is tasked with the interpretation of the Threat Intelligence Report. It extrapolates from the TIR the information needed to enrich the recipe instructions with concrete information. This includes the type of risk that must be remediated and the IP addresses (and possibly the TCP/UDP ports) of the impacted hosts and the attacker.

The input Analyzer stores the interpreted information into a Knowledge Base (KB), which is used to store all the data produced by all the RR tool components.

The Input Analyzer is also the module that parses and stores into the KB the additional information provided by the user as TIR Integration.

2.3.4.2.2. Recipe Filter

The Recipe Filter is the module that reads from the KB both the TIR and TIR integration. It extracts the labels and the additional information that allows categorizing the threats. From the extracted information, this module properly selects the Recipe that applies. More precisely, this module queries the RR Recipe DB and:

- extracts the applicable recipes
- checks the satisfaction of the enabling constraints associated with selected recipes
- produces the Applicability Report
- produces the Applicable Recipe list.

The Applicability Report is then presented to the user. The Applicability report is currently a list of Recipes and the information that the Recipe Filter was not able to collect or verify the Enabling Constraints. Examples of Enabling constraints are:

- check for the presence of specific attributes (e.g., the IP addresses of the C2 for malware infections);
- need for specific security capabilities (e.g., if the Security Capability Catalogue contains an element able to filter by URLs).

The recipes in the Applicability Report for which all the constraints are satisfied are listed in the Applicable Recipe List.

2.3.4.2.3. Recipe Instruction Interpreter

This module is in charge of deploying the recipe that it receives as input. The recipe to deploy is received as user input.

When the user selects a recipe amongst the applicable ones presented by the Recipe Filter, the Recipe Instruction Interpreter (RII) interprets the deployment instructions contained in the recipe. This module concretizes them using the information contained in the Knowledge Base.

For the interpretation of deployment instructions, the RII may need to delegate computations to specific Enrichment Modules. These modules expose methods and attributes that can be used when writing recipes and save their results. Currently, two Enrichment Modules have been implemented.

- The *Landscape Analysis Module* is in charge of working on the network graph. For instance, it exports:
  - *operations on nodes and edges*, e.g., querying for nodes having specific properties (find_node)
  - *operation on paths*, e.g., list all the paths between two network nodes, check reachability between nodes (list_paths)
- The *Capability Management Module* represents the interface to the PALANTIR Security Capability Catalogue (the ‘Level 7’ used when adding a firewall).

The result of the execution of the recipe, which is also stored in the KB, is a set of instructions that represent the modifications to the current landscape:

- changes to the landscape such as:
  - adding new nodes, including adding new security capabilities
  - deleting edges and disconnecting nodes from the network
  - moving nodes to different networks
- change to the security capabilities configurations:
  - adding rules to the configurations of specific security capabilities.

2.3.4.2.4. Output Generator

When the interpretation of all the deployment instructions is complete, the Output Generator reads the KB and provides the output to the user.

This module outputs:

- the needed modifications to the landscape, including, for example, new security capabilities that must be deployed and the necessary alterations to the connections amongst network nodes, which will be sent to the Security Capability Orchestrator for their deployment;
- the configurations for both the new and the existing Security Capabilities, which will also be sent to the Security Capability Orchestrator for deployment;
- a set of deployment logs describing all the actions taken to remediate the risk.

This module is currently a simple interface to the KB. However, more features could be needed in the future for directly interacting with the Security Capability Orchestrator and pushing commands.

2.3.4.3. Differences with D2.1

The D2.1 only provided the description of the functionalities and requirements of the RR Module. Both these data have not been changed. On the other hand, this deliverable presents the initial architecture, workflows, and data models.

3. Specifications

This section continues with mapping the relevant PALANTIR requirements provided in D2.1 to actual technical specifications for each TI subcomponent. The technologies and frameworks mentioned in this section are further described in Section 4.

3.1. Distributed Collection and Data Preprocessing

Table 1 Specifications of the DCP subcomponent

Req. ID	Requirement description	Collector	Registry	Data Collection	Data Preprocessing	Data Anonymization	Data Storage
R1.1.3	The platform MUST provide near-real-time (NRT) data processing functionalities.	✓	✓	✓	✓	✓	✓
DCP_S1	A tool; named nfcapd has been used for collecting netflows in the Collector module. These collected netflows are the input of the DCP subcomponent. Kafka connectors in the Data Collection module implemented Kafka Connect API have been used for ingesting data to Kafka and fetching data from Kafka to insert them to Elasticsearch. Data Preprocessing module is a Spark Streaming application which preprocess and anonymizes collected data, using the Data Anonymization module, which implements Crypto-PAn algorithm.						
R1.4.6	The platform SHOULD provide an AI based solution to deliver services, and be shared across the plain field; however, the data-sharing must ensure anonymity.	--	--	--	--	✓	--
DCP_S2	The anonymization of data is happening in Data Anonymization module. This module uses Crypto-PAn algorithm for anonymizing IP addresses. It also provides an API for HTTP requests for anonymizing or de-anonymizing IP addresses. The mapping between the original and the obfuscated IP addresses is stored in a Redis database.						
R1.5.1	The platform SHALL be able to collect and analyse events from heterogeneous sources in near real time in order to detect security incidents.	✓	✓	✓	✓	✓	✓
DCP_S3	A tool, named nfcapd has been used for collecting netflows in the Collector module. These collected netflows are the input of the DCP subcomponent. Kafka connectors in the Data Collection module implemented Kafka Connect API have been used for ingesting data to Kafka and fetching data from Kafka to insert them to Elasticsearch. Data Preprocessing module is a Spark Streaming application which preprocess and anonymizes collected data, using the Data Anonymization module, which implements Crypto-PAn algorithm. The scalability offered by Data Collection and Data						

	Preprocessing modules can handle a big volume of data with low latencies and achieve near real-time ingestion and preprocessing.						
R1.5.2	The platform SHALL be able to analyse and combine different modalities of data to detect anomalies in nearly real time.	✓	✓	✓	✓	✓	✓
DCP_S4	A tool, named nfcapd has been used for collecting netflows in the Collector module. These collected netflows are the input of the DCP subcomponent. Kafka connectors in the Data Collection module implemented Kafka Connect API have been used for ingesting data to Kafka and fetching data from Kafka to insert them to Elasticsearch. Data Preprocessing module is a Spark Streaming application which preprocesses and anonymizes collected data, using Data Anonymization module, which implements Crypto-PAn algorithm. The scalability offered by Data Collection and Data Preprocessing modules can handle a big volume of data with low latencies and achieve near real-time ingestion and preprocessing.						
R1.5.7	The data involved in the analytics processes MUST be anonymized.	--	--	--	--	✓	--
DCP_S5	The anonymization of data is happening in the Data Anonymization module. This module uses Crypto-PAn algorithm for anonymizing IP addresses. It also provides an API for HTTP requests for anonymizing or de-anonymizing IP addresses. The mapping between the original and the obfuscated IP addresses is stored in a Redis database.						

3.2. Multimodal Anomaly Detection

Table 2 Specifications of the MAD subcomponent

Req. ID	**Requirement description**	**MIDAS**	**Isolation Forest**	**AutoEncoder**	**GANomaly**
R1.5.2	The platform SHALL be able to analyse and combine different modalities of data to detect anomalies in nearly real time	✓	✓	✓	✓
MAD_S1	Different Anomaly Detection modules from MAD subcomponent target different modalities of data, e.g., network flow data and system logs. The requirement of operating in near real time is addressed either by design (e.g., MIDAS implementation works in constant time and space) or by using scalable frameworks for ML inference (e.g., PySpark).				
R1.5.5	The platform SHOULD provide analytics able to detect the most common threat types (malware, MitM, volumetric attacks).	✓*	✓*	✓*	✓*

MAD_S2	The benchmark datasets used for the preliminary evaluation covers part of the most common threat including, e.g., malware, DoS, DDoS and PortScan traffic. In this first release, MitM traffic is not available in our datasets.				
R1.5.6	The platform SHOULD provide analytics able to detect phishing attacks.	--	--	--	--
MAD_S3	Not addressed in the first release. The fulfilment of this requirement is planned for the second release (D5.2).				
R1.5.7	The data involved in the analytics processes MUST be anonymized	✓	✓	✓	✓
MAD_S4	Input data for MAD modules is provided as output of DCP after personal information has been anonymized.				
R1.5.8	The platform SHALL provide periodic retrain functionalities for its analytics components (e.g., on a monthly basis).	--	--	--	--
MAD_S5	The first release has been more focused on the inference phase, assuming a one-time offline training for the models. DCP subcomponent already defines in its operations the storage of the data in ELK. The amount and type of data available for periodic re-training is also affected by the data retention policies. The fulfilment of this requirement is planned for the second release (D5.2).				
R2.1.1	The analytics of the platform SHOULD be able to scale with respect to the number of data sources, the volume and the velocity of data streams.	✓	✓	✓	✓
MAD_S6	The scalability of the analytics components is achieved either by design (e.g., MIDAS implementation works in constant time and space) or by using scalable frameworks for ML inference (e.g., PySpark).				
R2.1.2	The analytics components of the platform SHOULD be able to deal with the computational and memory limitations posed by large datasets.	✓	✓	✓	✓
MAD_S7	The technical implementation of MAD components is based on scalable distributed machine learning frameworks and on linearly scalable algorithms with respect to the input data.				
R2.2.1	PALANTIR deploys various big data analytics frameworks that have demands in computational power. They MUST be regularly evaluated during development, such that they are shown to be accurate with real-time data.	✓*	✓*	✓*	✓*
MAD_S8	The preliminary evaluation of benchmark datasets (Section 4) confirms accurate inference on real-time data. The fulfilment of this requirement is planned for the second release (D5.2).				
R2.2.2	PALANTIR SHOULD outperform existing conventional methods from potential competitors.	--	--	--	--

MAD_S9	Fulfilment of this requirement premises an integrated PALANTIR platform. It will therefore be addressed in the final release of the Hybrid Threat Intelligence framework (D5.2).				
R2.2.3	The time to discover critical info & alerts in the security dashboard MUST NOT exceed 1 minute.	✓	✓	✓	✓
MAD_S10	The time required to process a single flow by all anomaly detection components (MIDAS, IF, AutoEncoder, GANomaly) is less than 1 second.				
R2.2.7	The platform MUST showcase a reduction of false positives and negatives of at least 15% compared to commercial solutions.	--	--	--	--
MAD_S11	Fulfilment of this requirement premises an integrated PALANTIR platform. It will therefore be addressed in the final release of the Hybrid Threat Intelligence framework (D5.2).				
R2.6.2	The PALANTIR modularity level SHOULD allow enough independence of all modules so as if any module needs to be replaced, this has no consequences to the other modules.	✓	✓	✓	✓
MAD_S12	The three modules part of MAD run independently on each other consuming data from a common Kafka topic.				
R2.7.5	PALANTIR SHOULD reuse existing open- source software and tools, where it is appropriate and possible according to the license.	✓	✓	✓	✓
MAD_S13	The three modules part of MAD are all based on open-source software.				
R2.7.6	The architecture of PALANTIR MUST be open, extensible, providing ability to add new functional components.	✓	✓	✓	✓
MAD_S14	MAD design allows the addition of further AD modules to complement MAD detection performance.				

