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)

D3.1 PALANTIR Secure Services Ecosystem - First Release

Document Identification			
Status	Final	**Due Date**	30/11/2021
Version	1.0	**Submission Date**	30/11/2021

Related WP	WP3	**Document Reference**	1.0
Related Deliverable(s)	D4.1, D5.1, D3.2, D4.3, D5.2	**Dissemination Level (*)**	PU
Lead Participant	i2CAT	**Lead Author**	i2CAT
Contributors	i2CAT, UBI, UMU, ORION	**Reviewers**	UBI
			UMU

Keywords:
SecaaS, Security Orchestrator, Security Capabilities Catalogue, Risk Analysis Framework, architecture, design, specification, implementation, interfaces, data models

Document Information

List of Contributors	
Name	Partner
Carolina Fernández, Maxime Compastié	i2CAT
Mattia Zago, Antonio López	UMU
Stelios Tsarsitadilis	UBITECH
Georgios Xylouris	ORION

Document History			
Version	Date	Change editors	Changes
0.1	6/10/2021	Carolina Fernández	Initial Table of Contents
0.2	10/11/21	Antonio López	UMU contribution
0.3	12/11/21	Carolina Fernández	i2CAT contribution

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	i2CAT	28/11/2021
Quality manager	INFILI	29/11/2021
Project Coordinator	DBC	30/11/2021

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AAC	Authentication & Authorisation Control (SO module)
AE	Attestation Engine (subcomponent for TAR)
ATR	Attestation & Remediation (module for SO)
API	Application Programming Interface (SO module)
BUp	Backup (SC type)
CA	Certificate Authority
CFG	Configuration (module for SO)
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
DB	Database
DBL	DB Layer (module for SO)
DoS	Denial of Service
DPI	Deep Packet Inspection (SC type)
EMS	Element Management System (SC module)
ENISA	European Network and Information Security Agency
FW	Firewall and Router (SC type)
GB	Gigabyte
GDPR	General Data Protection Regulation
HW	Hardware
IDS	Intrusion Detection System (SC type)
IP	Internet Protocol
JWT	JSON Web Token
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
L-SL	Lightweight Shipper for Logs (SC type)
LCM	Life-Cycle Management (module for SO)
LSPL	Low Level Security Policy Language (used by SC)
MMML	Multi-Modal Machine Learning (TI subcomponent)
MON	Monitoring (module for SO)
MSPL	Medium Level Security Policy Language (processed by SC)
NFV	Network Function Virtualisation
NFVI	NFV Infrastructure
NFVO	NFV Orchestrator
NNS	Network Netflow Sniffer (SC type)
NS	Network Service
OLM	Operator Lifecycle Manager (indirectly used by SC and SO)
OSI	Open Systems Interconnection
PCAP	Packet CAPture
PKI	Public-Key Infrastructure
PKG	Package (module for SO)

Abbreviation / acronym	Description
POL	Policies (module for SO)
RAF	Risk Analysis Framework (PALANTIR component)
RAM	Random Access Memory
REST	REpresentational State Transfer
RM	Reference Measurement (processed by AE)
RR	Remediation and Recommendation (TI subcomponent)
SC	Security Capability
SCC	Security Capabilities Catalogue (SCO subcomponent)
SCC-O	Security Capability Onboarding (SCC module)
SCDV	Security Capability and Descriptors Validation (SCC module)
SCMA	Security Capability Metadata Access (SCC module)
SCO	Security Capabilities Orchestrator (PALANTIR component)
SCPM	Security Capability Package Maker (SCC module)
SCR	Security Capability Registration (SCC module)
SCS	Security Capability Search (SCC module)
SecaaS	Security-as-Service
SEM	Security Element Manager (SC module)
SLA	Serviec Level Agreement
SM	Service Matcher (Portal subcomponent)
SME	Small and Medium sized Enterprises
SSH	Secure SHell
SO	Security Orchestrator (SCO subcomponent)
SW	Software
TAR	Trust, Attestation & Recovery
TI	Threat Intelligence (PALANTIR component)
VDU	Virtual Deployment Unit (computing element for the VNF)
VIM	Virtualised Infrastructure Manager
VL	Virtual Link (networking element for the NS)
VM	Virtual Machines
VNF	Virtual Network Function
VNFD	VNF Descriptor
WP	Work Package
WTA	Web-based Traffic Analysis (SC type)

Executive Summary

This report departs from the architecture provided for WP3-related subcomponents in D2.1 and dives deeper into their detailed definition. Specifically, this involves the following components or subcomponents from the PALANTIR architecture: (i) Security Capabilities (SCs), or SecaaS (Security-as-a-Service); (ii) Security Capabilities Orchestrator (SCO), in charge of (a) managing the SC's life-cycle management via the Security Orchestrator (SO), and (b) keeping a searchable Security Capabilities Catalogue (SCC); and the (iii) definition of the Risk Analysis Framework (RAF).

In **section 2**, every subcomponent has its design described. It provides a general view on the key ideas sustaining the design for each subcomponent, along with any other general or transversal design-related data. Then, it describes the foreseen interfaces and communication channels that will interact with other PALANTIR subcomponents. After this, it describes every internal module, making its goal explicit, behaviour and relations in the internal interactions or workflows devised for the subcomponent. Finally, architectural or design changes that have diverged from D2.1 are explained.

Section 3 gathers the requirements described in D2.1 and revisits its application within WP3. To do so, it maps the architectural requirements to implementation specifications. This effectively refines more general requirements to specific technical needs to be fulfilled and is an exercise towards a more clear and concise definition of the expected features of each subcomponent.

Section 4 compiles the list of specific technologies in use per module within each subcomponent. Each subsection highlights their objective, benefits and adequacy to the expected environment to be in use per module.

Finally, the **annexes** are provided in a per-subcomponent basis, where each subcomponent has its own annex and can provide the most relevant data to its current design and development status; considering either data models in use and/or internal interactions between modules (also called intra-component), as well as user guides related to the subcomponent.

1. Introduction

1.1. Objectives and goals of the deliverable

This deliverable, "D3.1 PALANTIR Secure Services Ecosystem – First Release", highlights the current work performed in tasks T3.1 to T3.4 in relation to the design of each of the related subcomponents. The document also provides their technical specs (obtained, iterated and updated from the general requirements agreed in D2.1) and the low-level implementation details related to the specific dependencies, libraries and tools in use or expected to be in use.

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 WP3 and/or in the other directly or indirectly related WPs (such as WP4 and WP5). This report provides design and implementation hints that can be of use to any technical external reader involved in the virtualised instance orchestration and cybersecurity environment.

This deliverable is the first iteration, out of two, within the WP3-related deliverables. The next deliverable is due month 30, where the final design and implementation must be provided, as well as other technical interesting details such as the inter-component APIs and other interfaces in use to enforce the cross-WP workflows within the PALANTIR platform.

1.2. Relation with D2.1 and other WPs

This document uses D2.1 as a starting point to dive into lower technical details about the WP3-related components and subcomponents. Specifically, this refines first the original PALANTIR architecture into the design of each component and subcomponent. This work also leverages the latest version of the internally agreed workflows between architectural components; in such a way that the expected behaviour described in the workflows is taken into account and updated by the design and interactions here described. Secondly, the list of requirements indicated in D2.1 are mapped here to technical specifications, explicitly stating the need for the specific subcomponent impacted by the requirement.

Besides this, the work exposed here relates strongly to WP4, whose components and subcomponents directly interact with those from WP3. These account for attestation and remediation operations, as well as the mapping of identified threats to specific available SCs, consequently implemented as SecaaS services, and to the visualisation of the information provided by the lower layers in both WP3 and WP4.

Finally, there is also an indirect relation to WP5, since it defines the identified threats from monitoring records provided by WP3, as well as the mitigation recommendations. Then, WP4 propagates these and WP3 uses them to exert such recommendations as applicable configurations within the SCs.

2. Design

This section details the design choices for the WP3-related components and its internal subcomponents (and modules). Specifically, the following sections document the design for the SCs, the SCO (which contains the SO and SCC subcomponents) and the RAF.

2.1.Security Capabilities

Regarding the SCs, the consortium settled on a standardised structure aligned with the requirements in section 3.1, and finally partially implemented in section 4.1. The standard design or internal structure for the SCs is introduced first. This aligns with the requirements and implementation details defined in the upcoming sections.

Each SC features two main modules: the Security Element Manager (SEM) and the Virtual Network Function (VNF). The former provides well-defined and standard interoperability interfaces to enable the dynamic deployment and configuration of the latter. Apart from the two main modules, which will be described thoroughly later on in this section, Figure 1 presents the general architecture of any SC, including inter-component communications and interfaces, internal networking, functionalities to be offered, and level of designated security policy to be used.

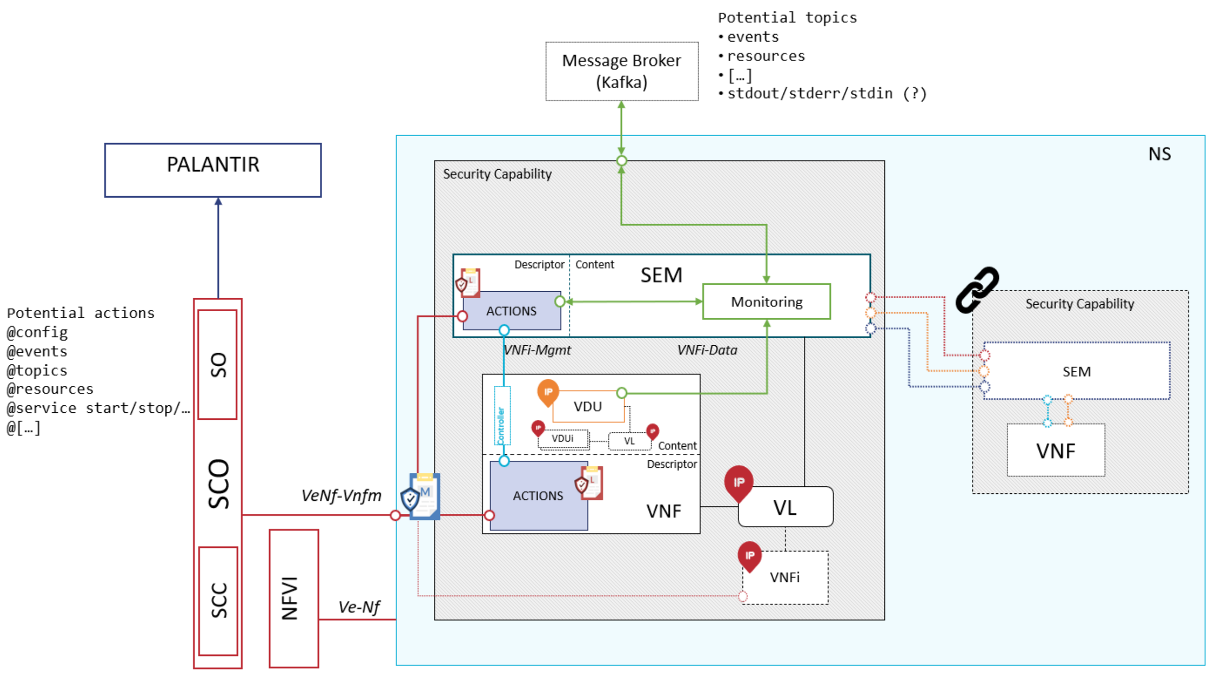

Figure 1: SC detailed architecture.

The rest of this section is organised to cover the multiple aspects included in Figure 1.

2.1.1. Interfaces and communication channels

To begin with, Figure 1 identifies different interfaces. In the SC context, two internal interfaces appear between the SEM and the VNF, the VNFi-Mgmt and the VNFi-Data. Besides, it shows two external interfaces with SO and the NFV Infrastructure (NFVI): the VeNf-Vnfm and Ve-Nf, respectively. The following sections explain all these interfaces.

As identified first in D2.1, the SO will manage and connect to any given SC via a management interface named "VeNf-Vnfm". SC developers must implement lifecycle management actions (*e.g.,* service initialisation and controlled shutdown, among others) and service monitoring (*e.g.,* health checks, performance, and resource usage). The endpoints exposed by this interface must also provide reconfiguration capabilities and direct translation from Medium-Level Security Policies (MSPL) to Low-Level Security Policies (LSPL) [1]. For both of these functionalities, the communication achieves the SEM first, and the reconfiguration/LSPL policies and the service monitoring are both conducted to the VNF via the VNFi-Mgmt link.

On the other hand, the SCs need a direct connection to the physical infrastructure (NFVI) to be operated. The name of such a link is "Vn-Nf" and can be used for different purposes. For example, the NFVI should expose the available resources to SCs due to possible constraints in their executions. The SO subcomponent also manages this connection since it is in charge of indicating the deployment time, and has a holistic view of the PALANTIR platform.

When communicating with other components, we use the Kafka message broker. Kafka is the jointly chosen principal channel for inter-component communications in PALANTIR. As such, each SC shall provide Kafka-related capabilities. In the current architecture, such functionalities are delegated to the Monitoring Module within the SEM controller. This module uses the VNFi-Data interface in order to collect the data produced thanks to such VNFs deployed in the NS context, with the intention of processing, formatting and introducing them into the given Kafka topic.