3.3. Threat Classification and Alarm Management

Table 3 Specifications of the TCAM subcomponent

Req. ID	**Requirement description**	**Random Forest**
R1.5.3	The platform SHALL be able to automatically classify the type of anomaly/threat and to share the intelligence information in a standard format.	✓*
TCAM_S1	TCAM implements automated threat classification relying on supervised learning (Random Forest). In this first release, threat findings are provided in JSON format,	

	attaching the class label and classification confidence score to the existing data schema. Threat sharing functionalities that are leveraging standardised threat representation (STIX/TAXII) are planned for the second release of the Hybrid Threat Intelligence framework (D5.2).	
R1.5.5	The platform SHOULD provide analytics able to detect the most common threat types (malware, MitM, volumetric attacks).	✓
TCAM_S2	TCAM complements the threat detection capabilities of MAD by predicting the class of discovered anomalous activity. Since TCAM relies on supervised learning, it can be trained to identify the threat label of any attack for which a labelled training dataset exists.	
R1.5.6	The platform SHOULD provide analytics able to detect phishing attacks.	--
TCAM_S3	Not currently addressed due to lack of training data; planned for second TCAM release.	
R1.5.8	The platform SHALL provide periodic retrain functionalities for its analytics components (e.g. on a monthly basis).	✓*
TCAM_S4	The first TCAM release focused on the inference phase (classification of detected anomalies). Both the design of the subcomponent (selected algorithms) and its implementation details (selected frameworks) support periodic retrain functionalities, which are planned for the second release (D5.2).	
R2.1.1	The analytics of the platform SHOULD be able to scale with respect to the number of data sources, the volume and the velocity of data streams.	✓
TCAM_S5	The implementation of the TCAM subcomponent relies on the most widely used, open-source processing engine for big data, thus ensuring horizontal and vertical scalability by design.	
R2.1.2	The analytics components of the platform SHOULD be able to deal with the computational and memory limitations posed by large datasets.	✓
TCAM_S6	The technical implementation of TCAM is based on a distributed machine learning framework that is not limited to the computational or memory constraints of a single machine.	
R2.2.1	PALANTIR deploys various big data analytics frameworks that have demands in computational power. They MUST be regularly evaluated during development, such that they are shown to be accurate with real-time data.	✓
TCAM_S7	The current TCAM release implements feature engineering functionalities using rolling window statistics on streaming (timeseries) data to improve model accuracy. The preliminary evaluation on benchmark datasets (Section 4) confirms accurate inference on real-time data.	
R2.2.2	PALANTIR SHOULD outperform existing conventional methods from potential competitors.	--

TCAM_S8	Fulfilment of this requirement premises an integrated PALANTIR platform. It will therefore be addressed in the final release of the Hybrid Threat Intelligence framework (D5.2).	
R2.2.3	The time to discover critical info & alerts in the security dashboard MUST NOT exceed 1 minute.	✓
TCAM_S9	The inference time of the currently developed TCAM module measured on regular data streams of benchmark datasets does not exceed 10 seconds.	
R2.2.7	The platform MUST showcase a reduction of false positives and negatives of at least 15% compared to commercial solutions.	--
TCAM_S10	Fulfilment of this requirement premises an integrated PALANTIR platform. It will therefore be addressed in the final release of the Hybrid Threat Intelligence framework (D5.2).	
R2.6.2	The PALANTIR modularity level SHOULD allow enough independence of all modules so as if any module needs to be replaced, this has no consequences to the other modules.	✓
TCAM_S11	Additional threat classification modules can be added to TCAM in order to improve threat detection accuracy, with no consequences to the operational lifecycle of the existing ones. The current module (Random Forest) is itself an ensemble of several independent predictors (decision tree classifiers).	
R2.7.5	PALANTIR SHOULD reuse existing open-source software and tools, where it is appropriate and possible according to the licence.	✓
TCAM_S12	TCAM relies solely on open-source software and distributed machine learning frameworks.	
R2.7.6	The architecture of PALANTIR MUST be open, extensible, providing the ability to add new functional components.	✓
TCAM_S13	TCAM supports the addition of complementary ML-based detection modules that focus on the analysis of different data modalities (e.g., netflow, syslog).	

3.4. Recommendation and Remediation

Table 4 Specifications of the RR subcomponent

Req. ID	**Requirement description**	**RR subcomponent**
R1.1.3	The platform MUST provide near-real-time (NRT) data processing functionalities.	✓*

RR_S1	Remediating threat scenarios is a crucial task. Therefore, the RR tool needs to be fast in identifying the recipes to choose from and determining the actions that need to be enforced in order to mitigate the risks posed by the identified threats. However, Real-Time reactions are not needed also because this tool requires the administrators in the loop to make decisions and confirm choices. The current evaluation did not highlight potential issues in reaching NRT data processing.	
R1.3.29	The platform SHOULD prevent and react against Ransomware attacks	--
RR_S1	The RR tool has only been tested against the botnet scenario proposed by the Use Cases. Ransomware is definitely a case the RR tool will deal with in a future version.	
R1.5.4	The platform SHALL be able to analyse an attack report to produce an ordered set of suggested actions (e.g., VNFs configuration) to mitigate the attack	✓
RR_S1	The RR tool has been designed explicitly to meet this goal. At the current stage of development, it already meets this requirement for selected threat scenarios and will cover more scenarios in the future versions.	

4. Implementation

This section covers the implementation details of the different subcomponents introduced in previous sections. The logic and technologies are all explained in the following subsections, mirroring the structure of Section 2. The code for each subcomponent for this first release of the Hybrid Threat Intelligence framework is available at the following repositories: DCP [9], MAD [10], TCAM [11], RR [12].

4.1. Distributed Collection and Data Preprocessing

4.1.1. Implementation details

The Distributed Collection and Data Preprocessing (DCP) subcomponent has been designed and developed in such way, that all of its modules can be deployed either dockerized (i.e., in a Kubernetes cluster) or non-dockerized. Figure 18shows the dockerized architecture of DCP subcomponent. Apache Kafka [13] has been selected as PALANTIR's message bus.

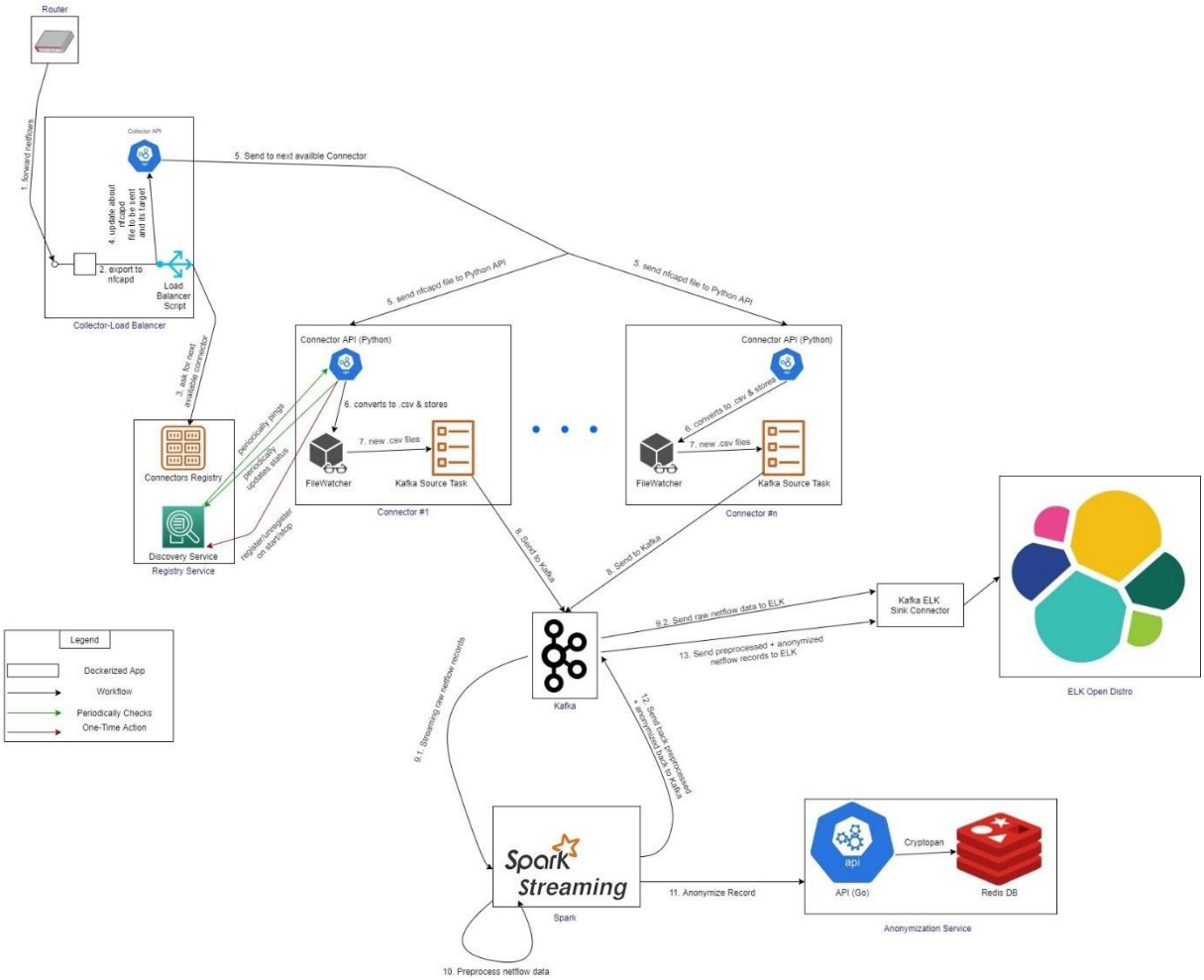

Figure 18: The entire pipeline schema for Distributed Collection and Data Preprocessing mechanism

The **Collector** module, which is responsible for collecting forwarded binary netflows, uses the nfcapd tool to capture these netflows and save them to nfcapd files. Also, a Python service is developed for sending the nfcapd files into the available connectors, which will ingest them to Kafka. This service makes an HTTP request in the registry module every time that a new nfcapd file is created. The Registry

will send a response with the next available connector. Finally, the Collector will try to send the nfcapd file in this connector along with its filename.

For the implementation of the **Registry** module, a Python service and a Python API have been developed. The developed service is a health check service. It pings all registered connectors to check if they are still online. If it gets an error response or no response at all from a connector, the registry removes this connector from the list of the available connectors. The provided API has been implemented using the Flask library[14], and is used by all the connectors to be registered, unregistered, or updated. Also, it is used by the collector when it searches for the next available connector.

Developed **Source Connectors** are submodules of the data collection module. They have been designed and implemented on top of the Kafka Connect API. Kafka Connect is an API, which is used to ingest data to Kafka from other sources, or to fetch data from Kafka and ingest them to other sources. It also supports distributed deployment of Kafka connectors. Source connectors are designed to ingest data from different sources into Kafka. The developed source connectors for netflow data consist of three different parts, which are shown in Figure 19. As depicted in it, the collector module looks for new nfcapd files and distributes them among all available netflow source connectors. If the source connectors are running dockerized or if they are running in separate machines (either physical or virtual), the netflows will not be forwarded to a separate folder as shown in the image, but they will be sent over HTTP using their connectors' API. If all connectors run in the same machine, the collector module can distribute the collected nfcapd files in separate directories (one for each connector).

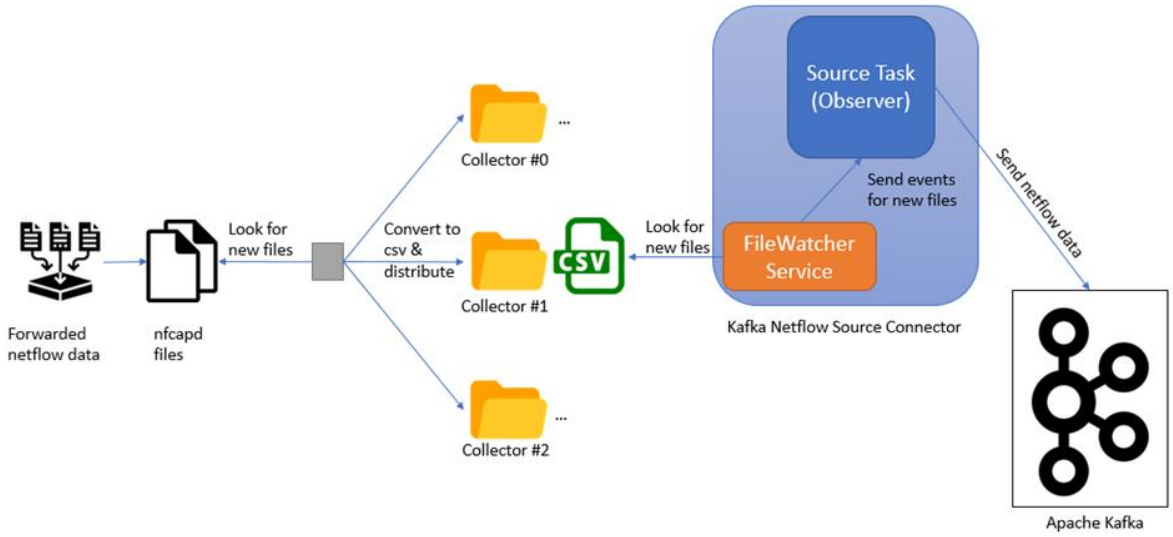

Figure 19: Architecture of designed source connectors for netflow data

The first part is an API, developed using Python and Flask, which can be used for communication with the registry and for receiving files from the Collector module. The second is a service, named FileWatcher Service, which monitors a directory for new CSV files with netflow records (a similar service is deployed for syslog data). FileWatcher emits events every time a new CSV file is detected, so that all registered listeners know about this. Finally, there is the SourceTask which is responsible for ingesting the csv files into Kafka. It reads them line by line and creates a Kafka record for each line. SourceTask knows about the new CSV files because it is a registered listener in FileWatcher's events. Both FileWatcher Service and Source Task have been implemented in Java following the observer pattern. SourceTask also follows the Kafka Connect architecture implementing the provided API. As it is shown in the figure above, the observer in this case is SourceTask.

Sink Connectors are also submodules of the data collection module. They have also designed and implemented using Kafka Connect API. These connectors had also been developed in the Java programming language. This type of connectors is used to fetch data from Kafka and forward it to another destination. For storage of Data Collection and Preprocessing, Elasticsearch [15] has been

selected. The developed sink connectors retrieve netflow records as Kafka messages, converts them to appropriate JSON format, and finally, they send them to deployed Elasticsearch.

The **Data anonymization** module consists of three submodules.

Firstly, the application service which is anonymizing/ de-anonymizing IP addresses. This application has two core functions. As mentioned before, it is both **anonymizing** an IP address, given an original IP address to return its obfuscated version and **de-anonymizing** an IP address, given an obfuscated IP address to return its original version. Furthermore, the application features the following elements:

- Usage of Crypto-PAn algorithm for anonymization which anonymizes IP address keeping subnet structure.

- Implementation of a REST-ful anonymization service.

- (De-)/Anonymize IP addresses making POST requests.
- Usage of Go programming language, due to its good concurrency, using goroutines.

- Measurement of the delay time. It measures the time needed until the anonymization function ends. It starts measuring the time, when the POST HTTP request is received from Spark (data preprocessing module) until the anonymization and the insertion of the obfuscated IP have been completed. The time elapsed is also, included in the response of the API, along with the original and the obfuscated IP addresses.

- A Redis database where the data is stored. The usage of the Redis database is to keep a mapping between original & obfuscated IP addresses. The advantages of this selection are that it runs in-memory, which is extremely fast, and it persists the data. The last one is the Redis Insight web application. This GUI provides an intuitive Redis admin GUI and helps optimize the usage of Redis in our application.

The **Data preprocessing** module is responsible for fetching raw data from Kafka, preprocessing them with some defined functions and then sending the preprocessed data back to Kafka. This module and its preprocessing functions have been developed as Spark [16] functions. The communication with Kafka for both reading and writing messages has been achieved using Spark Streaming and, more specifically Structured Streaming. All Spark preprocessing functions have been written in Scala.

For the netflow case, data anonymization can also be considered as a preprocessing function, but it takes place in another module, the data anonymization module. However, the data preprocessing module is the one that initiates the anonymization function and waits for its result using HTTP requests. For this reason, a list with all IP addresses that need to be anonymized must be provided during the startup of this module. IP addresses in this list can be included using three different ways: **i)** specific IP addresses can be provided, **ii)** a subnet (with its network mask) can be defined and **iii)** ranges of IP addresses can also be included (only ranges with subnet mask /24 can be recognized). The results of this module are three different preprocessed outputs, that will be pushed in three separate Kafka topics. The first one includes the preprocessed data without applying anonymization function. The second output includes the raw data after the application of the anonymization function. Finally, the third output contains the preprocessed data with anonymization function applied to them. The sink connectors of data collection module will consume the third output, in order to ingest them to Elasticsearch. For the syslog case, the log parsing and TF-IDF operations mentioned in Section 2.3.1 are written in Python using the PySpark library.

The **Data storage layer** is used to store not only the raw netflow data but also the preprocessed ones. It is deployed to an OpenDistro for ElasticSearch server among with a core app of the suite, Kibana. Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows storing, searching, and analyzing big volumes of data quickly and in near real time. Moreover, Kibana is utilized to visualize the data from Elasticsearch data and to navigate the Elastic Stack.

4.1.2. Preliminary Evaluation

A number of preliminary benchmarks have been run in order to evaluate the performance of the developed DCP modules. A benchmark mode has been designed for each module. When this mode is enabled, these modules will store relevant information and timestamps about the retrieved netflows in a

text file during each phase of the ingestion and preprocessing pipeline. More specifically, two different benchmarks are designed, one that includes preprocessing & anonymization of the netflow data and a second one that does not include preprocessing or anonymization.

In order to run the aforementioned benchmarks, one Kafka broker has been used, meaning that all topics will have only one partition, a Spark deployed in Kubernetes cluster with one executor for the data preprocessing module, one instance of netflow source connector and netflow sink connector. The Intrusion Detection Evaluation Dataset (CIC-IDS2017) [17] was selected to simulate one hour of ingestion of netflows from Day 1 (Monday). Collected netflows will be dumped in nfcapd files every 5 minutes. In total, 535.989 netflow records have been ingested into the pipeline for each benchmark in a time interval of 60 minutes.

The first benchmark uses all modules of the ingestion pipeline. The raw netflows are collected in nfcapd files and are forwarded to netflow source connectors. The connectors extract the netflows and ingest them to Kafka as separate messages. The preprocessing module then fetches the data, applies the defined preprocessing and anonymization functions and writes the results back to Kafka. Finally, the preprocessed & anonymized netflows are retrieved from Kafka, they are transformed and finally pushed to Elasticsearch. The flow of this benchmark, along with the separate points where measurements have been taken, is shown in Figure 20.

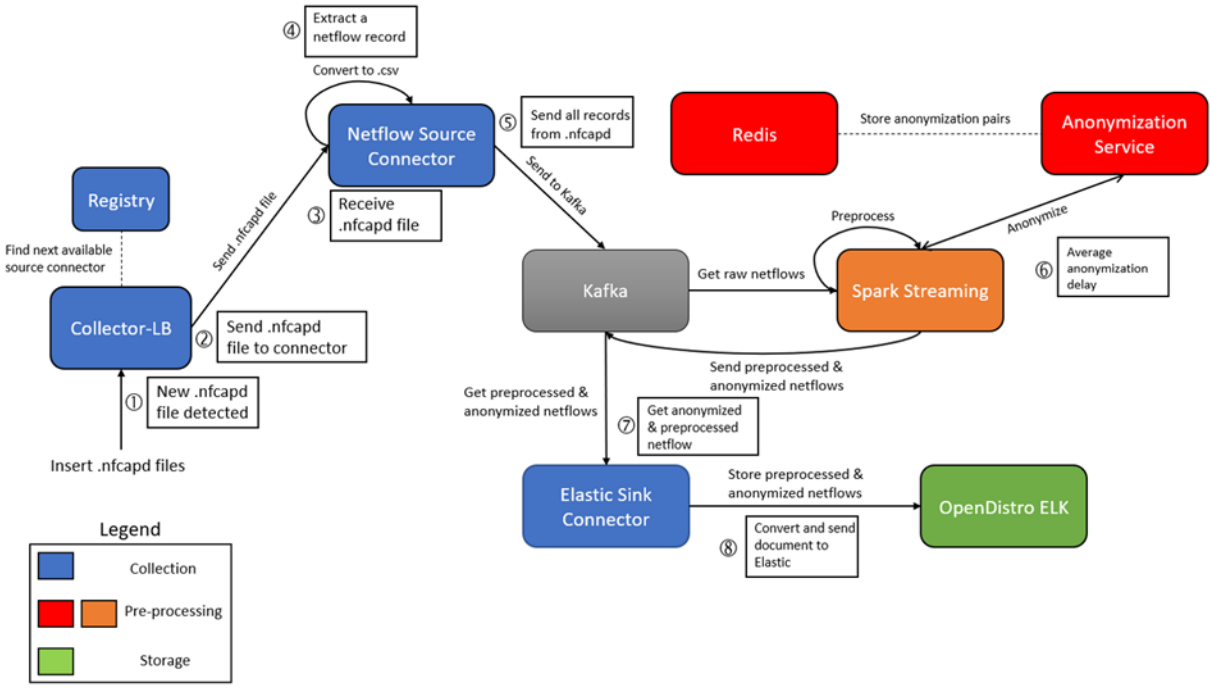

Figure 20: Benchmark with preprocessing flow

For the benchmark that does not involve any preprocessing or anonymization functions, the average end-to-end delay for a netflow record is 61 seconds. We are also reporting the minimum and the maximum time needed in order to ingest a netflow end-to-end, which is 5 seconds and 191 seconds respectively. Finally, we have measured the average time needed from the Crypto-PAn algorithm of the data anonymization module to anonymize the given IP addresses. The average anonymization time needed for this algorithm to achieve its goal is 1.56 ms.