2.1.2. Modules

Each SC contains one or more VNFs, which form a Network Service (NS). The actual number of Virtual Deployment Units (VDUs), Virtual Links (VL) and other internal elements is left to the final developer, and as such, there are no particular restrictions. However, each SC shall only expose one interface for command and configuration functionalities (typically referred as "mgmt" network) and one interface for communication (typically referred as "data" network"). If the developer requires multiple chained VNFs, the orchestration and interconnection of those cannot be delegated to the SO but instead has to be managed internally within the SEM module (represented in the right side of Figure 1 by the three connections between both the primary and the chained SCs). For simplicity, Figure 1 depicts the SEM and SC functionalities as separate entities; however, there are no strong constraints on the location of the SEM elements. For example, on the one hand, those functionalities might be implemented as system services within the same image for simpler SCs; while, on the other hand, they might be deployed as separated VDUs/VNFs in scenarios with higher complexity.

In PALANTIR, the SEM module is the leading enabler for the dynamic deployment and integration of the SecaaS functionalities. Indeed, both the monitoring and mitigation aspects, prepared by the PALANTIR dynamic approach to cyber-resilience, are built upon the interchangeability of one SC by another, since the SEM provides abstracted means to interact with the SC; no matter how their internal implementation is carried out. The primary service fulfilled by the SEM is provided by the Monitoring Component, an abstract block that manages the I/O in a standardised format, common to all SCs. For example, a Deep Packet Inspection (DPI) SC will ingest network traffic and return alerts and detection events. For that, the SEM takes care of the communication and interoperability aspects, including authentication and encryption requirements demanded by the connected components and services. From the perspective of the intelligence plane of PALANTIR, addressed in WP5, the actual implementation of the required functionalities of a DPI are relevant only to a certain degree; indeed, thanks to the SEM, deploying SNORT rather than SURICATA is a matter of service matching and not a technological challenge.

In other words, future PALANTIR developers might provide equivalent SecaaS packages that abstract from the technical implementation details and yet deliver the same high-level functionalities.

Regarding these SecaaS functionalities, internally, they must implement at least one VDU. In other words, the minimum packaged SC consists of a NS including a VNF that contains at least a VDU with all the offered services and the minimal SEM interfaces. An IP address will be assigned to such VDU "from" the SCO for communication and eventual services exposition; however, the developer might request more depending on the SC requirements. The developers manage the internal networking that has to be formally specified in the descriptor of the package.

2.1.3. Guaranteed functionalities

Each SC shall offer the I/O capabilities associated with the chosen PALANTIR message broker (Kafka). Specifically, each SC shall provide two output streams, one for the logs and monitoring data and one for the actual SC functionalities output. The former shall be disabled by default and enabled only upon SO request via Juju actions. It is up to the developer to accept input data via Kafka or other means as well as other interface provisions, such as APIs or web services. T3.1 provides a ready-to-use monitoring solution that developers can integrate to fulfil this requirement without creating a custom one (more details are provided in section 4.1).

Each SC's SEM shall provide a minimal set of Juju actions that can interpret a given MSPL statement and convert it to LSPL statements (*i.e.,* configuration files) for the implemented services. WP5 takes care of the formalisation of such format. The SO transmits the orchestration, management and configuration operations towards the VeNf-Vnfm interface, and these are enacted via Juju's Charmed Operator Lifecycle Manager (OLM) actions. Potential actions might include i) configurations (*e.g.,* IP assignments, firewall rules, and others); ii) events and topics (*e.g.,* enable verbose logging, configure topics for generated events, and others); iii) resources (*e.g.,* minimum guaranteed resources); and iv) service management (*e.g.,* restart conditions, halting, and others).

2.1.4. Types of Security Capabilities

At this time, the internal components, connections, and architecture of the SCs have been presented. However, the SCs implemented and deployed in the PALANTIR project belong to two families: monitoring and remediation SCs. These two types of SCs cover the mechanisms needed to protect the Small and Medium sized Enterprises (SME) clients, as well as the PALANTIR infrastructure. Thanks to the monitoring and remediation SCs, the attacks or threats can be detected and mitigated. The first type is defined by the different mechanisms that may monitor the network and devices to find possible anomalous activities or unauthorised behaviours. The second type incorporates the SCs able to react or mitigate once an attack is detected. They can deploy actions in the network or system, such as blocking traffic, removing malicious files, etc. The following sections present the main characteristics and the types of SCs covered by PALANTIR.

The PALANTIR platform covers different types of SCs in order to implement monitoring and remediation functionalities. The selected SCs for the first release of the Secure Services Ecoststem are: an Intrusion Detection System (IDS), a Deep Packet Inspection (DPI), a Network NetFlow Sniffer (NNS), a Lightweight Shipper for Logs (L-SL), a Firewall and Router (FW), a Backup (BUp) and a Web-based Traffic Analysis (WTA) solution. An important aspect to mention is that the maturity level of the SCs can differ, taking into account the degree of implementation performed so far. For this reason, the explanation shown below may differ across the SCs presented. Finally, the list of different SCs is susceptible to change by D3.2, following the architecture presented in Figure 1, in case the implementation requirements impose it.

2.1.4.1. Intrusion Detection System (IDS)

The IDS SC implements the required threat detection techniques. There are two primary threat detection techniques: signature-based detection and anomaly-based detection. The first technique uses known characteristics of such attacks to identify them in the monitored traffic or system. For anomaly-based detection, this technique stores the normalised baseline and generates an alert when detecting an unknown behaviour or activity. These detection techniques are essential when deciding whether to go with a signature- or anomaly-based detection engine. At the implementation level, presented in section 4.1.1, most of the software listed incorporates signature-based rules. The anomaly-based detection approach will be implemented with the help of the Multi-Modal Machine Learning (MMML) and

Remediation and Recommendation (RR) subcomponents, being designed and implemented by PALANTIR's T5.2 and T5.4 respectively, as part of the Threat Intelligence (TI) component.

The IDS is commonly installed into one or more systems, devices or sensors of the network with the purpose of correctly analysing all parts of the environment. The workflow of an IDS is the following: the IDS has a predefined knowledge base with characteristics of known threats. Leveraging this, the IDS is able to match the traffic analysed via attack signatures. After detecting a threat, the IDS can enable established rules to generate an alert to notify of the intrusion found. This alert will be transmitted to the specific Kafka broker for a further (more holistic approach) analysis.

2.1.4.2. Deep Packet Inspection (DPI)

In this case, the DPI implements the functionality required to inspect IP packets through the different Open Systems Interconnection (OSI) layers. The DPI allows organisations to discover different attacks, such as Denial of Service (DoS), malware, web attack, and data exfiltration attack. Besides, the DPI can act as an IDS since it performs traffic analysis and detects anomalous messages. The DPI can be compared to traditional packet filtering, yet both technologies differ in the inspection depth of the messages. Specifically, the packet filtering analyses only the packet's header, and the DPI also enters into the content of the message.

Regarding the functioning of a DPI, it shares the steps presented for the IDS SC, analysing the traffic, enabling the predefined rules for attack detection, and notifying the user upon the discovery of the attack. Both DPI and IDS technologies can be integrated into more complex tools that incorporate different functionalities to provide complete solutions as central defence components. Section 4.1.2 presents the implementation software that can incorporate DPI as an integrated tool.

2.1.4.3. Network Netflow Sniffer (NNS)

The network traffic needs to be collected for the MMML subcomponent and for the distributed collection and data pre-processing capabilities offered by PALANTIR platform (part of TI component). In this context, the NNS is implemented to collect the network traffic and prepare the packets to be sent to other PALANTIR components. This SC can be understood as a supporting element, and not as monitoring or remediation SC. In this sense, this type of SC is also important for the correct implementation of the PALANTIR project. The relation between components across the PALANTIR platform imposes the requirement of providing information, mainly acquired by the SCs (here acting as sensors) and distributed to the other components. Section 4.1.3 presents specific implementation details of an NNS.

2.1.4.4. Lightweight Shipper for Logs (L-SL)

As commented above, PALANTIR will use Kakfa to send relevant information between all components and subcomponents. Besides, the information transmitted must follow an understandable form for all parts of PALANTIR. For this reason, the L-SL is designed and selected to be included in the SC. In this sense, the L-SL is in charge of collecting the information created by other SCs, formatting it, and sending it through the specific Kafka topic designated by each specific data. This technology will be used to implement the SEM component, presented in Figure 1. Although the L-SL can be classified as SEM and not as SC, this technology can also be offered as an individual SC to collect distributed data from different sensors, translating them to the proper format.

2.1.4.5. Firewall and Router (FW)

Traditionally, the Firewall solution has been one of the most adopted to protect the organisation network. To this end, PALANTIR delivers different SCs with Firewall and Router capabilities designated to protect the SME client and the PALANTIR infrastructure. This technology offers integrated protection of the network environment through packet filtering, load balancing, and firewall rules created to avoid the intrusion of malicious activities occurring inside the organisation. In recent years, firewall technology has evolved to more integrated solutions with different capabilities added, besides the traditional firewall specifications. Section 4.1.5 will present the implementation details with the capabilities of the new Firewall solutions.

The main functionality of a Firewall is divided into the different types that already exist. Packet filtering, Stateful filtering, Application layer firewalls, Circuit level gateway and Stateful multilayer inspection are some of the available capabilities. In the PALANTIR context, the packet and stateful filtering will initially be developed, but more types can be incorporated into the project at a later stage.

2.1.4.6. Backup (BUp)

When an attack is produced, the data belonging to the organisation can be stolen or removed, and the recuperation of this information may be very difficult. Due to this problem, PALANTIR pretends to offer a Backup SC, with plug-and-play characteristics, which aims to perform repetitive backup procedures to maintain an updated copy of the state of the data belonging to an organisation. This SC is categorised into the mitigation kind of SCs, since it allows SMEs to avoid irreparable damage by restoring the latest information stored. Section 4.1.6 presents the development approach followed for this SC.

2.1.4.7. Web-based Traffic Analysis (WTA)

In an SME context, the workers' cybersecurity education is not enough to avoid different attacks, such as phishing or hijacking. PALANTIR provides a WTA SC aimed at analysing the web traffic and detecting possible anomalous information transmitted across the web environment, in order to prevent this situation. This SC monitors layer 7 application protocols and the packets to the IP layer. The WTA collects different information, such as layer 7 flows, application protocols, the websites visited by the SME, etc. The adoption of this technology within the SME client can mitigate the consequences of the lack of education of many employees in the cybersecurity field, preventing social engineering and phishing attacks.

The functionality of the WTA is similar to that showcased by the IDS and DPI SCs, since this technology can be understood as a more specific mechanism intended for the application layer.

2.1.5. Changes from D2.1

Regarding the usage of Dew and Edge techniques in general, initial experiments have been carried out; however, no changes are reported since D2.1.

There are new design and implementation requirements that motivate the following changes. On the one hand, the SO uses the Juju actions to provide the configuration changes and MSPLs. Accordingly, the more feasible option is to standardise this mechanism to receive the interaction with the SO via the VNFi-Mgmt. On the other hand, the SEM is the unique element able to collect directly data from the VNFs through the VNFi-Data. Hence, the SEM will deliver the data via Kafka, as agreed by the PALANTIR project.

From the cross-Work Package meetings it also emerged that, within the descriptors of the SCs, there should be a section dedicated to privacy and the aspects related to the General Data Protection Regulation (GDPR). For example, a DPI monitoring tool might collect (among others) the IP address and the PCAP files of the target SME asset. In such a scenario, the developer should disclose this information beforehand, hence enabling both the SM and the Portal to filter and present relevant information. To meet such request, here follows a non-binding example descriptor.

```
<gdpr>
  <sensitive name=’traffic’ type=’pcap’ />
  <sensitive name=’ipaddress’ type=’string’ />
  …
  <not-sensitive name=’country’ type=’string’ />
  …
</gdpr>
```

With respect to the SCs, four new types have been added: a Network NetFlow Sniffer (NNS), a Lightweight Shipper for Logs (L-SL), a Backup (BUp) and a Web-based Traffic Analysis (WTA). This incorporation is accompanied by the new implementation and managing requirements that have been identified from D2.1. Some of them are used for supporting tasks, such as transmission of traffic flows to other PALANTIR components, and others like BUp are used to provide more different security mechanisms in order to protect the SME client.

2.2. Security Orchestrator

Along with SCC (described in the next section), the SO is one of the two subcomponents inside SCO.

This subcomponent interacts with both the PALANTIR components (mostly in WP3 and WP4) and with a number of third-party tools that live in the operation layer (living in a lower level, dealing with infrastructure and runtime). Examples of these are the relevant infrastructure management frameworks (such as an OpenStack VIM or a Kubernetes cluster) or runtime-related tools (such as Docker), as well as the Network Function Virtualisation Orchestrator (NFVO). The NFVO is a key piece for the SC orchestration, as it allows onboarding or registering the SCs (or VNFs and NSs) and managing its life-cycle operations. The SO leverages such functionality and complements with features not covered by the NFVO, such as extending the data model for the PALANTIR needs, keeping track of specific information or nodes, determining the adequacy of deploying and configuring or just configuring, etc.