An additional benchmark is leveraged to calculate the ingestion time of raw netflows from the time they are collected until the time they are sent to the storage module. In this benchmark no preprocessing or anonymization function have been applied. The flow of this benchmark and the capture points are shown in Figure 21.

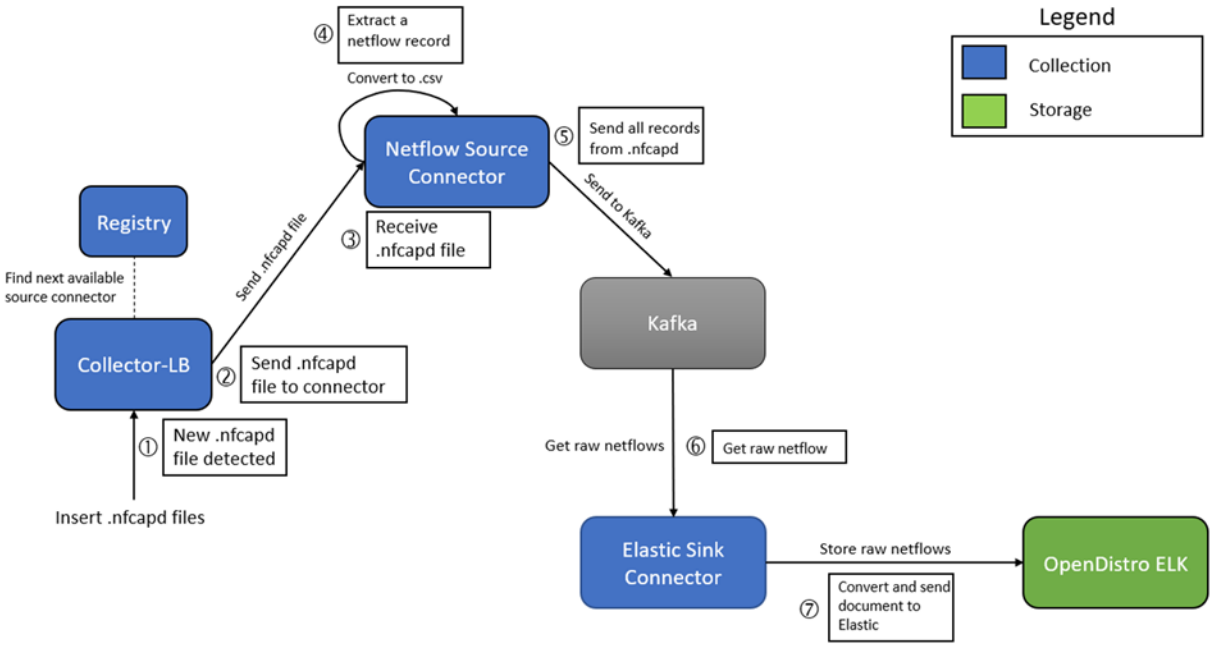

Figure 21: Benchmark without preprocessing flow

For this benchmark, where no preprocessing or anonymization functions are applied, the average end-to-end delay for a netflow record is 57 seconds. Also, the minimum time needed in order to ingest a netflow end-to-end is 4 seconds and the maximum time is 183 seconds.

In order to further improve the performance of the developed modules, additional benchmarks are planned using more instances of the developed modules in order to take advantage of their scalability and the distributed performance features.

4.2. Multimodal Anomaly Detection

4.2.1. Implementation details

MIDAS

The MIDAS anomaly detection module presented in Section 2.3.2 has been implemented starting from an open-source version of the algorithm [18]. The implementation has been modified to make it work in a streaming fashion by directly consuming events from a Kafka topic rather than reading in one batch the whole dataset. The Kafka topic from which events are read is *netflow-anonymized-preprocessed*. In this way, NetFlow events pre-processed and anonymized by the DCP subcomponents are fed into the MIDAS module. Input events are further processed to extract relevant information such as the endpoints of the connection and the timing information. The implementation has been ported to a Docker [19] container which has been tested in the PALANTIR testbed in conjunction with the DCP subcomponent to verify part of the TI pipeline from the ingestion of raw binary NetFlow data up to the anomaly detection. The container is based on python:3.8 image [20] part of Docker Official Images and requires the following Python modules: kafka-python [21], numpy [22] and numba [23] which are all automatically installed through pip [24] Python package installer during container build.

The anomaly detection algorithm requires defining a threshold against which the anomaly score is compared to declare whether a network connection is abnormal or not. We configured the threshold as the 99th percentile of the anomaly score values observed during a training period which is assumed to be free from anomalies. In the following subsection 4.2.2, we report a preliminary performance evaluation using a benchmark dataset. The threshold can be adjusted periodically and further increased/decreased to better tune the tradeoff among false positives (normal flows marked as anomalies) and false negatives (anomalous flows not marked as anomalies).

Benchmark datasets

For the network traffic analytics pipeline, the benchmark USTC-TFC2016 dataset [25] was used to train the machine learning algorithms. The dataset consists of two parts: Part I contains ten types of malware traffic from public websites, which were collected from real network environments and Part II contains ten types of normal traffic which were collected using traffic simulation software that is resembling the most common network applications.

For the syslog analytics pipeline, an open-source benchmark dataset (AIT Log Data Set V1.1 [26]) was utilized to train anomaly detection and threat classification models. The dataset contains synthetic log data suitable for the evaluation of intrusion detection systems collected from four independent testbeds. For the purposes of PALANTIR, a subset of sequential execution of multi-step attacks with sequential execution of the following attacks launched against web servers (e.g., network mapping, brute-force attacks and vulnerability scans using hydra and nikto tools, web shell attacks, etc.) was selected.

Additionally, prior to training, the TF-IDF vectorizer of the scikit-learn Python library [27] was used to convert textual (system) logs to their numerical representations. Stop-word removal and lower-case conversion was applied to the logs, while dimensionality reduction of the 145 features ordered by term frequency across the corpus was necessary to create a denser representation of the processed logs.

Autoencoder

The prototype versions of Autoencoder variants trained under the scope of MAD module use the Keras framework of TensorFlow on Python [28]. The common model specifications in both cases include the Leaky ReLU activation function [29], batch normalization using a batch size of 512, a latent vector size of 25% the original dimensions and the Adam optimizer [30]. Both models were trained for 60000 iterations. The following hyperparameters were specific to each model:

Autoencoder's learning rate is set equal to 0.002 and a Mean Squared Error loss is used. The Encoder consists of 3 layers having 128, 64 and 32 neurons respectively. The latent vector is of size 21 while the Decoder consists of 3 layers with 32, 64 and 128 neurons.

GANomaly

GANomaly's learning rate is set equal to 0.0002 for both the Generator and the Discriminator. The Encoders and Decoders have the same architecture as the ones used for the Autoencoder, described above. For the Discriminator labels, one-sided label smoothing of value 0.9 for the reconstructed inputs given by the output of the Generator is utilized, in order to prevent overconfidence of the Discriminator. However, a balancing scheme for training the Generator and Discriminator is not implemented and they are both trained equally.

Isolation Forest

MAD's IsolationForest algorithm is trained using the scikit-learn Python framework [27]. A hyperparameter search is leveraged to determine the optimal values for the decision trees comprising the forest (10 estimators) and the percentage of samples utilized by each tree (85% of the dataset). Each tree utilizes the full set of the available features.

4.2.2. Preliminary Evaluation

MIDAS

The performance of MIDAS has been evaluated using CIC-IDS2017 dataset [17], an Intrusion Detection benchmark dataset containing a mix of benign and common network attacks. The dataset includes traffic covering five consecutive days from Monday to Friday measured on a testbed infrastructure and includes 2.700.000 connections of which more than 500.000 are anomalous. Monday is the only day that contains normal traffic, while the remaining days contain both normal traffic and malign traffic including the following types of attack: Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS.

MIDAS' internal parameters (e.g., the size of the CountMinSketch (CMS) data structures and the temporal decay factors) have been set to the same values reported in the original paper, i.e., using CMS

with 2 hash functions and 1024 buckets, a temporal decay factor α=0.5 and a conditional merge threshold ε=1000. The size of MIDAS time slot has been set to 1 minute because the dataset does not provide a finer time granularity. We fed traffic data from Tuesday to Friday into MIDAS and computed the anomaly score of all the flows. The average processing rate is 222k flows/sec. By using scikit-learn Python library we computed the ROC-AUC=0.9925 and average Precision=0.9845. The Receiver Operating Characteristic (ROC) curve captures the TPR-FPR (the definition of TPR and FPR is reported below) tradeoff at different classification thresholds. ROC-AUC is the area under the ROC curve and provides an aggregate measure to quantify the performance of a classification model across all the classification thresholds. The Precision quantifies the quota of positive class predictions that actually belong to the positive class (i.e., quota of reported anomalies that are actually anomalous flows).

These two metrics are threshold-independent, and in order to obtain a concrete implementation and to investigate the misclassified flows, a specific value of the anomaly detection threshold must be defined. We computed two possible values based on the 99th and 99.999th percentile of the anomaly scores resulting from feeding into MIDAS the traffic data from Monday (which is known to be free from anomalies). The following Figure 22 reports the anomaly score of all the flows (split into train and test set) and the two thresholds computed on the train set portion as two horizontal lines.

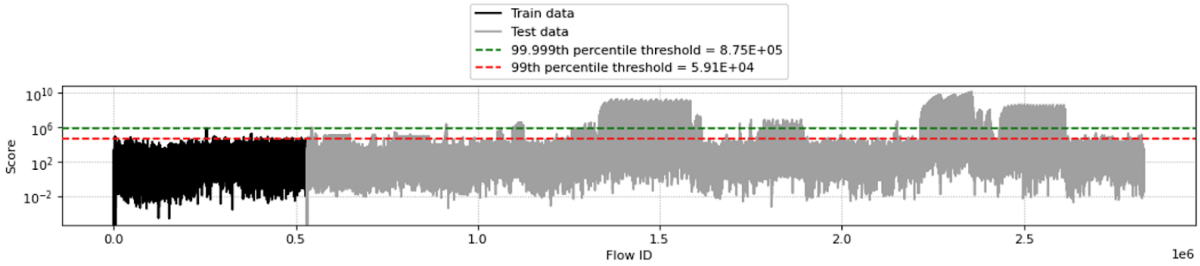

Figure 22: Anomaly scores of individual flows in CIC-IDS2017 dataset computed by MIDAS

For each one of the two thresholds, the following table reports additional performance metrics, namely:

- True Positives (TP): number of positive samples (i.e. anomalies) correctly detected as anomalies
- False Positives (FP): number of negative samples (i.e. normal flows) wrongly detected as anomalies
- True Negatives (TN): number of negative samples (i.e. normal flows) correctly marked as not anomalous
- False Negatives (FN): number of positive samples (i.e. anomalies) wrongly marked as not anomalous
- Accuracy: computed as (TP+TN)/(TP+TN+FP+FN), it quantifies the percentage of correct predictions
- Precision: computed as TP/(TP+FP), it quantifies the quota of positive class predictions that actually belong to the positive class
- Recall or True Positve Rate (TPR): computed as TP/(TP+FN), it quantifies the quota of positive samples that are correctly predicted as positive
- False Positive Rate (FPR): computed as FP/(FP+TN), it quantifies the quota of positive samples that are wrongly predicted as negative
- F1-score: computed as 2TP/(2TP+FP+FN), it is the harmonic mean of Precision and Recall

Table 5: MIDAS performance on CIC-IDS2017 for a couple of threshold values

99-th threshold = $5.91*10^4$	99.999-th threshold = $8.75*10^5$
TP = 552077 FP = 170806 FN = 1572373 TN = 5569	TP = 534348 FP = 86611 TN = 1656568 FN = 23298
Accuracy = 0.9233	Accuracy = 0.9522

Precision = 0.7637 Recall [TPR] = 0.9900 FPR = 0.0980 F1-score = 0.8623	Precision = 0.8605 Recall [TPR] = 0.9582 FPR = 0.0497 F1-score = 0.9067

As expected, a higher value of the threshold (i.e., $8.75*10^5$) provides a much lower number of FP at the cost of a reduced number of TP. Even if MIDAS is part of MAD and does not consider the specific labels of the anomalous flows, given that CIC-IDS2017 is a labelled dataset, we can a posteriori investigate its performance split by the specific type of attacks. The following Table 4 confirms that MIDAS, thanks to its ability to consider both temporal and spatial relations across network flows, works particularly well for (D)DoS and Port Scan. "Missed quota" column reports the % of flows undetected for each attack type, i.e., the misclassified negative samples ("FN" column) over the total number of flows ("Tot flows" column). Attack types colored in green (yellow) refer to attacks for which no more than 10% (50%) of flows have been misclassified. Red ones are the ones for which more than half of the flows have been missed as anomalies. Again, considering a higher value of the threshold (moving from the left side of the table to the right one) on one side reduces the FP (as shown in the previous table), but on the other side, it also decreases the TP, i.e., the number of FN increases.

Table 6: MIDAS performance on CIC-IDS2017 split by type of attack

AD threshold = $5.91*10^4$					AD threshold = $8.75*10^5$			
	FN	Tot flows	Missed quota			FN	Tot flows	Missed quota
Web Attack – Brute Force	1507	1507	1.0000		FTP-Patator	7938	7938	1.0000
Web Attack – XSS	652	652	1.0000		SSH-Patator	5897	5897	1.0000
Web Attack – Sql Injection	21	21	1.0000		Bot	1966	1966	1.0000
Heartbleed	11	11	1.0000		Web Attack – Brute Force	1507	1507	1.0000
Bot	1961	1966	0.9974		Web Attack – XSS	652	652	1.0000
Infiltration	17	36	0.4722		Web Attack – Sql Injection	21	21	1.0000
SSH-Patator	245	5897	0.0415		Heartbleed	11	11	1.0000
FTP-Patator	123	7938	0.0154		Infiltration	33	36	0.9166
DoS slowloris	84	5796	0.0144		DoS slowloris	1430	5796	0.2467
DoS Slowhttptest	75	5499	0.0136		DoS Slowhttptest	1309	5499	0.2380
DoS GoldenEye	47	10293	0.0045		DoS GoldenEye	312	10293	0.0303
PortScan	662	158930	0.0041		PortScan	1524	158930	0.0095
DdoS	101	128027	0.0007		DdoS	388	128027	0.0030
DoS Hulk	63	231073	0.0002		DoS Hulk	310	231073	0.0013

The high number of FP can be reduced by further fine-tuning the MIDAS internal parameters. E.g., by doubling the number of buckets from 1024 to 2048, we obtained ROC-AUC=0.9933 and average Precision=0.9841. When using the two 99^{th} and 99.999^{th} percentiles to define a couple of threshold values, as reported in the table below, we observed a highly reduced FP (and thus FPR). The price to pay is a reduced average processing rate of 128k flows/sec and the doubling of the memory requirements.

Table 7: MIDAS performance on CIC-IDS2017 fine-tuning example

99-th threshold = $1.63*10^5$	99.999-th threshold = $2.12*10^7$
TP = 546945 FP = 136972 TN = 1606207 FN = 10701	TP = 513330 FP = 3886 TN = 1739293 FN = 44316
Accuracy = 0.9358 Precision = 0.7997 Recall [TPR] = 0.9808 FPR = 0.0786 F1-score = 0.8811	Accuracy = 0.9791 Precision = 0.9925 Recall [TPR] = 0.9205 FPR = 0.0022 F1-score = 0.9552

As a final remark, it should also be noted that the output of the MAD subcomponents still has to be processed by the following TCAM subcomponent whose task is further reducing the FP when trying to classify the specific type of threat associated with the anomalies.

Autoencoder

The anomaly detection capabilities of the Autoencoder variant for the network traffic case are illustrated below on the CIC-IDS2017 Bot test set for three different decision boundaries. It should be highlighted that these results are only relevant for the botnet attacks, which corresponds to one of the most complex ones in terms of detection. The decision boundaries were drawn by accepting various percentages of the normal data as false positives. All results represent the highest overall precision across the three decision boundaries.

Decision Boundary	Autoencoder Precision	FPs
0.001%	0.75	3
0.006%	0.54	13
0.0011%	0.44	16

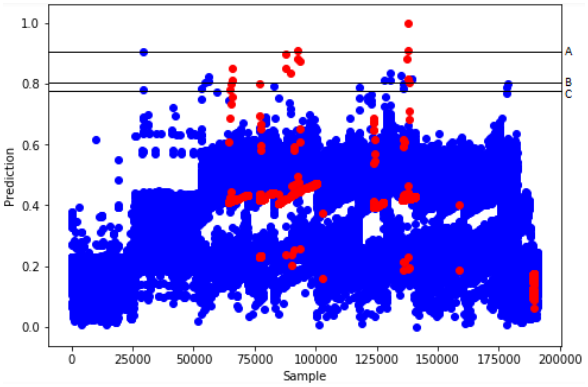

Figure 23: Results of the Autoencoder anomaly detection algorithm on the CIC-IDS2017 Bot test set

An interpretation of the above results is as follows: When accepting that 0.001% of normal traffic can be predicted as malicious, we get a decision boundary that contains that percentage of normal traffic on the wrong side of it. For that decision boundary, we get a precision of 0.75 from the Autoencoder, meaning 75% of the traffic labelled as malicious is indeed botnet traffic. We also present the underlying measurements of true positives and false positives for the same decision boundaries. While the Autoencoder has been able to detect some of the botnet occurrences, it cannot clearly separate between every instance of botnet and benign traffic.

GANomaly

Similarly, we have tested GANomaly on the same CIC-IDS2017 Bot test set for three different decision boundaries achieving significantly better results:

Decision Boundary	Autoencoder Precision	FPs
0.001%	0.92	11
0.006%	0.65	20
0.0011%	0.58	28

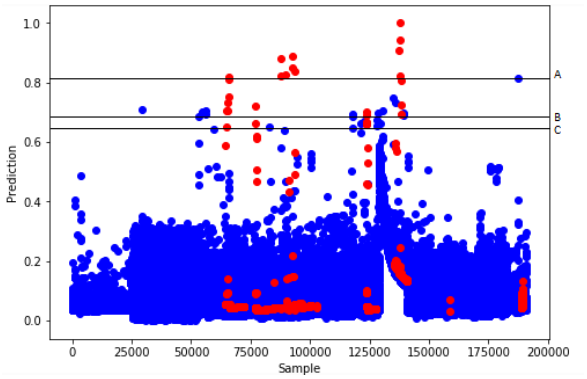

Figure 24: Results of the GANomaly anomaly detection algorithm on the CIC-IDS2017 Bot test set

As shown in Figure 24, both numerically and visually, GANomaly's performance surpasses that of the previous methods, being able to offer a precision-oriented solution for botnet detection purposes. It is apparent that GANomaly results in a more confident data distribution since the blue points, representing normal traffic, have an anomaly score that is on average closer to zero compared to their reconstruction error when passed through the Autoencoder. It can also be observed that the red points, representing malicious traffic, achieve better separation from the blue points with GANomaly, which is caused by a better modelling of the data distribution, making outliers stand out more.

The anomaly detection capabilities of the GANomaly variant for the network traffic case are also illustrated in Figure 25 using the USTC-TFC2016 test set. It can be easily observed that a clear separability between the benign and botnet classes exists, as a result of the difference between the produced reconstruction between the two different types of Netflow logs. This indicates that the model has learned the underlying patterns of normal traffic and is capable of reconstructing it, contrary to malicious (botnet) logs for which the model has no knowledge of, and therefore cannot recreate.

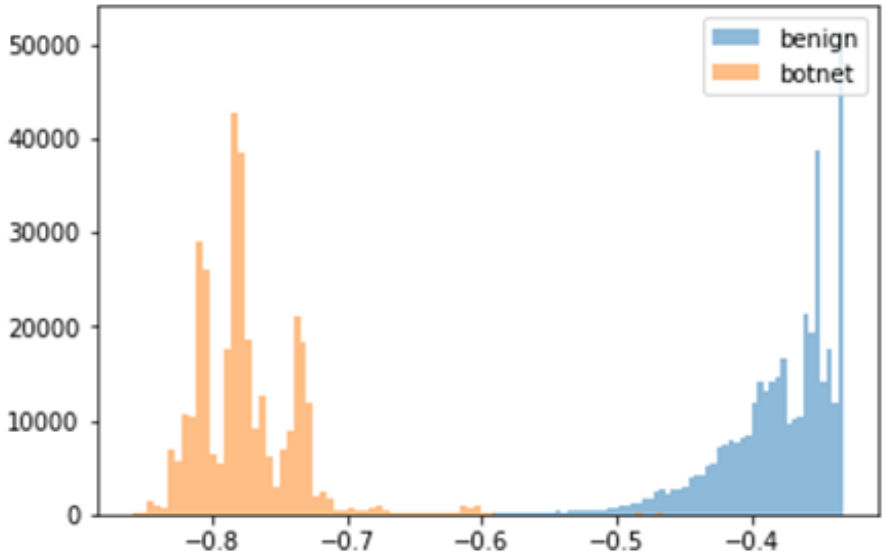

Figure 25: Results of the GANomaly anomaly detection algorithm on the USTC-TFC2016 test set

Isolation Forest on the syslog case

Similar to the network traffic case, the goal with regard to the system logs is to distinguish between benign and malicious behaviour. To this end, the separability between normal/benign and suspicious/botnet system logs is depicted in Figure 26 (for IsolationForest, highlighting the efficiency

of the aforementioned preprocessing and analytics operations). The figure is based on the AIT Log Data test set.