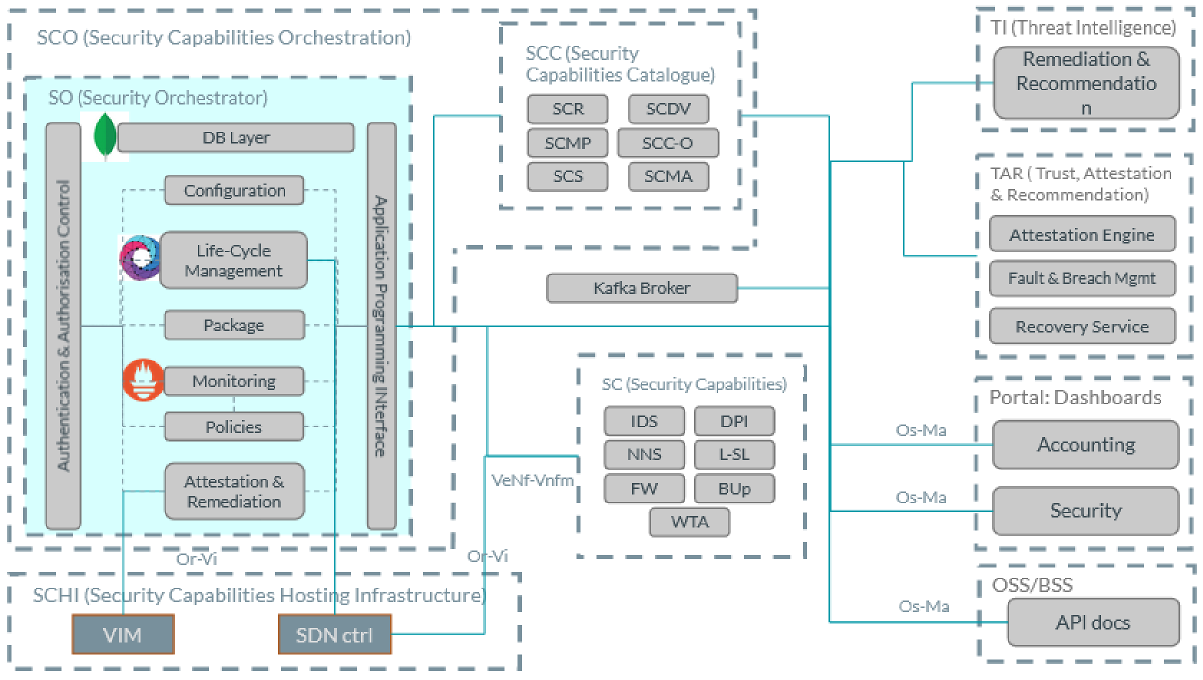
SCO (Security Capabilities Orchestration)
SO (Security Orchestrator)
DB Layer
Configuration
Life-Cycle Management
Package
Monitoring
Policies
Attestation & Remediation
Authentication & Authorisation Control
Application Programming Interface
SCC (Security Capabilities Catalogue)
SCR
SCDV
SCMP
SCC-O
SCS
SCMA
Kafka Broker
SC (Security Capabilities)
IDS
DPI
NNS
L-SL
FW
BUp
WTA
VeNf-Vnfm
Or-Vi
Or-Vi
SCHI (Security Capabilities Hosting Infrastructure)
VIM
SDN ctrl
TI (Threat Intelligence)
Remediation & Recommendation
TAR (Trust, Attestation & Recommendation)
Attestation Engine
Fault & Breach Mgmt
Recovery Service
Portal: Dashboards
Accounting
Security
OSS/BSS
API docs
Os-Ma
Os-Ma
Os-Ma

Figure 2: SO detailed architecture (left) with internal and external interactions.

Figure 2 shows the central position of SO in the PALANTIR architecture, where it receives instantiation and configuration requests for specific SCs, interprets such requests and extends these with any required specific logic. Besides this, it also acts as an abstraction layer between the internal PALANTIR logic and the third-party stacks, frameworks and tools involved in it.

As a central point, the SO must collect and expose a subset of the information related to the deployment and environment, so that this is accessible to any other component within the architecture. Specifically, two main approaches are in use: (i) enrichment of its internal state with results from the intercepted requests and the generated instances; and (ii) proactively polling for specific metrics or telemetry data, which could trigger other actions or logic.

As an abstraction layer, the internal handling and enrichment of data acquired from the operational layer allows two main abstracting operations: from the operation layer to PALANTIR and vice versa. In other words: (i) exposing such operational data (like runtime and environment specific) to the PALANTIR components; and (ii) adapting the requests from such components in a manner that is understandable to the lower layers.

2.2.1. Interfaces and communication channels

In order for the SO subcomponent to deal with the orchestration of the SCs, a number of interactions (firstly identified in D2.1) are expected:

- The NFVO, in order to onboard the SC packages and obtain information on available packages, running SC instances and extra details on both of them.
- The SCC, to provide specific details from the SC packages as obtained from the NFVO.
- The SCs themselves, in order to proxy specific operations: passing the MSPL statements in order to perform configuration (at any stage of the SC instantiation), requesting information such as the health status, etc.
- The NFVI and the VIM. In this case, it can retrieve specific runtime information related to the environment in use to instantiate the SCs, as well as communicating with other third-party tools running within the NFVI and required for proper instantiation.
- The Service Matcher (SM), from which will receive the list of possible SCs to be instantiated to mitigate a specific threat, and the type of infrastructure (delivery mode) where these should be deployed.
- An agent of the Attestation Engine (AE), so that it can retrieve specific runtime-related metrics on the running SCs in order to extract the Reference Measurements (RM) required to perform the attestation.

2.2.2. Modules

The design of the SO currently foresees two types of modules:

- Logic modules: a number of software modules providing specific functions required by the SO, interconnected among them to fulfil the expected internal workflows.
- Transversal modules: those considered as transversal logic that serves the rest of modules.

The next sections explain the logic modules first.

2.2.2.1. Application Programming Interface (API)

The API module acts as an externally facing entry point for the API requests, exposing a single set of interfaces to the rest of the PALANTIR architecture. This module allows requests for high-level operations by contacting its specific interfaces. Internally, this component aggregates operations from different components, which comprehend the higher-level operations or workflows. Thus, it solves the endpoints and ports of the other modules, its interaction and the required authentication. By default, it expects synchronous communication – yet it may consider asynchronous workflows for those operations with special requirements.

This module interacts with all other modules in order to proxy specific requests and enforce workflows.

2.2.2.2. Attestation & Remediation (ATR)

This module handles the operations related to the attestation and recovery, fruit of its interaction with the Attestation Engine (AE), which is part of the Trust, Attestation & Recovery (TAR) component in

WP4. On the one hand, the attestation operations leverage specific bits of data exposed from the runtime environment, used to deploy the SC instances, so that the AE can generate the RM data. On the other hand, when SO registers a new node or SC, it notifies the AE subcomponent and receives the attestation results – in case of the first failed attestation; this may result into applying specific mitigation behaviour in the failed node or SC.

This module interacts with the following modules:

- The AE agent, whose in-situ tooling for attestation metrics capture will rely on the appliances delivered by T4.4 and described in D4.2 and D4.4.

2.2.2.3. Configuration (CFG)

Similar to the API module, this one allows exposing the contents of the configuration used by each module of the SO. Expected for easier troubleshooting of the operator and potential integration with other components, the supported mode will be read-only.

This module does not interact with the interfaces of any other modules, but instead gathers their configurations, stored in a central location in the file system, and exposes these in a unified form.

2.2.2.4. Life-Cycle Management (LCM)

This module is the core of the logic that receives the orchestration requests and implements any extra layer of logic. Some sample of extra logic relates to identifying whether to directly instantiate and configure an instance of SC or, on the contrary, re-use a specific one. It can also set up the expected network connectivity to deploy the service in a way that allows it to comply with its specifications; may transmit the configuration requests or, potentially, other type of requested actions (*e.g.*, Unix commands) in order to obtain specific data.

This module features the following internal interactions:

- POL (Policies): to submit the policy or configuration to a running instance of SC.

2.2.2.5. Monitoring (MON)

Its responsibility is to retrieve specific metrics from the SC instances. In the current design, the operator would be able to choose from the following: (i) selecting a metric from a general pool of metrics expected to be available per SC, or (ii) defining custom metrics to fetch on specific SCs.

In the former case, the metrics will be extracted and persisted using the built-in selected third-party tools. In the latter, the MON module will implement its particular access modes (*i.e.,* SSH via keys) to fetch the custom metrics defined by the operator, as well as translating the data to persist it. The operator is expected to provide the name of the metric, the Unix-like command (thus assuming all SCs are Unix-based) to be executed to obtain such data – which will be subject to filtering within this module – and the target SC where to retrieve such data.

This module has the following internal interactions:

- POL: to provide the telemetry data evaluated by that module.

2.2.2.6. Package (PKG)

Used to handle the SC packages in order to (i) interact with SCC to either retrieve or provide SC information that the SCC persists, or (ii) onboard the SC packages to the NFVO and fetch specific bits of information from the NFVO's internal registry. Such registry contains both the registered SC packages, as well as their detailed data (after extracting their internal files, parsing and persisting these). It is worh noting here that the PKG module also takes care of registering the image associated to the SC package, in order to later quickly instantiate and configure the desired capabilities.

2.2.2.7. Policies (POL)

This module covers two separate functionalities.

On the one hand, those related to the configuration policies received in order to configure the SCs. Configurations are MSPL statements (configuration intentions), as described in section 2.1.3. In PALANTIR, the MSPL is obtained within the platform and transmitted to SO. It internally traverses the LCM module in order to reach any given running instance of SCs. The SC will internally translate the MSPL to a specific, low-level configuration, the LSPL. This module will apply any required extra logic before transmitting the MSPL to the SC via the VeNf-Vnfm interface, as described in section 2.1.1.

On the other hand, the functionalities are related to the monitoring policies. Similar to MON, the operator would be able to indicate the expected behaviour or policies to apply by SO, provided that a given threshold is surpassed for the value of a specific metric.

This module has the following internal interactions:

- MON: where this module monitors the telemetry data obtained and persisted there, in order to raise relevant alerts and execute actions previously defined by the operator.

Finally, the next sections cover transversal modules or layers.

2.2.2.8. Authentication & Authorisation Control (AAC)

This transversal module or layer is expected to cover the authentication and authorisation operations required to interact with any other module within SO. Requests addressed to other modules will be diverted here, evaluated and the outcome of this authentication and authorisation control returns a successful or failed event.

This layer interacts with all other modules in order to receive authentication requests and authorise access from a module to the resources provided by another one.

2.2.2.9. Database Layer (DBL)

A database layer that is common to any other module within SO. All internal data is persistent in this database, organised by specific modules and types of data and accessible by any other internal module.

This layer is in use by all other modules in order to store their own data models, isolated from others yet accessible if communicating with the specific module.

2.2.3. Changes from D2.1

The architecture for the SO subcomponent, as stated in D2.1, has been updated during the refinement process occurring during the first iteration on the design phase. The motivation for such a change is the extension and simplification of the design and the structure to follow once translated to logic in the development phase, along with a more advanced decomposition or separation of concerns in order to enforce the modularity design principle.

Specifically, there was a regrouping of the internal modules (*i.e.*, “events” and “policies” into the “POL” module) and others had its name changed (*i.e.*, “metrics” to “MON”).

Besides this, there are new modules as part of the SO design. There are two categories: logic modules and transversal layers.

First, the newly added modules, foreseen at this stage of design, are: (i) an “API” module that acts as a proxy to interact with the interfaces of all other modules; (ii) an “ATR” module that handles attestation and recovery operations from the SO perspective; and (iii) a “CFG” module that exposes the current configuration of all modules.

Regarding the transversal layers, (i) a database layer (“DBL”) is expected to persist data for all modules and provide a coherent view of data within SCO; and (ii) an authorisation layer (“AAC”) is to be implemented so that it will grant access to all modules.

2.3. Security Capabilities Catalogue

PALANTIR has a single storage subcomponent for SCs, allowing security developers to make new functionalities available to client enterprises of the platform. The SCC is the subcomponent of the SCO that is mainly concerned with securely storing SCs as sets of metadata, serving SC metadata to other PALANTIR components, such as the Billing and SM (T4.3) and the TAR (T4.4), and securely onboarding SC packages to the SO. The SCC also tightly integrates with the UI frontend of the PALANTIR Portal (T4.1), which will be further elaborated upon in D4.1, allowing for visually defining and managing SCs.

The SCC stores the metadata of the SC packages in a trusted way, keeping the SC software image reference, the security and privacy specifications, billing information and other VNF operational metadata. The packages are essentially functional, and include all necessary metadata about SCs, which are stored in the catalogue, along with attestation data about the aforementioned metadata and the main software of the SC. The software of the capabilities is, in turn, packaged as images, such as images for containers or Virtual Machines (VMs). These software images are not stored within the SCC, but only the references to registries that hold them, and the specific image ID and version to use. Only during the registration and onboarding process is the image fetched, and appropriately packaged along with all the descriptors, the accompanying operational and security metadata, and sent to the SO. The SO, in turn, proceeds further with the onboarding process once it obtains the SC packages. During this step, the SO also requests the appropriate tools to register the image or images associated to the SC package, in order to later quickly instantiate and configure the desired capabilities. The SO may further query the SCC in case any more metadata or execution parameters are updated since the onboarding, and for any environment variables not included in the package, during the deployment.

The SCC is also searchable, by making search queries based on stored metadata. The RAF component can identify the proper SCs by searching for SCs and correlating with the outcome of its own analysis. The SCC search and fetch functionality is also used by the Security and the Accounting Dashboards, so as to leverage the UI of the Portal, to enact the deployment of SCs or to have an inspection of the overall deployment, through a combination with the interfaces of the SO and the Billing components. Finally, listing and searching for SCs is enabled in the SC Marketplace UI in the dashboard, by means of the SCC functionality.