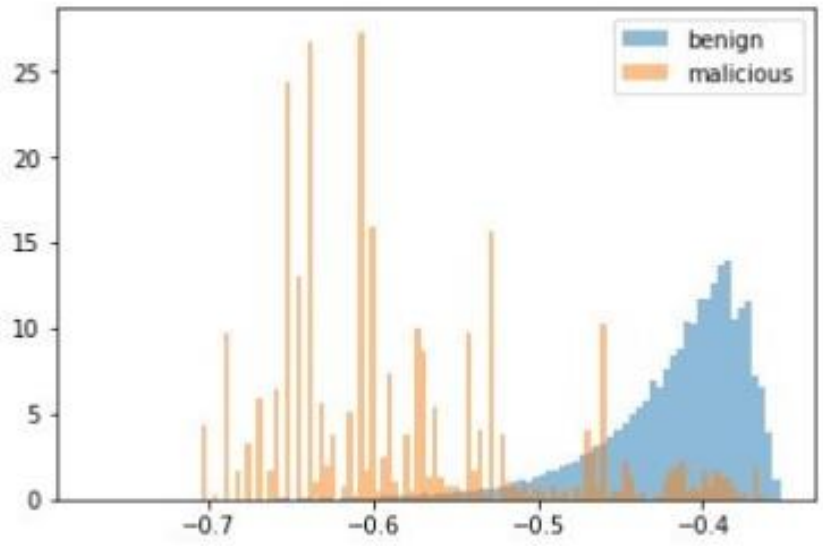

Figure 26: Results of the Isolation Forest anomaly detection algorithm on the AIT Log Data test set

4.3. Threat Classification and Alarm Management

4.3.1. Implementation details

For the purposes of the TCAM module, RandomForest classifiers were trained for both network traffic and system log analytics pipelines. RandomForest classifiers are trained using the scikit-learn Python framework [27].

Regarding the former case, a hyperparameter search is leveraged to determine the optimal values for the decision trees comprising the forest (161 estimators) and the minimum number of features considered by each tree when splitting a node (9 samples). A random subset with the size of the square root of the available features is used for each tree classifier, while the number of samples used to fit each decision tree is set as half of the available data to avoid overfitting.

Similar to the previous case, a hyperparameter search was leveraged to determine the optimal values for the decision trees comprising the forest (17 estimators), the minimum number of features considered by each tree when splitting a node (443 samples) and the minimum number of samples required per leaf (12 samples) to increase regularisation. A random subset with the size of the square root of the available features was used for each tree classifier, while the number of samples used to fit each decision tree was set as half of the available data.

4.3.1. Preliminary Evaluation

The classification results using RandomForest on the USTC-TFC2016 test set along with the importance of each feature involved in the classification process, are depicted in Figure 27. It is evident that the algorithm is capable of distinguishing between the different attack and benign traffic classes with high confidence.

```
acc: 0.993877719864317
label 0 acc: 0.9997424406325658
label 2 acc: 0.9772975332896747
label 5 acc: 0.9656992084432717
label 3 acc: 0.9969066849974222
label 9 acc: 0.9993464052287582
label 8 acc: 1.0
label 7 acc: 0.9981120201384519
label 10 acc: 0.9854781054512958
label 1 acc: 0.9957446808510638
label 4 acc: 0.9683098591549296
label 6 acc: 0.9773082942097027
```

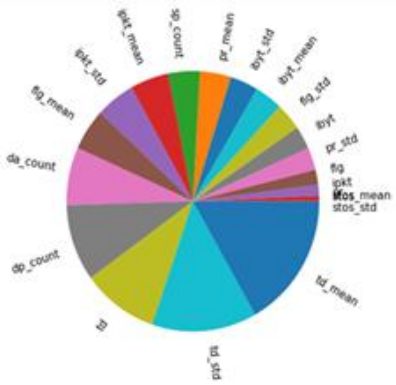

Figure 27: Classification accuracy and feature importance of RandomForest on USTC-TFC2016 test set for the netflow case

In a similar fashion to the netflow case, a benchmark was designed for the syslog case based on the test set of the AIT Log data. Given that the dataset is highly imbalanced (i.e., not all classes are equally represented) the test set only contained 3 types of labels: Label_0 which corresponds to benign logs, Label_1 and Label_2 that refer to hydra-ssh and nikto attacks respectively.

Label	Classification Accuracy
Label_0 (benign traffic)	0.9998
Label_1 (hydra_ssh)	0.9399
Label_2 (nikto scan)	0.8943
Overall test set	0.9933

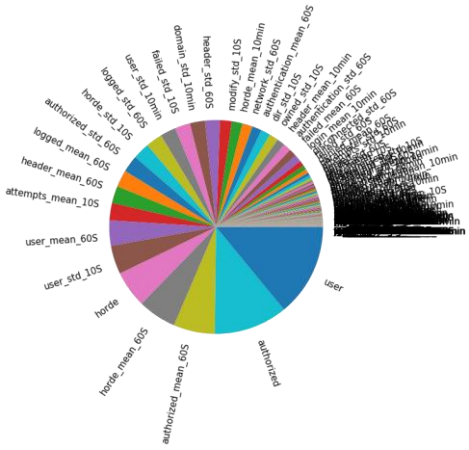

Figure 28: Classification accuracy and feature importance of RandomForest on AIT Log data test set for the syslog case

It should be noted that -after the successful training and testing of the aforementioned prototypes- production-level versions of the aforementioned anomaly detection and threat classification models were developed using the Spark v3.2 distributed ML framework [31] using the same hyperparameters.

4.4. Recommendation and Remediation

4.4.1. Implementation details

The Recommendation and Remediation tool presented in Section 2.3.4 has been developed as a set of Python 3.11 scripts. It has been designed in a modular fashion to ease the development of new features and the integration of new Enrichment Modules. Moreover, it has been designed to be called with an API and to be easily exported as a service.

More details on the current implementation of its modules are described below.

- *Input Analyser*. This module uses standard Python libraries to read, parse and store into the KB the TCAM outputs.
- *Recipe Filter*. This module uses standard Python libraries to query the RR Recipe DB. Both the Applicability report and the Applicable Recipes List are presented to the user as files.

- *Knowledge Base*. This database is implemented as a Python dictionary in the current version. Persistence is achieved by storing the dictionary as a JSON file. Even if it is unnecessary for this version of the prototype, the following versions of the KB will use more structured databases. We are investigating the use of JSON document databases (e.g., MongoDB).
- *RR Recipe Database*. This database contains a set of abstract recipes, i.e., sequences of remediation actions, written in a high-level custom language. It is implemented as a Python dictionary in the current version. Persistence is achieved by storing the dictionary as a JSON file. More structured database technologies are under investigation (e.g., traditional SQL, noSQL, and JSON document databases), but they have been evaluated as superfluous in the current stage and for the near future.
- *Recipe Instruction Interpreter*. This module works as an interpreter of the recipe descriptions. The interpreter is based on the Natural Language ToolKit (nltk) Python package [32], one of the most used frameworks to handle human language data. Furthermore, its functionalities are extended by means of Enrichment Modules. The RII Enrichment Modules currently available are described below.
    - The Landscape Analysis uses the igraph Python package [33], one of the most widely employed packages, for managing the graph that describes the network topology.
    - On the other hand, the Capability Management Module just uses standard Python libraries.
- *Output Generator*. This module reads the KB and produces the RR tool outputs. It does not employ any external Python package.

The Recommendation and Remediation tool is provided as a Docker container, based on the python:3.11-rc-bullseye official Docker image [20].

4.4.2. Preliminary Evaluation

We have evaluated the RR tool on the botnet use case, the reference scenario for the WP5. For this purpose, we have designed

- a target network and described it with the Landscape Description Language;
- a sample Threat Intelligence Report;
- a set of 7 recipes that have been stored in the RR Recipe DB.

The preliminary evaluation of the RR tool indicates that it does not pose any performance constraint. Also, a preliminary analysis of the complexity of the algorithms only indicates a risk for the computation of the network paths, which is not considered a significant issue for networks with less than a billion nodes.

Further performance evaluations are expected in the coming months with larger synthetic networks (to evaluate the scalability against the landscape complexity and size) and potentially more sophisticated recipes (which could be developed to address other project use cases).

Finally, a more thorough validation is expected from the use case owners. They will be asked to

- validate the appropriateness and effectiveness of the recipes proposed for all the use cases;
- identify the missing reactions and reaction strategies that may be the starting point for defining new recipes;
- the correctness of the deployment of the proposed recipes, both in terms of changes imposed to the target network and changes to the configurations of the security capabilities involved.

5. Conclusions

This deliverable provided the initial detailed view on the architecture, low-level design and specifications of the subcomponents belonging to the Hybrid Threat Intelligence framework, constituting the core line of work in WP5.

The information provided is the current, up-to-date architecture and design for each subcomponent and corresponding module(s). This design encompasses everything from the internal separation of logic concerns to the interactions between modules, as well as the structure of the top-level components of the PALANTIR platform. All of these specifications and designs create the path for each of them to be developed. The work of this deliverable will have an influence on future talks about integrations with other components.

Aside from the design, this deliverable contains low-level, development-related details in the form of technical specifications (mapped from the general requirements introduced in D2.1), that can be used as a guideline to monitor the progress of the technical activities of WP5. Finally, this deliverable covers the implementation details for each subcomponent, showcasing the selection of technologies, open-source tools and frameworks and illustrating their intended functionality with the help of preliminary evaluations.

In our efforts to provide a richer representation of the threat landscape that closely resembles human decision making, we envision the analysis of multimodal data types as a core functionality of the final PALANTIR release. Following up on the advances made by WP5 partners (UPM, TID) in other Horizon 2020 projects (i.e., 5GROWTH [34] and 5G-CLARITY [35]), there are plans to introduce additional data aggregation features to the Threat Intelligence component, namely the Semantic Data Aggregator (SDA). The SDA is a semantic, model-driven monitoring framework that enables data collection, data transformation, and data aggregation from different monitoring elements, and coordinates the flow of these data among a set of heterogenous data sources and data consumers. By using formal data models, defined by means of the YANG[36] modelling language, the SDA adapts data collected from the available sources into the formats suitable for consumers. In the scope of WP5, the SDA is envisioned as an element extending the capabilities of the DCP subcomponent within the Hybrid Threat Intelligence framework. In this regard, the definition of a YANG model for NetFlow-based monitoring data has started. By applying a formal model that is agnostic to the data source, i.e., the NetFlow collector, the SDA provides interoperability to current and prospective consumers present within PALANTIR framework, especially the MAD subcomponent of the Hybrid Threat Intelligence framework. Nevertheless, focus will not limit to NetFlow as more sources and consumers, such as IPFIX, syslog or network telemetry mechanisms, are in the radar.

As the project enters its second phase, the next steps will mostly focus on integration activities with the rest of the PALANTIR components and on adapting new functionalities (e.g., SDA), while fulfilling any requirements that have not yet been met in the implementation phase. Such examples are: the aggregation of threat findings from complementary analytics-based threat detection modules, the finetuning and retraining of ML/DL models to address additional attack types relevant to the established Use Cases, the implementation of standardised alerts as part of the alarm management functionalities to enable live threat sharing, and the development of additional mitigation policies to make full use of the currently available or planned SCs. To this end, the existing code in the repositories is expected to be updated and improved upon in the months to come.

6. References

[1] Z. S. Harris, "Distributional Structure," WORD, 1954, doi: 10.1080/00437956.1954.11659520.

[2] S. Bhatia, B. Hooi, M. Yoon, K. Shin, and C. Faloutsos, "Midas: Microcluster-based detector of anomalies in edge streams," 2020, doi: 10.1609/aaai.v34i04.5724.

[3] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation forest," 2008, doi: 10.1109/ICDM.2008.17.

[4] O. Kompougias *et al.*, "{IoT} Botnet Detection on Flow Data using Autoencoders," 2021.

[5] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "GANomaly: Semi-supervised Anomaly Detection via Adversarial Training," 2019, doi: 10.1007/978-3-030-20893-6_39.