2.3.1. SCC design and interaction

The design of SCC enables interactions with many PALANTIR components, by exposing functionalities to them and interacting with the SO when it comes to onboarding. The main goals of SCC are:

- To ensure the integrity of the SC when registering and onboarding.
- To provide search functionality, facilitating the filtering out and selecting of SCs, either by users of the platform, or by other PALANTIR core components.
- To serve the complete set of metadata and accompanying details regarding a SC package to users and other core components.

As such, the SCC has been designed based on the foreseen interactions. The main functionality of the SCC is exposed as an API that is consumed by other PALANTIR components. The most important interactions are between the SCC and the SO, where one completes the other. Most operations which involve the SCC also engage the main types/roles of PALANTIR users, and therefore the SCC is also heavily tied to SC-related views on the Portal. Other core components are involved in the SC-related views, and use some functionality of the SCC in order to complete their tasks. All the aforementioned entities, their interfaces and expected interactions with other PALANTIR components are to be elaborated upon in D3.2.

The SCC itself is broken down into modules based on the pieces of functionality that comprise it. The diagram of Figure 3 signifies the different interactions between the SCC modules, and the direction of the flow of data for each one is signified by the direction of the arrows. Note that some interactions are bi-directional, as data is not fed only to one side. Also note that other PALANTIR components, as consumers of SC data, may invoke the SCC search or SC descriptors and metadata functionality by means of the exposed REST API, so as to carry out their functionality. Finally, for the onboarding process, the SCC interacts with the API of the SO by means of a REST client.

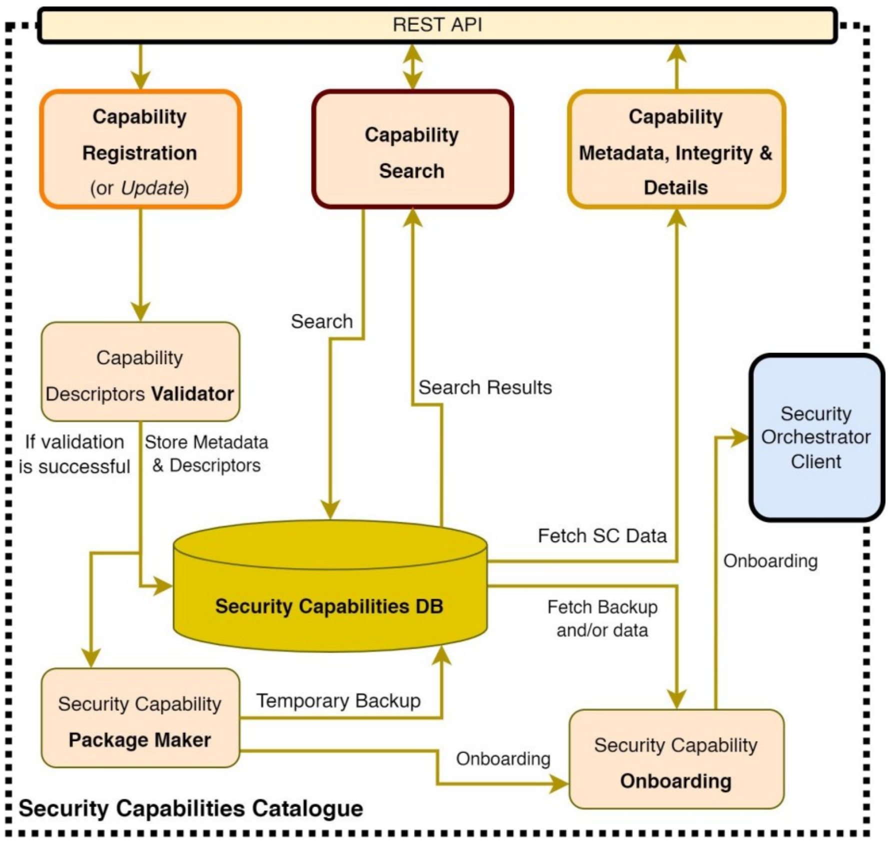
REST API
Capability Registration (or Update)
Capability Search
Capability Metadata, Integrity & Details
Capability Descriptors Validator
Search
Search Results
Security Orchestrator Client
If validation is successful
Store Metadata & Descriptors
Fetch SC Data
Onboarding
Security Capabilities DB
Fetch Backup and/or data
Security Capability Package Maker
Temporary Backup
Onboarding
Security Capability Onboarding
Security Capabilities Catalogue

Figure 3: SCC architecture and interactions diagram.

The rest of this section is organised to explain the multiple aspects included in Figure 3, presenting the overall workflow and explaining the functionality per module.

2.3.1.1. SCC general workflow

The SCC interacts with multiple components. However, the kinds of interactions made available by the SCC are homogenised. The REST API allows for authenticated users of the platform to interact with it, as well as core components. A SC developer can view the status of their already registered SCs, and can sort through them by searching them. The most important purpose for the SC developer, however, is to register a new SC. The catalogue then proceeds to verify the validity of the input metadata and descriptors. Once the input data is verified, the catalogue stores a new SC entry in its database, and proceeds to also create a SC package with the VDU and every descriptor and manifest included. The SC entry includes the version, the security descriptors and the operational VNFDs, and will be used by other components and user types for search and inspection purposes and is kept in the DB along with a reference to the SC package. The SC package is temporarily stored in the SCC's object storage until it is onboarded on the orchestrator. Once the onboarding of the package is complete, the SC is finally deployable by the SO.

In the meantime, other core components and end-users can use the SCC API to search for SCs that match some desired criteria, as they are defined in the SC entry. When a search query is done, it shows the matching results and includes some basic information, while also yielding the link to the complete SC inspection for each match. Finally, SCs can be inspected by querying one specific SC in the REST API. The status of the SC is included in the response, along with all descriptors, security-enhancing metadata, privacy descriptors and integrity metadata. By exposing SCs as resources in this way, each of the authorised PALANTIR components can inspect the current capabilities of the platform, ensure that each capability is trusted, and select the appropriate capabilities for a monitoring, protection or remediation purpose specific to a set of circumstances.

The access to specific data and operations are dependent on the role of the user, in the case when PALANTIR Dashboard accesses the SCC, or to the type of PALANTIR component that accesses it. The user or entity that creates or makes changes to entries in the SCC is always authenticated and stored in the entry itself, and access to SCC entries is restricted only to authenticated users and trusted PALANTIR components. The authentication happens on the API level and is based on the users and roles management that the PALANTIR platform uses.

2.3.1.2. Security Capability Registration (SCR)

This module ensures that the inserted SC description is correctly formulated and complete, and creates a validation task. The validation carried out in this module is only meant to happen on a basic level, to ensure the data validity and determine whether the SC's version is the initial or an updated one. It also aids in performing a validation on the transient image, to ensure its source and type are valid, so that it can be downloaded and validated as well. The SC registration is a service available only to the PALANTIR Dashboard component, and only for the SC developer user role, through the SCC REST API as an endpoint. Once this procedure finishes, a basic entry is added to the SCC's database, and the state of the SC is set to "pending validation".

2.3.1.3. Security Capability and Descriptors Validation (SCDV)

This module downloads the image of the SC software and validates the descriptor values. The digest of the image is matched with the provided value, and any available descriptors of the image, such as its name or ID, are matched as well with the ones provided. If all these steps pass, then the validator prepares the inserted metadata as a new entry in the SC Database, and the entry is stored in it. In the meantime, the SC metadata for the manifests are prepared and fed to the SC Package Maker, which completes the creation of the SC package. It then stores in the SC Database the internal reference to the package created by the Package Maker and the way to address the SC on the SO upon onboarding. This whole process runs as a background task, once the basic registration is done. Once this process is complete, the state of the SC is set to "validated". If an error occurs, or a mismatch is found, then the whole operation is aborted, and the SC is deleted from the database.

2.3.1.4. Security Capability Package Maker (SCPM)

The Package Maker creates the SC package as a ".tar.gz" file that is compatible with the SCO, and temporarily stores it in the SC object store. This file includes the image of the software of the SC, along with the descriptors, manifests and attestation data coming from the validator. Its primary concern is ensuring the compression and temporary storage of the SC package file. The package is then sent to the onboarding module to complete the registration and onboarding process. This whole process is executed as a background task, once the validation is done. Once this process is complete, the state of the SC is set to "created".

2.3.1.5. Security Capability Onboarding (SCC-O)

This module completes the onboarding process, and for that, it uses a client that consumes the API of the SO. It sends the package to the SO, and waits for the event that the SC is finally onboarded. Once the SO notifies that the onboarding is complete, the status of the SC changes to "onboarded", which means that the SC can be deployed and used. Once confirmed, the temporary copy of the package is marked for removal from the SCC after a short period. The SCC Onboarding process runs as a background task and ensures that the onboarding process on the side of the SO goes smoothly before proceeding with the cleanup and finalisation. Once it is done, the SC entry includes the link to instantiate the SC in the SO, and any complimentary metadata required, to deploy an instance.

2.3.1.6. Security Capability Search (SCS)

This module creates the necessary search query to the SC Database, based on the descriptor fields and values or ranges that are provided for each field. This service is accessible from the outside of the SCC as a REST endpoint. It ensures that the entity that makes this query, either an end-user or a PALANTIR component, is authenticated based on the restrictions applied in the REST API level. It also ensures that the search is appropriately constrained based on the role used when interacting with the API. Based on parameters and user roles, the results of the queries that are enacted can serve a lot of different purposes in other PALANTIR components, as the search can take into account many combinations of parameters, and return only the matching SCs registered in the SCC.

2.3.1.7. Security Capability Metadata Access (SCMA)

This module serves the Security Capability's complete set of metadata, integrity and privacy descriptors, and other accompanying details such as status, to users or components with the appropriate access. This service is accessible from the outside of the SCC as a REST endpoint, and access constraints and rules apply here in a similar manner as in the search module. The inspection of a SC is needed in various situations by other PALANTIR components, either to inspect a SC, select it, ensure its integrity or instantiate it through the SO.

2.3.2. Changes from D2.1

Since D2.1, there have been some changes to the overall design of SCC. The motivation for such change is to better account for the foreseen interactions with other components, and the way PALANTIR users will interact with the SCC, depending on their role. The previous sections explain these changes, and Figure 3 shows an interaction diagram that captures all of them.

Furthermore, the nature of SCs has been revised by the PALANTIR consortium, and as such the decision was made to not include SDN flows or P4 programs as SCC entries. Instead, any functionality is packaged by means of dynamically deployable VNF software, the accompanying SEM, and any environment variables and parameters the SC and the included VNF might require. The SCC does not hold any information that might directly affect network hardware, and instead serves necessary data and metadata about SCs and onboards the resulting package to the SO. Further actions upon the client SME network are done by the deployed SCs, which the catalogue helps the SO instantiate and the security developer to define and maintain, or the SO itself.

2.4. Risk Analysis Framework

The Risk Analysis Framework (RAF) is the PALANTIR component that is responsible for the interaction with the SMEs, so that it can provide an analysis of SME risk based on business profiling, asset profiling and asset vulnerabilities assessment.

RAF provides a simplified and comprehensive view of risk management or risk assessment for use within SMEs. The philosophy of this framework is to be suitable for collecting information even from non-experts and avoid obfuscating workflows for the information entry.

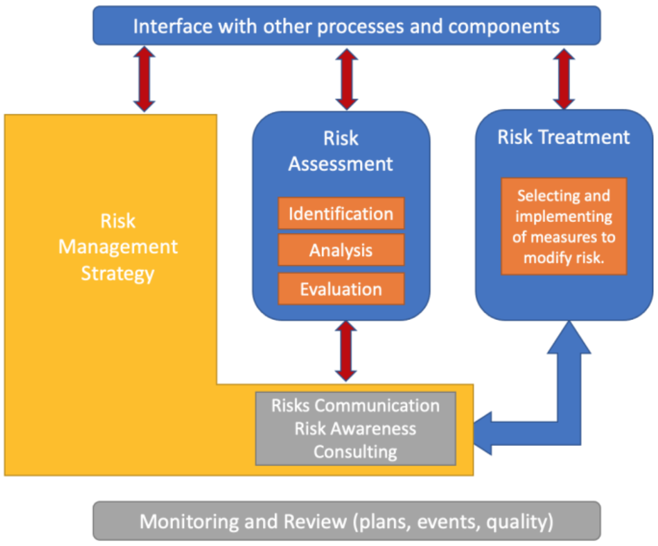
Interface with other processes and components
Risk Management Strategy
Risk Assessment
Identification
Analysis
Evaluation
Risk Treatment
Selecting and implementing of measures to modify risk.
Risks Communication
Risk Awareness
Consulting
Monitoring and Review (plans, events, quality)

Figure 4: simplified ENISA Risk Management Framework.

2.4.1. Workflow and phases

The RAF design and workflows based on the ENISA Risk Management process, in a simplified manner that makes it fit for small SMEs and self-employed professionals [2]. Figure 4 provides the overview of the ENISA Risk Management process.

The block representing the Risk Management Strategy is not considered a part of RAF, as it is more related to company practices for risk management (which is out of scope for the project). The main focus of RAF is the Risk Assessment block, which was simplified in order to be applicable for small SMEs and non-expert professionals on maintaining a certain risk strategy. The Risk Treatment block is jointly shared between different components of the PALANTIR framework. The part that is considered for RAF is the selection or suggestion of measures to modify risk. The monitoring and review block will be implemented as a workflow in the project, in order to allow SMEs risk and security awareness to be monitored.