[6] H. Attak *et al.*, "Application of distributed computing and machine learning technologies to cybersecurity Application of distributed computing and machine learning technologies to cybersecurity," *Comput. Electron. Secur. Appl. Rendez-vous (C&ESAR), 19-21 Novemb. 2018*, 2018.

[7] P. A. A. Resende and A. C. Drummond, "A survey of random forest based methods for intrusion detection systems," *ACM Computing Surveys*. 2018, doi: 10.1145/3178582.

[8] "SECURED H2020 Project." https://cordis.europa.eu/project/id/611458.

[9] "Distributed Collection and Data Preprocessing (DCP) Source Code." https://github.com/palantir-h2020/ti-dp.

[10] "Multimodal Anomaly Detection (MAD) Source Code." https://github.com/palantir-h2020/ti-mmml-ad.

[11] "Threat Classification and Alarm Management (TCAM) Source Code." https://github.com/palantir-h2020/ti-mmml-tc.

[12] "Recommendation and Remediation (RR) Source Code." https://github.com/palantir-h2020/ti-re.

[13] "Apache Kafka." https://kafka.apache.org/.

[14] W. framework Flask, "Flask Web Framework," *Flask*. 2018.

[15] "Elastic." https://www.elastic.co/.

[16] "APACHE Spark." https://spark.apache.org/.

[17] "Intrusion Detection Evaluation Dataset." https://www.unb.ca/cic/datasets/ids-2017.html.

[18] "MIDAS repository." https://github.com/Stream-AD/MIDAS.

[19] "docker." https://www.docker.com/.

[20] "docker for python." https://hub.docker.com/_/python.

[21] "kafka for python." https://pypi.org/project/kafka-python/.

[22] "numpy." https://pypi.org/project/numpy/.

[23] "numba." https://pypi.org/project/numba/.

[24] "pip - Python package installer." https://pip.pypa.io/en/stable/.

[25] "USTC-TFC2016 dataset." https://github.com/yungshenglu/USTC-TFC2016.

[26] L. Max, S. Florian, W. Markus, H. Wolfgang, and R. Andreas, "AIT Log Data Set V1.1." Zenodo, 2020, doi: 10.5281/zenodo.4264796.

[27] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, 2011.

[28] F. Chollet, "Keras," *J. Chem. Inf. Model.*, 2013.

[29] V. Nair and G. E. Hinton, "Rectified linear units improve Restricted Boltzmann machines," 2010.

[30] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," 2015.

[31] X. Meng *et al.*, “MLlib: Machine learning in Apache Spark,” *J. Mach. Learn. Res.*, 2016.
[32] “NLTK.” https://www.nltk.org/.
[33] “igraph.” https://igraph.org/.
[34] “5G-GROWTH 5G-enabled Growth in Vertical Industries H2020 project.” https://5growth.eu/.
[35] “5G-CLARITY EC H2020 5G Infrastructure PPP Phase 3 Project.” https://www.5gclarity.com/.
[36] “YANG Modelling Language.” http://www.yang-central.org.

7. Annex

7.1. Annex A: Interfaces of the Threat Intelligence subcomponents

Table 8: DCP_DC_001 interface specification

METHOD SPECIFICATION	
Interface ID	DCP_DC_001
Interface Point	Collector
Interface Name	Binary Netflow Collection
Description	This interface is deployed on Collector in order to collect binary netflow data forwarded from infrastructure's network devices
Data Source	Network devices
Data Destination	Collector
Data Volume	Forwarded netflows in binary format from network devices
Implementation Mechanism	The interface is deployed in Collector using the nfcapd tool
Syntax	N/A
Pre-condition	N/A
Post-condition	Collected netflows will be dumped in nfcapd files

Table 9: KAFKA_001 interface specification

METHOD SPECIFICATION	
Interface ID	KAFKA_001
Interface Point	Kafka
Interface Name	Kafka topic for anonymized & pre-processed netflow data
Description	This interface exists as a Kafka topic, where anonymized & pre-processed netflow records are ingested. Both anonymization and pre-processing functions have been applied to these records
Data Source	Data Preprocessing module (DP)
Data Destination	Kafka, Data Collection and Data Preprocessing subcomponent (DPC), Multimodal Anomaly Detection subcomponent (MAD), Threat Classification and Alarm Management (TCAM)
Data Volume	Netflow records in CSV format, separated by comma, as they are transformed after application of anonymization & pre-processing functions
Implementation Mechanism	The interface is implemented as a Kafka topic
Syntax	netflow-anonymized-preprocessed Record schema (CSV): ts, te, td, sa, da, sp, dp, pr, flg, fwd, stos, ipkt, ibyt, opkt, obyt, in, out, sas, das, smk, dmk, dtos, dir, nh, nhb, svln, dvln, ismc, odmc, idmc, osmc, mpls1, mpls2, mpls3, mpls4, mpls5, mpls6, mpls7,

	mpls8, mpls9, mpls10, cl, sl, al, ra, eng, exid, tr, tpkt, tbyt, cp, prtcp, prudp, pricmp, prigmp, prother, flga, flgs, flgf, flgr, flgp, flgu Column names are explained in Table 24 in Annex
Pre-condition	Consumers must be registered in the specified Kafka topic
Post-condition	Consumers will fetch Kafka messages with the schema, described above, that will contain netflow records that are both anonymized & pre-processed

Table 10: DCP_DC_002 interface specification

METHOD SPECIFICATION	
Interface ID	DCP_DC_002
Interface Point	Registry service.
Interface Name	Register Kafka Source Connector.
Description	This endpoint exists in the Registry service, in order to add a newly deployed Kafka Source Connector for netflow data.
Data Source	Kafka Source Connector for netflow data.
Data Destination	Registry Service.
Data Volume	The name and url of the new Kafka Source Connector must be provided.
Implementation Mechanism	The interface is implemented as a REST API endpoint.
Syntax	**/register** Body schema (JSON): *{* *name: string,* *url: string* *}* The method can be requested under 1 condition: POST request with provided body
Pre-condition	
Post-condition	A JSON response with status code 200 and a success message will return if everything works fine. Otherwise, if any error occurred a JSON response with status code 400 and an error message will be returned.

Table 11: DCP_DC_003 interface specification

METHOD SPECIFICATION	
Interface ID	DCP_DC_003
Interface Point	Registry service.

Interface Name	Unregister Kafka Source Connector.
Description	This endpoint exists in the Registry service, in order to delete an existing Kafka Source Connector for netflow data.
Data Source	Kafka Source Connector for netflow data.
Data Destination	Registry Service.
Data Volume	The name of an existing Kafka Source Connector must be provided.
Implementation Mechanism	The interface is implemented as a REST API endpoint.
Syntax	**/unregister** Body schema (JSON): *{* *name: string* *}* The method can be requested under 1 condition: POST request with provided body
Pre-condition	
Post-condition	A JSON response with status code 200 and a success message will return if everything works fine. Otherwise, if any error occurred a JSON response with status code 400 and an error message will be returned.

Table 12: DCP_DC_004 interface specification

METHOD SPECIFICATION	
Interface ID	DCP_DC_004
Interface Point	Registry service.
Interface Name	Update a record of an existing Kafka Source Connector.
Description	This endpoint exists in the Registry service, in order to update an existing Kafka Source Connector for netflow data.
Data Source	Kafka Source Connector for netflow data.
Data Destination	Registry Service.
Data Volume	The name and the updated url of an existing Kafka Source Connector must be provided.
Implementation Mechanism	The interface is implemented as a REST API endpoint.
Syntax	**/update** Body schema (JSON): *{* *name: string,* *url: string* *}* The method can be requested under 1 condition: POST request with provided body

Pre-condition	
Post-condition	A JSON response with status code 200 and a success message will return if everything works fine. Otherwise, if any error occurred a JSON response with status code 400 and an error message will be returned.

Table 13: DCP_DC_005 interface specification

METHOD SPECIFICATION	
Interface ID	DCP_DC_005
Interface Point	Registry service.
Interface Name	Get the next available Kafka Source Connector for netflow data.
Description	This endpoint exists in Registry service, in order to balance the load between all available connectors. It returns every time the next available connector, using round robin distribution.
Data Source	Registry Service.
Data Destination	Collector.
Data Volume	No params or body needs to be provided.
Implementation Mechanism	The interface is implemented as a REST API endpoint.
Syntax	**/target** The method can be requested under 1 condition: GET request
Pre-condition	At least one Kafka Source Connector for netflow data must be already registered.
Post-condition	A JSON response with status code 200, the name and the URL of a Kafka Source Connector for netflow data and a success message will return if everything works fine. Otherwise, if any error occurred a JSON response with status code 400 and an error message will be returned.

Table 14: DCP_DC_006 interface specification

METHOD SPECIFICATION	
Interface ID	DCP_DC_006
Interface Point	Registry service.
Interface Name	Get all available Kafka Source Connectors for netflow data.
Description	This endpoint exists in the Registry service, in order to get all available Kafka Source Connectors for netflow data.
Data Source	Registry Service.
Data Destination	
Data Volume	No params or body needs to be provided.
Implementation Mechanism	The interface is implemented as a REST API endpoint.
Syntax	**/services**

	The method can be requested under 1 condition: GET request
Pre-condition	
Post-condition	A JSON response with status code 200 and a JSON array with all available connectors will return if everything works fine. Otherwise, if any error occurred a JSON response with status code 400 and an error message will be returned.

Table 15: DCP_DC_007 interface specification

METHOD SPECIFICATION	
Interface ID	DCP_DC_007
Interface Point	Kafka Source Connector.
Interface Name	Receive nfcapd file for conversion.
Description	This endpoint exists in all Kafka Source Connectors for netflow data. It is responsible for retrieving a nfcapd file and converting it to .csv format.
Data Source	Collector
Data Destination	Kafka Source Connector for netflow data.
Data Volume	The filename and the binary content of the file must be provided.
Implementation Mechanism	The interface is implemented as a REST API endpoint.
Syntax	**/convert** HTTP Request Parameters: Filename: string Body schema (JSON): *{* *data: string* *}* The method can be requested under 1 condition: POST request with provided body
Pre-condition	
Post-condition	A JSON response with status code 200 and a success message will return if everything works fine. Otherwise, if any error occurred a JSON response with status code 400 and an error message will be returned.

Table 16: DCP_DC_008 interface specification

METHOD SPECIFICATION	
Interface ID	DCP_DC_008
Interface Point	Kafka Source Connector.
Interface Name	Ping the connector.

Description	This endpoint exists in a Kafka Source Connector for health check purposes.
Data Source	Registry
Data Destination	Kafka Source Connector.
Data Volume	No params or body need to be provided.
Implementation Mechanism	The interface is implemented as a REST API endpoint.
Syntax	**/ping** The method can be requested under 1 condition: GET request
Pre-condition	
Post-condition	A JSON response with status code 200 and a success message will return if everything works fine. Otherwise, if any error occurred a JSON response with status code 400 and an error message will be returned.