The elements to be implemented for PALANTIR are the Risk Assessment block, the interface with other PALANTIR framework components that require output from RAF, as well as some consulting and best practices related to the identified asset vulnerabilities and risk management processes.

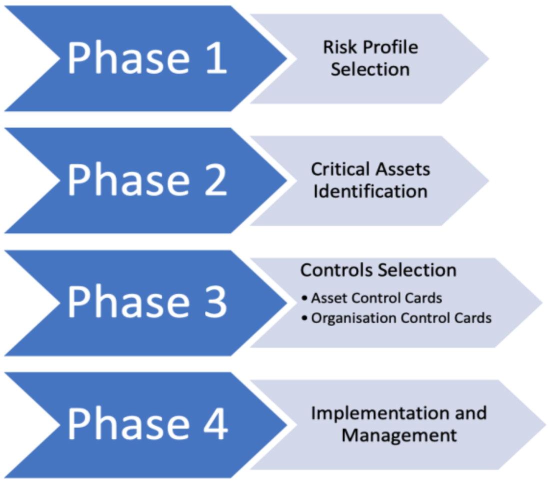

Figure 5: summary of the four phases in RAF.

The phases anticipated for RAF remain as described in D2.2 and are depicted in Figure 5.

The RAF design and workflows is based on the ENISA Risk Management process, in a simplified manner that makes it fit for small SMEs and self-employed professionals [2]. Figure 4 provides the overview of the ENISA Risk Management process.

- Phase 1: Risk Profile Selection

  During this phase, the SME expert provides information related to four risk areas: i) legal and regulatory; ii) productivity; iii) financial stability; and iv) reputation and loss of customer confidence. The classified risks are high, medium and low for each area.

- Phase 2: Critical Assets Identification

  This phase contains two sub-phases, namely a) Assets Identification and b) Security Requirement identification per asset category. During the first phase the SME expert, in collaboration with PALANTIR experts, will identify the SME assets and classify at the following categories: i) systems; ii) network; iii) people; and iv) applications. This identification will allow further mapping to known attacks and vulnerabilities. The assessment team determines what is important to the organisation (e.g., information-related assets) and selects those assets that are most important to the organisation, also referred to as critical assets.

  During the second part of this phase, the risk assessment teams from both the PALANTIR and the SME need to recognise their security requirements. The considered security requirements are: (i) confidentiality, (ii) integrity, and (iii) availability.

  The output of this process will be a table of critical assets classified by asset categories and a list of corresponding security requirements, together with justification or supporting information considered during the evaluation. Below is a simple exemplary table for a medical practitioner case.

Table 1: sample output for Phase 2, for the case of a medical practitioner.

Critical Asset	**Asset Category**	**Components**	**Security Requirements**	**Rationale for selection**
Doctor Patient Registry	Application	Database	Confidentiality Availability	This application is essential for the Doctor day-to-day business operations. In addition, data available in the database are considered private

- Phase 3: Controls Selection

  Based on the profile categories (conducted in Phase 1), Phase 3 follows a three-step approach:

  - Step 1: Select Organisation Control Cards

    Analysis teams select organisational control cards (see annex C of [2]) for the risk areas identified during phase 1 (Risk Profiling) and thereby define the direction for information security efforts in the organisation.

  - Step 2: Select Asset Based Controls

Based on the risk profile and the asset security requirements, SME analysis teams can use the Asset Control Cards Table (see annex B of [2]) to identify the appropriate asset controls.

   - Step 3: Document List of Selected Controls and Rationale

     This step covers the documentation for the rationale behind selecting each control card and the necessary actions for implementation.

- Phase 4: Implementation and Management

  During this phase, the analysis team identifies actions and recommends an action list, setting forth the direction for security improvement. This phase sets the mitigation plans, then applies monitoring and control.

  This phase allows for monitoring the security KPIs overseeing the implementation of the results of the evaluation. PALANTIR, being a holistic framework, takes into consideration the usage of asset control cards and provides means for implementation of mitigation for limiting the risk. The parts that fall on the organisational control cards recommendations are not part of the automate mechanisms offered by PALANTIR.

2.4.2. Changes from D2.1 and D2.2

The specifications of RAF refine and expand the original view shared in D2.1 and D2.2. In summary, the specifications here provided indicate how the software components handle the interactions, the maintenance of information and the processing required in order to provide automation in the risk assessment for SMEs. Such specifications lead to a refinement and extension of the original view shared in D2.1 and D2.2, regarding the initial decision for adopting the ENISA Risk Management Framework. Parts of the risk management that belong to the organisation management and policy enforcement are not considered as automation targets.

3. Specifications

This section proceeds to map the PALANTIR requirements, provided in D2.1, as technical specifications that map to each of the WP3 components.

3.1. SecaaS

PALANTIR will deliver Security-as-a-Service through the implementation and deployment of SCs. Initially, two types of SCs will be addressed: monitoring and remediation. The first one is to monitor and analyse the network traffic, usage resources, operating system calls, etc., and to generate alerts or reports upon the detection of any anomalous event. The second type initiates after the malicious activity is produced and intends to act against this attack and avoid severe damage to the infrastructure.

The SCs targeted for implementation during the first release are: an Intrusion Detection System (IDS), a Deep Packet Inspection (DPI), a Network NetFlow Sniffer (NNS), a Lightweight Shipper for Logs (L-SL), a Firewall and Router (FW), a Backup (BUp) and a Web-based Traffic Analysis (WTA) solution. Table 2 indicates the **requirements mapping** for each SC, taking into account the list presented in D2.1. For each SC, a column per SC indicates whether it addresses the specific requirement.

Table 2: technical specifications for SecaaS.

Req. ID	Requirement description	IDS	DPI	NNS	L-SL	FW	BUp	WTA
R1.3.1	The platform SHALL be able to instantiate security capabilities.	✓	✓	✓	✓	✓	✓	✓
SC_S1	The SCs shall be deployed as instances into the platform.							
R1.3.2	The platform SHALL be able to configure security capabilities, whether already deployed or newly instantiated.	✓	✓	✓	✓	✓	✓	✓
SC_S2	The SCs shall be configured where needed, being deployed or before instantiating.							
R1.3.3	The platform SHALL provide a variety of SecaaS packages on the Catalogue.	✓	✓	✓	✓	✓	✓	✓
SC_S3	The variety of SCs shall be allocated as packages into the SCC.							
R1.3.4	The security capabilities SHALL provide the maximum feasible set of the expected (open) connectors so as to be handled similarly in the catalogue and interact in a similar manner with the orchestration tools.	✓	✓	✓	✓	✓	✓	✓
SC_S4	The SCs shall provide the connectors needed to interact with the different components (SCO, SCC, etc.)							
R1.3.5	The security capabilities SHALL provide the privacy specifications that are shown to infrastructure administrators that ultimately deploy such services.	✓	✓	✓	✓	✓	✓	✓

SC_S5	The SCs shall offer the privacy specifications required by the administrator before their instantiation takes place.							
R1.3.6	The security capabilities SHALL implement the expected (open) Element Management System (EMS) hooks so as to be configured by the platform.	✓	✓	✓	✓	✓	✓	✓
SC_S6	The SCs shall implement the Element Management System needed to interact with the platform.							
R1.3.7	The security capabilities SHALL be uploaded to the catalogue as a pre-packaged bundle containing its basic dependencies. Any external dependency SHALL be provided before its uploading to the Catalogue.	✓	✓	✓	✓	✓	✓	✓
SC_S7	The SCs shall incorporate all external dependencies before being uploaded as a pre-packaged bundle into the SCC.							
R1.3.8	The security capabilities SHOULD be available in source form and publicly shared so as to allow reusing by others as well as logic auditing.	✓	✓	✓	✓	✓	✓	✓
SC_S8	The SCs should be publicly shared in source form for reusing and auditing purposes.							
R1.3.14	The platform SHALL be able to retrieve the basic status for the security capabilities instantiated or available (in the Catalogue).	✓	✓	✓	✓	✓	✓	✓
SC_S9	The SCs shall provide their basic status where needed by the platform.							
R1.3.19	The platform SHOULD deliver adaptive filtering and traffic control capabilities.	✓	✓	✓	--	✓	--	✓
SC_S10	The SCs should deliver adaptive filtering and traffic control functionalities.							
R1.3.20	The platform SHOULD deliver port and service scanning capabilities.	✓	--	--	--	--	--	--
SC_S11	The SCs should deliver port and service scanning functionalities.							
R1.3.21	The platform SHOULD deliver remote attack detection capabilities.	✓	✓	--	--	✓	--	✓
SC_S12	The SCs should deliver remote attack detection functionalities.							
R1.3.22	The platform MUST provide protection from data exfiltration attempts.	✓	✓	--	--	✓	--	✓
SC_S13	The SCs must provide protection from data exfiltration attempts.							

R1.3.23	The platform SHOULD offer packet inspection capabilities.	✓	✓	--	--	--	--	--
SC_S14	The SCs should provide DPI functionalities.							
R1.3.24	The platform SHOULD deliver intrusion detection and prevention capabilities.	✓	✓	--	--	✓	--	✓
SC_S15	The SCs should deliver IDS functionalities.							
R1.3.25	The security capabilities MAY implement techniques such as exact data matching, structured data fingerprinting, statistical methods.	✓	✓	--	--	✓	--	✓
SC_S16	The SCs may develop different mechanisms to provide data security and statistical methods.							
R1.3.26	The platform SHOULD deliver additional security capabilities in function of specific use cases tasks.	✓	✓	✓	✓	✓	✓	✓
SC_S17	The SCs should implement different functionalities to cover the use cases tasks.							
R1.3.29	The platform SHOULD prevent and react against Ransomware attacks.	✓	✓	--	--	✓	--	✓
SC_S18	The SCs should deliver prevention and reaction ransomware functionalities.							

As a general comment to the mapping of requirements, the SCs cover the complete list of T3.1-related requirements presented in D2.1. In this sense, different aspects can be highlighted. For instance, the IDS SC is the only one addressing R1.3.20, but as indicated in section 4.1, one implementation software can develop different SCs, giving real-time capabilities to detect and mitigate anomalous activities detected. In other words, if specific software includes IDS and FW capabilities, this software will detect and implement mitigation mechanisms (*e.g.,* firewall rules) in a faster way. On the other hand, some of the listed requirements are handled by all proposed SCs because, in general, these express characteristics that must be met or managed. As per changes introduced since D2.1, R1.3.29 has been included since this requirement should be addressed with the collaboration of some of the presented SCs.

3.2. Security Orchestrator

The SO comprises all modules related to the orchestration and life-cycle management of SCs, along with its monitoring. Given the central position it occupies in the architecture, it also provides ancillary functionality to other components and subcomponents within WP3 and WP4.

Table 3 describes how the requirement maps to the technical specification that is to be fulfilled in the SO subcomponent, as well as the internal modules that contribute to the specification.

Table 3: technical specifications for SO.

Req. ID	**Requirement description**	**AAC**	**API**	**ATR**	**CFG**	**LCM**	**MON**	**PKG**	**POL**

R1.2.6	Security mechanisms used in a complex cybersecurity eco-system SHALL be able to identify, distribute and allocate responsibilities between 5G ecosystem stakeholders.	√	√	--	--	√	--	√	√
SO_S1	The SO shall be able to transmit the relevant mitigation requests to the requested SCs, disregarding their origin.								
R1.3.2	The platform SHALL be able to configure security capabilities, whether already deployed or newly instantiated.	√	√	--	√	√	--	--	--
SO_S2	The SO shall be able to configure SC instances from the MSPL configuration received within the PALANTIR platform. Furthermore, if the requested SCs are not running, the SO shall previously instantiate them.								
R1.3.7	The security capabilities SHALLbe uploaded to the catalogue as a pre-packaged bundle containing its basic dependencies. Any external dependency SHALL be provided before its uploading to the Catalogue.	√	√	--	√	--	--	√	--
SO_S3	The SO shall handle the external dependencies required by the SCs in such a way to coordinate the needed actions among other involved components or tools.								
R1.3.9	The platform SHOULD be able to monitor the deployed security capabilities and expose such data through programming interfaces for other internal components.	√	√	--	√	--	√	--	--
SO_S4	The SO should monitor the SC instances and expose the fetched metrics and telemetry data through programmable interfaces, accessible to other components.								
R1.3.10	The platform SHALL be able to deploy security capabilities from the Catalogue to operate with a copy of network data (off-the-path traffic).	√	√	--	√	√	--	√	--
SO_S5	The SO shall instantiate SCs in a way they can process network data in an offline manner (*e.g.,* through logs).								
R1.3.11	The platform MAY be able to deploy security capabilities from the Catalogue to operate with online network data (on-the-path traffic).	√	√	--	√	√	--	√	--
SO_S6	The SO may instantiate SCs in a way they can process live network data (*e.g.,* through the usage of mirrored interfaces).								
R1.3.14	The platform SHALL be able to retrieve the basic status for the security capabilities instantiated or available (in the Catalogue).	√	√	--	√	√	--	√	--
SO_S7	The SO shall expose the list of both available SC packages and the SC running instances through programmable interfaces, accessible to other components. It shall also expose the particular status and useful details of the SC running instances.								
R1.3.15	The platform SHOULD be able to decide whether to reuse existing security capabilities or if new ones have to be instantiated, according to the received policy specifications.	--	--	--	√	√	√	√	--
SO_S8	The SO should identify appropriate running instances of SCs in order to use and configure								

	according to policies, or otherwise instantiate new SCs.								
R1.3.16	The platform SHOULD implement SDN technology in conjunction with NFV technology to facilitate the provision of SecaaS solutions.	--	--	--	✓	✓	--	✓	--
SO_S9	The SO should configure proper rules on SDN devices as required by a component or its internals (*e.g.,* to perform SDN-based port mirroring on specific SC instances) where needed to assist its correct behaviour.								
R1.4.1	PALANTIR SHOULD deploy mechanisms for the periodic attestation of the platform and the running applications', services' and configurations' integrity.	--	--	✓	--	✓	--	--	--
SO_S10	The SO should provide the available specific runtime and environmental data for its usage during part of the attestation process.								

3.3. Security Capabilities Catalogue

The SCC is composed of modules related to the storage, onboarding, searching, and accessing data about SCs. The SCC provides functionality to other components and subcomponents within other WPs, and contributes to the actualisation of functionality related to subcomponents of the SCO.

Table 4 describes how each requirement maps to the technical specification that the SCC subcomponent must filfill, as well as the internal modules that contribute to the specification.

Table 4: technical specifications for SCC.

Req. ID	**Requirement description**	**SCR**	**SCDV**	**SCPM**	**SCC-O**	**SCS**	**SCMA**
R1.3.1	The platform SHALL be able to instantiate security capabilities.	✓	✓	✓	✓	--	--
SCC_S1	In order for the SO to instantiate SCs, the SC shall first need to be registered and onboarded to the SO, which is done through the SCC.						
R1.3.2	The platform SHALL be able to configure security capabilities, whether already deployed or newly instantiated.	--	--	--	--	✓	✓
SCC_S2	The platform shall provide the necessary metadata view and search for SCs, in order for the SO to provide the necessary configuration options.						
R1.2.6	Security mechanisms used in a complex cybersecurity eco-system SHALL be able to identify, distribute and allocate responsibilities between 5G ecosystem stakeholders.	✓	--	--	--	✓	✓
SCC_S3	The SCC shall be able to store the relevant mitigation capabilities of SCs, as well as search for SCs matching them, and show them for a requested SC, disregarding their origin.						

R1.3.3	The platform SHALL provide a variety of SecaaS packages on the Catalogue.	✓	✓	✓	✓	✓	✓
SCC_S4	The SCC shall provide the ability to register and onboard SecaaS packages as SCs, search for appropriate SCs based on metadata, and view their properties.						
R1.3.7	The security capabilities SHALL be uploaded to the catalogue as a pre-packaged bundle containing its basic dependencies. Any external dependency SHALL be provided before its uploading to the Catalogue.	✓	✓	✓	✓	--	--
SCC_S5	The SCC shall provide a registration and onboarding endpoint, which is used by a SC developer through the Portal. There, the developer will be able to add all metadata for the SC, while the software and dependencies to be uploaded are all pre-packaged as an image for VMs or containers.						
R1.3.10	The platform SHALL be able to deploy security capabilities from the Catalogue to operate with a copy of network data (off-the-path traffic).	--	--	--	✓	--	✓
SCC_S6	The SCC shall provide the metadata and descriptors for SCs that may operate on off-the-path-traffic, and shall be able to onboard those SCs on the SO.						
R1.3.11	The platform MAY be able to deploy security capabilities from the Catalogue to operate with online network data (on-the-path traffic).	--	--	--	✓	--	✓
SCC_S7	The SCC shall provide the metadata and descriptors for SCs that may operate on on-the-path-traffic, and shall be able to onboard those SCs on the SO.						
R1.3.14	The platform SHALL be able to retrieve the basic status for the security capabilities instantiated or available (in the Catalogue).	--	--	--	--	✓	✓
SCC_S8	The SCC shall provide all metadata and descriptors for any available SC, as well as search functionality based on some of these parameters.						
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	--	--	--	--	✓	✓
SCC_S9	The SCC shall provide the necessary security enhancing metadata for the SCs, as well as the necessary search capability based on these parameters, so that TI or any other component or user is able to find the appropriate SCs in order to mitigate attacks.						
R1.4.1	PALANTIR SHOULD deploy mechanisms for the periodic attestation of the platform and the running applications', services' and configurations' integrity.	--	--	--	--	✓	✓

SCC_S10	The SCC should provide the necessary integrity metadata for SCs, as part of the search and metadata access functionalities; so that the AE can perform periodic attestations on the deployed SCs.						
R1.2.7	The PALANTIR eco-system SHALL be able to publish security KPI measuring the compliance of stakeholder with their Security Level Commitments.	--	--	--	--	--	✓
SCC_S11	The SCC shall provide the security metadata of a SC, as part of the metadata access, in order to be able to measure the compliance KPIs.						

3.4. Risk Analysis Framework

Table 5 describes how each requirement maps to the technical specification that the RAF component must achieve through its internal proceses.

Table 5: technical specifications for RAF.

Req. ID	**Requirement description**	**Origin of requirement**
R1.2.7	The PALANTIR eco-system SHALL be able to publish security KPI measuring the compliance of stakeholder with their Security Level Commitments.	*All use cases.*
RAF_S1	Based on the suggested mitigations and best practices, the RAF will allow monitoring on the establishment new practices of each SME as well as use available incident reports to formulate the Security Level Commitment KPI.	
R1.2.10	The PALANTIR platform SHOULD provide risk profiling and assessment for a set of input attack surfaces provided from the corresponding stakeholder.	*All use cases.*
RAF_S2	The RAF will identify all assets related to the SME business and infrastructure in order to automatically retrieving vulnerabilities from asset databases and associated attack vectors (*e.g.,* a firewall operating with firmware with known vulnerabilities) and will assimilate this information to the resulting risk profiling.	
R1.2.11	The PALANTIR platform SHOULD support management capabilities for the vulnerabilities of the system under test.	*All use cases.*
RAF_S3	System under test is considered to comprise the total or part of the infrastructure assessed. Through control cards and via the collaboration of external to RAF components, the RAF will provide means to manage the vulnerabilities (*i.e.,* provide guidelines and best practices to deploy security services).	

R1.2.12	The PALANTIR platform SHOULD provide coherent mitigation plans for corresponding threat and attack vectors.	*All use cases.*
RAF_S4	The RAF will implement both an (i) asset vulnerabilities and attack vector database and (ii) associated best practices list, for various SW and HW components related to certain functions and operations. Based also on the SCs supported by PALANTIR, it will be able to provide mitigation plans.	
R1.3.31	The platform SHOULD provide a service supporting risk assessment framework	*End-user Questionnaire: question 21.* *Technical Questionnaire: question 7.* *All use cases.*
RAF_S5	Exploiting the PALANTIR SME dashboard, the RAF will implement an intuitive user interface in order to acquire SME information required for the risk assessment in two phases: one for the profile risk assessment and second for the asset identification and risk assessment. At the end, a unified risk level will be calculated for the SME.	
R1.4.5	The platform SHOULD provide a solution to deliver an incident response plan tailored to specific end-users.	*End-user Questionnaire: question 32, 33 and 34.* *All use cases.*
RAF_S6	Based on the assessed risk level, identified assets and available SecaaS solutions. Specific incident response plans will be created and communicated to the SME.	

4. Implementation

This section covers the implementation details of the different subcomponents within WP3. Information such as specific technologies and techniques will be indicated, along with their application to each of the components and subcomponents developed.

4.1. SecaaS

This section describes the implementation details of the SCs identified so far in the PALANTIR context. The different software tested and used by each SC is presented, as well as possible customisations and modifications that will be performed to change the base behaviour to adapt such SC to the PALANTIR needs. The list of developed software is susceptible to changes in D3.2.

The first and foremost functionalities offered by SEM are the monitoring capabilities. Within PALANTIR, T3.1 has picked the open-source FileBeat [3] log shipper as the primary data and log aggregator. In order to avoid technology lock-in, but especially to guarantee compatibility with multiple deployment models (cloud, edge, and on-premise), the SCs here referenced have been deployed as Docker containers, LXC containers, and QEMU virtual machines.

4.1.1. Intrusion Detection System (IDS)

There are two primary threat detection techniques: signature-based detection and anomaly-based detection. The choice of a specific technique is important when selecting a detection engine. Different open-source software has been tested to implement an IDS as part of the PALANTIR project. At this time, additional functionality has not been added to the original software, apart from possible detection and prevention rules that can be incorporated against specific attacks.

The currently considered list of software included for IDS is as follows:

- Snort (version 2) [4]: one of the most widely used open-source IDS, based on signatures, since this software functions with the incorporation of matching rules to detect known threats.
  Requirements: 64-bit amd64 (x86-64) compatible CPU, 1GB RAM, 8 GB or larger disk drive.
- Suricata [5]: just like Snort, Suricata is another open-source IDS with similar characteristics. This software encompasses detection, prevention and monitoring capabilities and has been designed for simple deployments.
  Requirements: 64-bit amd64 (x86-64) compatible CPU, 1GB RAM, 8 GB or larger disk drive.
- Wazuh [6]: it is defined as an "Open Source Security Platform" since it incorporates extra functionalities, when compared with the previous ones. Some of the offered capabilities are the Security Analytics, Intrusion Detection, Log Data Analysis and Configuration Assessment. Thus, Wazuh can be understood as an integrated tool composed of interesting security mechanisms and solutions.
  Requirements: 64-bit amd64 (x86-64) compatible CPU, 16GB RAM, 100 GB or larger disk drive

4.1.2. Deep Packet Inspection (DPI)

Understanding DPI as a particular case of IDS, the software presented in the above section could be used to implement such a SC. Besides, the nDPI toolkit, maintained by ntop (explained below), can analyse traffic flows, obtaining specific information such as metadata and application protocols. It is worth mentioning that the nDPI [7] libraries are integrated into other tools into the ntop website [8], nProbe [9], PF_RING [10], n2disk [11], nBox [12] and nScrub [13]. All this software is susceptible to being implemented in the PALANTIR project if required at later, more mature stages.

Requirements: 64-bit amd64 (x86-64) compatible CPU, 1GB RAM, 8 GB or larger disk drive.

4.1.3. Netflow Network Sniffer (NNS)

A functional NNS has not been tested at this time, and therefore it is not possible to provide examples of this SC. However, other complementary SCs (like a firewall) could also offer the network data sniffing functionality meanwhile this SC is implemented. In the meantime, we suggest using Docker image `networkstatic/nflow-generator` with the command

```
# /go/bin/nflow-generator -t COLLECTOR_IP -p COLLECTOR_PORT
```

to generate randomised Netflow for testing purposes (*e.g.,* data flows, stress tests, among others) until the delivery of the NNS SC.

4.1.4. Lightweight Shipper for Logs (L-SL)

FileBeat allows watching for dynamic information (such as network events or log streams, among others) to parse and send this information as JSON and funnel it through an output module towards either the Kafka message broker or an Elastic stack. Compatible outputs are: Elasticsearch Service, Elasticsearch, Logstash, Kafka, Redis, File and Console. For example, it is possible to enable the corresponding input module [14] for processing and shipping SNORT-related events and alerts.

To provide a specific example, when processing a Netflow record, FileBeat will generate a JSON similar to those available in the official repository available in GitHub [15].

Requirements: 64-bit amd64 (x86-64) or 32-bit CPU, 1GB RAM, 8 GB or larger disk drive.

4.1.5. Firewall and Router (FW)

The firewalls offer the functionalities located at the centre of any security management of the SMEs infrastructure, and they might come in the form of dedicated hardware appliances or virtual services. In the context of SecaaS, it is worth mentioning that both dedicated firewall-only services are a valid alternative, although suites that are more complete are preferable. Within T3.1, the consortium opted for implementing firewalling services with full networking and security capabilities.

- Virtual routers:
  - pfSense [16] offers full routing capabilities with a plugin catalogue for other security services such as VPN, traffic inspection and monitoring, firewall, NAT and vLANs, among others.
    Requirements: 64-bit amd64 (x86-64) compatible CPU, 1GB RAM, 8 GB or larger disk drive.
  - OPNsense [17] is a pfSense fork with improved GUI and shorter firmware update cycle.
    Requirements: 1 GHz dual core cpu, 2GB RAM, 4 GB or larger disk drive.
- Virtual firewall:
  - iptables is a user-space filtering control that allows the enforcement of stateless and stateful rules for several protocols. It also supports NATting features which will not be used in PALANTIR.
  - Requirements: 64-bit amd64 (x86-64) compatible CPU, 1GB RAM, 8 GB or larger disk drive.

4.1.6. Backup (BUp)

Although most of the SMEs should already have in place a backup solution, we explored the capabilities offered by TrueNAS [18] (previously FreeNAS) to provide a network storage solution. However, since the implementation of a shared storage solution requires precise configuration and deployment scenario analysis, we advise against automatic configuration and generic deployment that do not take into account the SMEs' requirements and needs (*e.g.,* storage capacity, raid configuration, access and firewalling, among others). By default, TrueNAS can be configured with Windows (SMB), Unix-like (NFS) and Bloch Shares (iSCSI), respectively, to be configured with different technologies and operating systems.

Requirements: 64-bit CPU, 16 GB RAM, 16 GB boot drive.

4.1.7. Web-based Traffic Analysis (WTA)

The ntopng [19] tool is the next generation version of the original ntop, a network traffic probe that monitors network usage. In its current form, it runs as a plugin for OPNsense and it offers API access.

Requirements: 64-bit amd64 (x86-64) or 32-bit CPU, 1GB RAM, 8 GB or larger disk drive.