Table 17: DCP_AS_001 interface specification

METHOD SPECIFICATION	
Interface ID	DCP_AS_001
Interface Point	Anonymization Service.
Interface Name	Anonymize an IP address.
Description	This endpoint exists in Anonymization Service, to anonymize given IP addresses.
Data Source	Data Preprocessing module (DP)
Data Destination	Anonymization Service.
Data Volume	The IP address to be anonymized.
Implementation Mechanism	The interface is implemented as a REST API endpoint.
Syntax	**/anonymize** Body schema (JSON): *{* *IpAddr: string* *}* The method can be requested under 1 condition: POST request with provided body
Pre-condition	
Post-condition	A JSON response with status code 200, a success message, the original and the obfuscated IP addresses and the time needed for execution (in seconds) will return if everything works fine. Otherwise, if any error occurred a JSON response with status code 400 and an error message will be returned.

Table 18: DCP_AS_002 interface specification

METHOD SPECIFICATION	
Interface ID	DCP_AS_002
Interface Point	Anonymization Service.
Interface Name	Deanonymize an IP address.
Description	This endpoint exists in Anonymization Service, to deanonymize given obfuscated IP addresses.
Data Source	
Data Destination	Anonymization Service.
Data Volume	The IP address to be de-anonymized.
Implementation Mechanism	The interface is implemented as a REST API endpoint.
Syntax	**/deanonymize** Body schema (JSON): *{* *IpAddr: string* *}* The method can be requested under 1 condition: POST request with provided body
Pre-condition	The given IP address must be the result of a previous anonymization function. Otherwise, the original IP cannot be found in Anonymization Service's storage.
Post-condition	A JSON response with status code 200, a success message, the original and the obfuscated IP addresses and the time needed for execution (in seconds) will return if everything works fine. Otherwise, if any error occurred a JSON response with status code 400 and an error message will be returned.

Table 19: KAFKA_002 interface specification

METHOD SPECIFICATION	
Interface ID	KAFKA_002
Interface Point	Kafka
Interface Name	Kafka topic for raw netflow data.
Description	This interface exists as a Kafka topic, where collected raw netflow records are ingested.
Data Source	Kafka Source Connectors for netflow data.
Data Destination	Kafka.
Data Volume	Netflow records in CSV format, separated by comma, as they are extracted using nfdump tool.
Implementation Mechanism	The interface is implemented as a Kafka topic.
Syntax	**netflow-raw**

	Record schema (CSV): ts, te, td, sa, da, sp, dp, pr, flg, fwd, stos, ipkt, ibyt, opkt, obyt, in, out, sas, das, smk, dmk, dtos, dir, nh, nhb, svln, dvln, ismc, odmc, idmc, osmc, mpls1, mpls2, mpls3, mpls4, mpls5, mpls6, mpls7, mpls8, mpls9, mpls10, cl, sl, al, ra, eng, exid, tr Column names are explained in Table 24 in Annex
Pre-condition	Consumers must be registered in the specified Kafka topic.
Post-condition	Consumers will fetch Kafka messages with the schema, described above, that will contain raw netflow records.

Table 20: KAFKA_003 interface specification

METHOD SPECIFICATION	
Interface ID	KAFKA_003
Interface Point	Kafka
Interface Name	Kafka topic for anonymized netflow data.
Description	This interface exists as a Kafka topic, where collected anonymized only netflow records are ingested. No pre-processing function has been applied to these records.
Data Source	Data Preprocessing module (DP).
Data Destination	Kafka.
Data Volume	Netflow records in CSV format, separated by comma, as they are transformed after anonymization.
Implementation Mechanism	The interface is implemented as a Kafka topic.
Syntax	**netflow- anonymized** Record schema (CSV): ts, te, td, sa, da, sp, dp, pr, flg, fwd, stos, ipkt, ibyt, opkt, obyt, in, out, sas, das, smk, dmk, dtos, dir, nh, nhb, svln, dvln, ismc, odmc, idmc, osmc, mpls1, mpls2, mpls3, mpls4, mpls5, mpls6, mpls7, mpls8, mpls9, mpls10, cl, sl, al, ra, eng, exid, tr Column names are explained in Table 24 in Annex
Pre-condition	Consumers must be registered in the specified Kafka topic.
Post-condition	Consumers will fetch Kafka messages with the schema, described above, that will contain anonymized only netflow records.

Table 21: KAFKA_004 interface specification

METHOD SPECIFICATION	
Interface ID	KAFKA_004
Interface Point	Kafka.

Interface Name	Kafka topic for pre-processed netflow data.
Description	This interface exists as a Kafka topic, where pre-processed only netflow records are ingested. No anonymization has been applied to the records of this topic.
Data Source	Data Preprocessing module (DP)
Data Destination	Kafka
Data Volume	Netflow records in CSV format, separated by comma, as they are transformed after application of pre-processing functions.
Implementation Mechanism	The interface is implemented as a Kafka topic.
Syntax	**netflow- preprocessed** Record schema (CSV): ts, te, td, sa, da, sp, dp, pr, flg, fwd, stos, ipkt, ibyt, opkt, obyt, in, out, sas, das, smk, dmk, dtos, dir, nh, nhb, svln, dvln, ismc, odmc, idmc, osmc, mpls1, mpls2, mpls3, mpls4, mpls5, mpls6, mpls7, mpls8, mpls9, mpls10, cl, sl, al, ra, eng, exid, tr, tpkt, tbyt, cp, prtcp, prudp, pricmp, prigmp, prother, flga, flgs, flgf, flgr, flgp, flgu Column names are explained in Table 24 in Annex
Pre-condition	Consumers must be registered in the specified Kafka topic.
Post-condition	Consumers will fetch Kafka messages with the schema, described above, that will contain pre-processed only netflow records.

Table 22: KAFKA_005 interface specification

METHOD SPECIFICATION	
Interface ID	KAFKA_005
Interface Point	Kafka
Interface Name	Kafka topic for netflow-based threat findings.
Description	This interface exists as a Kafka topic, where the analyzed netflow assigned with an anomaly score and a threat label are send to other PALANTIR components for policy recommendation and visualization purposes.
Data Source	Threat Classification and Preprocessing (TCAM)
Data Destination	Kafka.
Data Volume	Netflow data in JSON format enriched with additional information derived from MAD and TCAM operations.
Implementation Mechanism	The interface is implemented as a Kafka topic.
Syntax	**threat-findings-netflow** Record schema (JSON): {

	"Threat_Finding": { "Time_Start": timestamp, "Time_End": timestamp, "Time_Duration": float, "Source_Address": string, "Destination_Address": string, "Source_Port": integer, "Destination_Port": integer, "Protocol": string, "Flag": string, "Soure_tos": integer, "Input_packets": integer, "Input_bytes": integer }, "Threat_Label": string, "Classification_Confidence": float, "Outlier_Score": float },
Pre-condition	Consumers must be registered in the specified Kafka topic.
Post-condition	Consumer components (e.g. RR, Portal) must comply with the above schema to properly parse the output of the analytics process.

Table 23: KAFKA_006 interface specification

METHOD SPECIFICATION	
Interface ID	KAFKA_006
Interface Point	Kafka
Interface Name	Kafka topic for syslog-based threat findings.
Description	This interface exists as a Kafka topic, where the analyzed syslog assigned with an anomaly score and a threat label are send to other PALANTIR components for policy recommendation and visualization purposes.
Data Source	Threat Classification and Preprocessing (TCAM)
Data Destination	Kafka.
Data Volume	Syslog data in JSON format enriched with additional information derived from MAD and TCAM operations.
Implementation Mechanism	The interface is implemented as a Kafka topic.
Syntax	**threat-findings-syslog** Record schema (JSON):

	{
	"Timestamp": timestamp,
	"Hostname": string,
	"PID": integer,
	"Message": string,
	"Threat_Label": string
	"Classification_Confidence": float,
	"Outlier_Score": float
	}

Pre-condition	Consumers must be registered in the specified Kafka topic.
Post-condition	Consumer components (e.g. RR, Portal) must comply with the above schema to properly parse the output of the analytics process.

7.2. Annex B: Data models

Table 24: Netflow schema

Column Abbreviation	Description	Column Abbreviation	Description
ts	Start Time - first seen	mpls1	MPLS label 1
te	End Time - last seen	mpls2	MPLS label 2
td	Duration	mpls3	MPLS label 3
sa	Source Address	mpls4	MPLS label 4
da	Destination Address	mpls5	MPLS label 5
sp	Source Port	mpls6	MPLS label 6
dp	Destination Port	mpls7	MPLS label 7
pr	Protocol	mpls8	MPLS label 8
flg	TCP Flags	mpls9	MPLS label 9
fwd	Forwarding Status	mpls10	MPLS label 10
stos	Source Tos	cl	Client latency
ipkt	Input Packets	sl	Server latency
ibyt	Input Bytes	al	Application latency
opkt	Output Packets	ra	Router IP Address
obyt	Output Bytes	eng	Engine Type/ID
in	Input Interface num	exid	Exporter ID
out	Output Interface num	tr	Time the flow was received by the collector
sas	Source AS	tpkpt	Total flow packets for bidirectional flows. For unidirectional this value will be the same as input packets (ipkt).

das	Destination AS	tbyt	Total flow bytes for bidirectional flows. For unidirectional this value will be the same as input bytes (ibyt).
smk	Source mask	cp	Flag if flow destination port is a port, used by common services. Values is 1 if a port is a common port, otherwise it is 0. Ports of common services: FTP(20,21), SSH(22), Telnet(23), SMTP(25), DNS(53), DHCP(67,68), TFTP(69), HTTP(80), POP3(110), NNTP(119), NTP(123), IMAP4(143), SNMP(161), LDAP(389), HTTPS(443), IMAPS(993), RADIUS(1812), AIM(5190)
dmk	Destination mask	prtcp	TCP Protocol Flag. Values is 1 if protocol is TCP, 0 otherwise.
dtos	Destination Tos	prudp	UDP Protocol Flag. Values is 1 if protocol is UDP, 0 otherwise.
dir	Direction: ingress, egress	pricmp	ICMP Protocol Flag. Values is 1 if protocol is ICMP, 0 otherwise.
nh	Next-hop IP Address	prigmp	IGMP Protocol Flag. Values is 1 if protocol is IGMP, 0 otherwise.
nhb	BGP Next-hop IP Address	prother	Other Protocol Flag. Values is 1 if protocol is not TCP, UDP, ICMP or IGMP, 0 otherwise.
svln	Src vlan label	flga	TCP Control Flag (A). Value is 1 if TCP Flag A is in flow's TCP flags.
dvln	Dst vlan label	flgs	TCP Control Flag (S). Value is 1 if TCP Flag S is in flow's TCP flags.
ismc	Input Src Mac Addr	flgf	TCP Control Flag (F). Value is 1 if TCP Flag F is in flow's TCP flags.
odmc	Output Dst Mac Addr	flgr	TCP Control Flag (R). Value is 1 if TCP Flag R is in flow's TCP flags.
idmc	Input Dst Mac Addr	flgp	TCP Control Flag (P). Value is 1 if TCP Flag P is in flow's TCP flags.
osmc	Output Src Mac Addr	flgu	TCP Control Flag (U). Value is 1 if TCP Flag U is in flow's TCP flags.

Table 25: Syslog schema

Column	**Description**
timestamp	Unix timestamp describing when the event occurred.
hostname	The name assigned to a device connected to a computer network.
service pid	The name of the service that generated the event. PID of the Program that generated the event (not always available).
message	Text description of the event.