4.1.8. Virtualised Scenario for Use Case 1

In order to demonstrate the behaviour of different SCs and provide the first demo about the Use Case 1 proposed on PALANTIR, we have designed a practical network topology composed of an attacker, a clinical environment, a SC, a virtual firewall, Kafka software and a Netflow generator.

As Figure 6 shows, it divides the scenario into the PALANTIR internal network and the SME internal network. In the first one, the PALANTIR-related tools are deployed, finding the SC with Snort and SEM (FileBeat software) incorporated. Besides, Kafka has a software communication link configured with the SC to collect its logs and pass them to the other PALANTIR components. The second scenario considers an attacker that has gained access to the network and can perform specific attacks to produce different malicious results. On the other hand, the clinical environment is composed of a MySQL DB with patient records and a dashboard integrated into the OpenEMR [20] tool.

The execution flow of this first demonstration scenario has different steps:

1) The attacker initiates a Denial-of-Service attack addressed to the SC;
2) The IDS SC (Snort) detects the attack and generates different logs which are collected by FileBeat and processed to modelling in JSON format;
3) FileBeat sends the processed logs through the specific Kafka topic; and
4) The data are available in PALANTIR platform and it is then possible to perform actions configured upon detecting an attack (such as sending an alert to SME client, creating a notification in the Portal, etc).

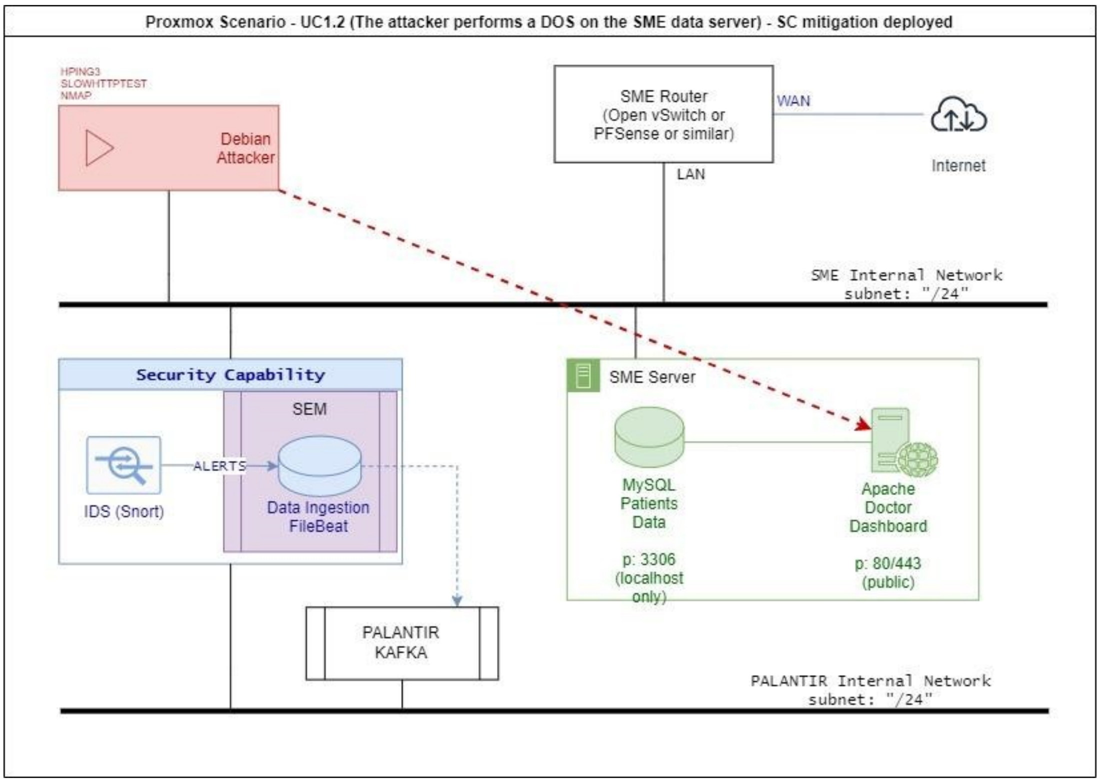
Proxmox Scenario - UC1.2 (The attacker performs a DOS on the SME data server) - SC mitigation deployed
HPING3
SLOWHTTPTEST
NMAP
Debian Attacker
SME Router (Open vSwitch or PFSense or similar)
WAN
Internet
LAN
SME Internal Network subnet: "/24"
Security Capability
SEM
ALERTS
IDS (Snort)
Data Ingestion FileBeat
SME Server
MySQL Patients Data
p: 3306 (localhost only)
Apache Doctor Dashboard
p: 80/443 (public)
PALANTIR KAFKA
PALANTIR Internal Network subnet: "/24"

Figure 6: virtualised scenario developed for Use Case 1.

The implementation of this scenario relies on Docker, creating a virtualised demo with portable and lightweight characteristics. The deployment method uses a docker-compose file, which creates the different Docker containers with all elements. The docker-compose.yaml in use is available in the public GitHub repository.

4.2.Security Orchestrator

This section proceeds to describe both the (i) current or expected implementation considerations; and the (ii) implementation details for the different modules within SO.

As per the considerations for the implementation of the design, SO follows different architectures or approaches, in order to provide a hybrid implementation that combines the benefits of both worlds. In this case, it relies on the microservices architecture but attempts to alleviate some of its issues, leveraging parts of the monolithic architectures. The final approach would fall near the miniservices architecture. On the one hand, the logic relies heavily on interactions between the small constituting parts (here, modules) so that a given workflow relies on multiple independent modules, each with a clear intent and somehow independent from others. On the other hand, transversal layers server common functionality to a number of modules, aggregating both common data and logic in use. In fact, miniservices may share data, instead of replicating databases for each of them. This also shared common logic. For instance, multiple modules may need to access the same data layer; or may need to authenticate against the same access control layer or clearinghouse.

With regard to the implementation details per module, there are some implementation details that are common to all the modules. Specifically, the language of choice for coding is Python3, the language of choice for documenting will be Markdown and the language in use for configuration will be JSON or YAML. Either the APIs will leverage Flask [21] or FastAPI [22] and its documentation will be both in markdown and in Swagger/OpenAPI [23]. The deployment will be carried out via Docker [24] (specifically through docker-compose [25]) and Kubernetes [26] for the production environments; as well as in venv for the development environment. The subsections below expose a non-exhaustive list of technologies used in the SO modules. Note that some of them will be in use for almost every module, like those related to the API, the DB access or the handling of the configuration elements.

4.2.1. Application Programming Interface (API)

This externally facing API currently uses the `Flask` library. Migration to `FastAPI` will be considered in the next iteration, as it provides a nice set of features and good integration with tools more tailored to production environments. It will support both JSON and YAML content types.

4.2.2. Attestation & Remediation (ATR)

This module uses Python clients that allow interacting with the runtime environment. Examples of these are the `docker` and `kubernetes` modules, or the ones that allow managing OpenStack [27]. Some examples of data provided through such clients are: virtual environment configuration, details on the running instances and virtualisation technology in use, image details for a given virtual node, etc.

4.2.3. Configuration (CFG)

This module gathers all other module's configuration from their related specific configuration files. Such files are stored in the same folder in the file system, at the root of the project. The CFG module reads the configuration elements directly from disk, ideally to observe their changes and dynamically re-reading these. Even though the default configuration format is YAML, the CFG module uses data transformation libraries, such as `pyyaml` and `json` where needed to expose, according to the Content-Type requested to its interfaces.

4.2.4. Life-Cycle Management (LCM)

Considered the main module of SO, the LCM module uses a myriad of libraries and technologies. It relies heavily on the `requests` library as well as specific clients in order to attack the APIs of third-party tools (like `docker`, `kubernetes` or specific OpenStack clients); as well as potentially libraries like `paramiko`, allowing for occasional remote code execution on the SCs. Any expected communication with the Kafka [28] message broker is subject to the needs of other components regarding the access to data. For instance, if some other component or subcomponent requires reading from the message broker some information collected by the SO. The `kafka-client` library will provide support for this.

4.2.5. Monitoring (MON)

This module internally triggers the instantiation of, at least, a local Prometheus [29] server instance. Once the operator registers the monitoring requests, the SO shall deploy the required `Prometheus` Node Exporter instances in the monitored SCs. The monitoring requests include the definition of a Unix-like command to execute remotely and enacted through the usage of libraries like `paramiko`. Such remote code execution is subject to previous filtering, undergoing validation stages against a command whitelist and against input validation.

4.2.6. Package (PKG)

The interaction with the NFVO will use the `requests` library to access the OSM [30] endpoints related to the package management. Other libraries, such as `pycurl`, may be required in order to fetch the SC packages in order to store, process or delete these during the package-related operations and prior to their onboarding in the NFVO.

4.2.7. Policies (POL)

First, libraries related to data processing (like `pyyaml`) and transformation (*e.g.,* those operating with XML structures) are in use to process and convert monitoring-related data, as well as inputs from the operators. The monitoring of the surpassed threshold is initially considered to be either implemented via Prometheus's alerts (for built-in or Prometheus-stored metrics) or through a custom daemon process using built-in libraries such as `multiprocessing` and `subprocess`.

Finally, the enactment of the expected policy actions could likely be offered as remote command execution to any of the managed SCs (via `paramiko`) or as a notification endpoint (using `requests`). A potential scenario for the application of the latter is that of an infrastructure operator that defines a policy to shut down a specific process or even the whole SC when it requires too many resources of a given type that are scarce in the infrastructure (*e.g.,* as the number of cores). The underlying assumption in such a case could be that a given process in the SC could be malfunctioning, and it may need to be re-instantiated.

4.2.8. Authentication & Authorisation Control (AAC)

As in the aforementioned module, the technology specs are to be determined in this case.

The authentication process will likely follow a Public-Key Infrastructure (PKI) approach, leveraging trusted X.509 certificates to perform a two-way authentication, at least between the internal modules within SO and ideally with the external requesting client. It is to be determined whether the Certificate Authority (CA) will be internal to SO or whether the certificates are generated and revoked manually and across the project.

4.2.9. Database Layer (DBL)

MongoDB [31] is the database engine of choice. This provides the SO with its own independent non-relational database that supports unordered collections of data in a JSON format. This fits especially well with data models in Python. Since data from all the other modules is persisted, each module writes to its own database within the same MongoDB instance, and uses collections to organise internally per type of data model.

4.3. Security Capabilities Catalogue

This section describes the current or expected implementation details for the SCC and its different modules. At a glance, the SCC is essentially a service that uses a database and an object store. The SCC service is composed of some logical modules whose design was explained in section 2.3.

In general, the way the SCC operates is as a "technology stack", as depicted in Figure 7. Since the SCC works in tandem with some other PALANTIR components, the figure below also includes the general interactions with other components, according to the technology in use.

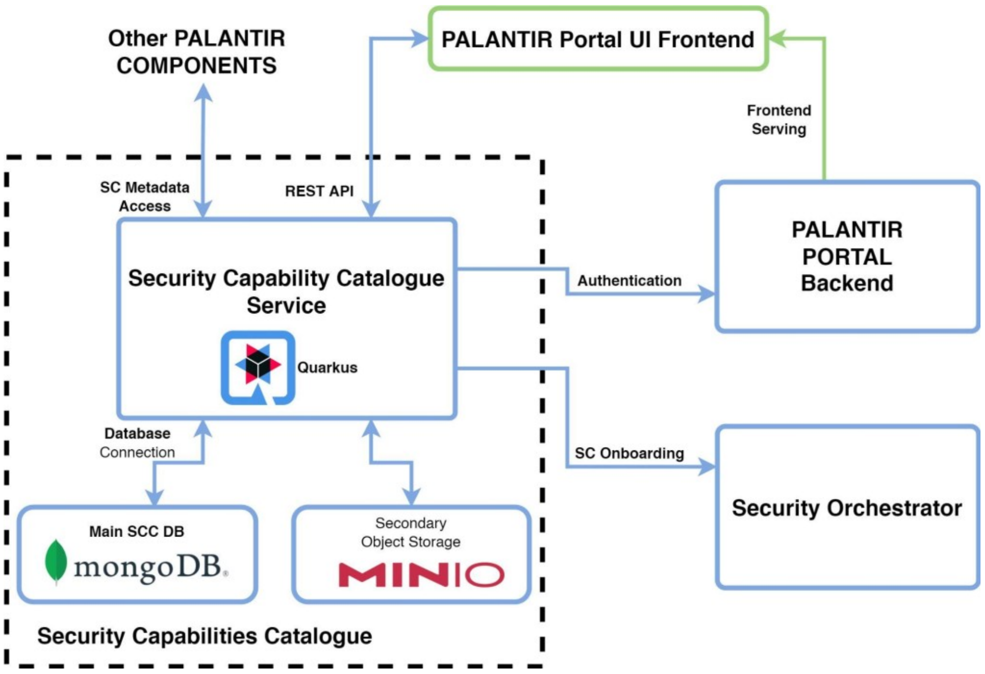

Figure 7: SCC technologies layout.

The SCC modules are parts of the SCC service, which is based on Quarkus [32]. These modules essentially represent Java Services, each implementing a specific functionality and wrapping any dependency, and together they compose the SCC Service.

The Security Capability Registration, Security Capability Search and Security Capability Metadata Access expose their functionality as a REST API, as previously mentioned. The Security Capability Data Validation, the SC Package Maker and SC Onboarding on SO are background tasks executed as threads within the SCC Service. However, at the core of everything in the SCC is the data model in use by the SCC's MongoDB [31], which is equivalent to the data model in use by the REST API. The data model can be viewed in annex 9.

Accessing the SCC is realised only via REST API, as the SCC implements specific functionalities. The clients of this REST API are to be filtered based on the identified role. The role is identified based on the JSON Web Token (JWT) [33], which is used for authentication of the client that interacts with the SCC. The SCC will interact with the user management of the Portal in order to achieve the appropriate user filtering, once the integration with it is determined. Essentially, the user management will check the user credentials and fetch the roles from the Portal's Identity Manager, and the SCC will use the Quarkus integrated security mechanisms [34] and SmallRye JWT [35] for authentication.

Access to the MongoDB database is done via a persistence layer added as a Quarkus plugin, which integrates with the MongoDB client and allows for management of records. In the case of the SCC, the `quarkus-mongodb-panache` plugin is used, and a Panache Mongo repository is made as an intermediate layer between the database and the core logic modules.

The API allows for the registration of a SC, viewing a list of SCs, searching for SCs based on some parameters, update of an existent SC, and deletion. The API layer is only concerned with REST resources and the basic data validation on the controller of each resource. Although it appears as though it is a simple set of CRUD operations, various intricacies have to be tackled separately. As such, there are modules formed as internal services in the context of a service facade, each dedicated to a specific set of operations. The subections below provide the implementation details for the SCC modules.

4.3.1. Security Capability Registration (SCR)

The Security Capability Registration service is considered during the coordination of incoming requests, to register or update a SC. After performing some checks during the request to ensure that there is no erroneous input, it marks the data insertion operation complete. Access to this service is delegated only to an end-user that is authenticated as a SC developer. The validation enacted at the request, however, and the overall pipeline of the registration and onboarding is not completed when the response is sent. The main procedures of the Security Capability Descriptors Validation, Package Maker and Onboarding services have to continue in the background as a long-running task. According to the operation, either the creation or update of a SC, as well as the inputs, this service creates the appropriate JobRunr [36] background job, containing operations from the other services, until the final onboarding to the SO takes place.

4.3.2. Security Capability and Descriptors Validation (SCDV)

The Security Capability & Descriptors Validation is concerned with fetching the software image, validating it and matching it with the descriptors, to the possible extent. If these succeed, the SCDV prepares all the data to be packaged by the subsequent module/service; *i.e.,* the Security Capability Package Maker. The software image file is downloaded from the indicated image registry, if the developer gives the platform access to the repository, or if it is a public repository. Alternatively, the software image file has been uploaded directly by the developer. The hash checksum of the file is created and matched with the one passed in by the developer. If the hashes do not match, the SC entry is deleted. Otherwise, support metadata is fetched from the database. The SCDV will conduct any further check on the image and passed parameters, which may be defined during the integration of the project and follow the hash matching. Finally, the metadata files are created, in order to have the complete package contents, and are formed based on the SCC entry. The hash of these files is also generated. If all steps succeed, the SCDV passes all the results to the SCPM, in order to continue.

4.3.3. Security Capability Package Maker (SCPM)

The Package Maker essentially gathers all required data, in the form of files, orders them in a hierarchy that is compatible with the SO, and then compresses them into a single tar.gz file. The SCPM module/service uses the Apache Commons Compress [37] library in order to implement the compression step. Finally, the compressed file is passed into the SC Onboarding, and is temporarily stored in the temporary packages bucket in MinIO [38].

4.3.4. Security Capability Onboarding (SCC-O)

The SC Onboarding uses a REST client that connects with the SO. Using the API of the SO, the package is sent, and the SO does the onboarding process on its side, while the SCC awaits for success or failure. The SCC's onboarding services retries (if possible) in cases of failure or error. Once the SO has completed its own onboarding process for the SC, the main procedure of the service is finished, as well as the background task that runs it.

4.3.5. Security Capability Search (SCS)

The SCS implements the search capabilities of the SCC. Here, the MongoDB-Panache extension of Quarkus [39] is heavily used, in order to form simple or complex queries on SCs. The SCS provides the option to query based on specific values for the queryable data fields of SC entries, and dynamically builds and executes queries on MongoDB. The role of the client/user that calls the endpoint that uses this service is used as a filter, because different users may have different access to SC data in some cases.

4.3.6. Security Capability Metadata Access (SCMA)

This simple service is concerned with fetching the appropriate data about a SC. Access to this service is delegated to an authenticated end-user, or to a client authenticated as a PALANTIR component. More importantly, the access to the metadata is filtered, based upon the role. The service fetches the appropriate document from the database as a response, if the criteria match.

4.4. Risk Analysis Framework

The integration of RAF, at least at its first release, is a semi-autonomous tool which runs in a standalone fashion and which uses simple graphical tools in order to collect required information from the SME.

Besides the parts based on questionnaires with closed answers, and which can be automatically evaluated; the information collected also requires human intervention. This intervention follows the form of discussion between the PALANTIR risk assessment expert and the SME delegate, and its aim is to better understand the finer details of security requirements and risk profiling for each SME. Hence, a fully automatic RAF is not anticipated.

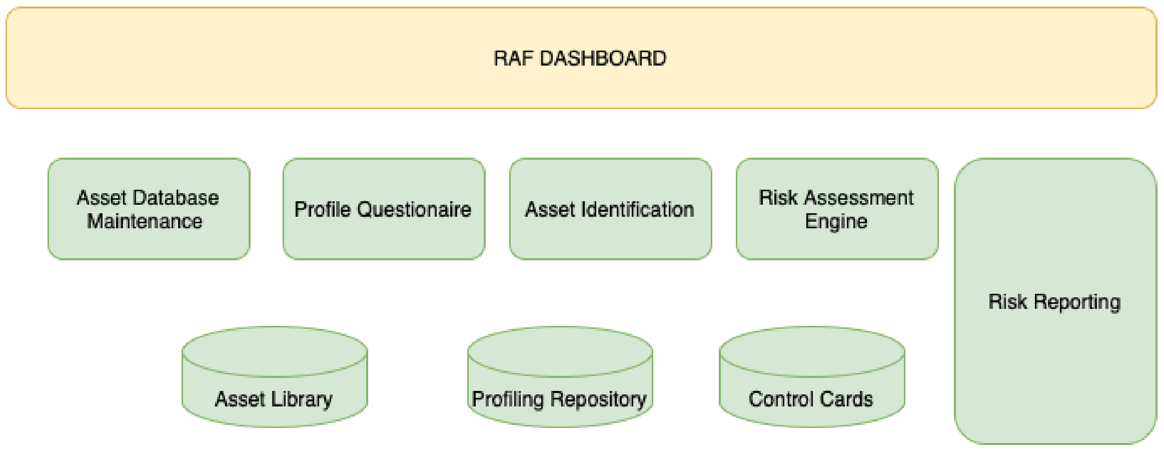

Figure 8: RAF layout.

The current implementation leverages the LimeSurvey platform [40]. The implementation supports the following phases:

- Phase 1: Questionnaire and automatic calculation of Risk Profile
- Phase 2: Asset identification using the ontology suggested by ENISA and a limited set of asset entries along with their known exploits and attack surface.

A risk calculation engine, under development, will take into account Phase 1 and 2 SME data and will automatically assess the risk while providing input for the consequent Phase 3. An output produced after the first year of the project relates to lessons learnt and best practices in the form of control cards.

Figure 8 presents the RAF layout. The information collected or used is organised in three databases: i) the Asset Library, which provides the set of well-known assets; ii) the Profiling Repository that collects all imported data from SMEs, as well as all relevant evaluations and outputs of RAF; and (iii) the Control Cards database, providing all available control cards within RAF.

The Asset Database Maintenance subcomponents are responsible for the maintenance of the Asset Library, updating information related to asset category entries, among others.

The Profile Questionnaire is a subcomponent responsible for the SME profile risk questions, used to lead to profile risk evaluation. Similarly, the Asset Identification is the subcomponent exposing the form to complete in order to collect SME assets and security requirements. The Risk Assessment Engine is the subcomponent that calculates the risk, taking into account information collected by Phase 1 and Phase 2 steps. Finally, the Risk Reporting subcomponent is responsible for creating the final reports that will include the risk evaluation and all relevant information related to organisation and asset management. In addition, this component will communicate the results of risk evaluation to other PALANTIR components for implementation and monitoring purposes. The SME users will be able to interface with the RAF through the dashboard, which is implemented by the Portal frontend.

5. Conclusions

This deliverable provided the first detailed view on the architecture, low-level design and specifications of the components and subcomponents belonging to WP3.

The information here provided is the current, up-to-date architecture and design for each component or subcomponent. Such design covers from the internal separation of logic concerns to the interactions between the internal modules. All of these pave the way for the development efforts within each of them, as well as for discussions of future integrations with other components.

Besides the design, this deliverable gathers the low-level, development-related details in the form of technical specifications (refined from the general requirements introduced in D2.1), the technical decisions, benefits and justifications for each subcomponent, as well as pointers to implementation.

6. References

[1] Valenza, F., Su, T., Spinoso, S., Lioy, A., Sisto, R., & Vallini, M. (2017). A formal approach for network security policy validation. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl., 8*, 79-100.

[2] ENISA, "Information Package for SMEs With examples of Risk Assessment / Risk Management for two SMEs" https://www.enisa.europa.eu/publications/archive/RMForSMEs (Feb. 2007).

[3] Elastic, "Filebeat: Lightweight shipper for logs" https://www.elastic.co/beats/filebeat

[4] Snort, "Snort" https://www.snort.org/downloads

[5] Open Information Security Foundation, "Suricata" https://suricata.io/download

[6] Wazuh, "Wazuh" https://wazuh.com/start

[7] nDPI, "nDPI official GitHub repository" https://github.com/ntop/nDPI

[8] ntop, "ntop's user's guides" https://www.ntop.org/support/documentation/documentation

[9] ntop, "ntop's nProbe" https://www.ntop.org/guides/nprobe/introduction.html

[10] ntop, "ntop's PF_RING" https://www.ntop.org/guides/pf_ring/get_started

[11] ntop, "ntop's 2disk" https://www.ntop.org/guides/n2disk/installation.html

[12] ntop, "ntop's nBox" https://www.ntop.org/guides/nbox/installation.html

[13] ntop. "ntop's nScrub" https://www.ntop.org/guides/nscrub/install.html

[14] Snort, "Elasticsearch's Snort module for receiving Snort/Sourcefire logs over Syslog or a file" https://www.elastic.co/guide/en/beats/filebeat/7.13/filebeat-module-snort.html

[15] PALANTIR, "Public repository for the SC subcomponent" https://github.com/palantir-h2020/sc-secaas

[16] Electric Sheep Fencing, "pfSense: free network firewall distribution" https://www.pfsense.rog/getting-started/

[17] Opnsense, "OPNsense: open source, easy-to-use and easy-to-build HardenedBSD based firewall and routing platform" https://opnsense.org/about/about-opnsense/

[18] iXsystems, "Truenas: world's most popular software-defined storage" https://www.truenas.com/

[19] ntop, "ntop: Web-based Traffic Analysis Tool" https://www.ntop.org/products/traffic-analysis/ntop

[20] OpenEMR, "Free and Open Source electronic health records and medical practice management application" https://www.open-emr.org

[21] Pallets, "Flask: web development, one drop at a time" https://flask.palletsprojects.com

[22] S. Ramírez, "FastAPI framework, high performance, easy to learn, fast to code, ready for production" https://fastapi.tiangolo.com

[23] SmartBear Software, "OpenAPI: API description format for REST APIs" https://swagger.io

[24] Docker, Inc., "Docker: Empowering App Development for Developers" https://www.docker.com

[25] Docker, Inc., "Docker Compose: tool for defining and running multi-container Docker applications" https://docs.docker.com/compose

[26] Cloud Native Computing Foundation, "Kubernetes: open-source system for automating deployment, scaling, and management of containerised applications", https://kubernetes.io

[27] Open Infrastructure Foundation, "OpenStack: Open Source Cloud Computing Infrastructure", https://www.openstack.org

[28] Apache Software Foundation, "Kafka: open-source distributed event streaming platform",

https://kafka.apache.org

[29] Cloud Native Computing Foundation, "Prometheus: open-source systems monitoring and alerting toolkit" https://prometheus.io/

[30] ETSI, "OSM: open source Management and Orchestration (MANO) stack aligned with ETSI NFV Information Models" https://osm.etsi.org/

[31] MongoDB Inc., "MongoDB: the application data platform" https://www.mongodb.com

[32] Red Hat, "Quarkus: Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM" https://quarkus.io/

[33] Auth0, "JWT: open, industry standard RFC 7519 method for representing claims securely between two parties" https://jwt.io/

[34] Red Hat, "Quarkus' security architecture and guides" https://quarkus.io/guides/security

[35] Red Hat, "SmallRye JWT: implementation of Eclipse MicroProfile JWT RBAC" https://smallrye.io/docs/smallrye-jwt

[36] R. Dehuysser, "JobRunr: an easy way to perform background processing in Java" https://www.jobrunr.io/

[37] Apache, "Apache Commons Compress: API for working with compressed files" https://commons.apache.org/proper/commons-compress

[38] MinIO, Inc, "MinIO: world's fastest object storage server" https://min.io

[39] Red Hat, "MongoDB Panache: simplified MongoDB usage" https://quarkus.io/guides/mongodb-panache

[40] LimeSurvey, "LimeSurvey: world's #1 open source survey tool" https://community.limesurvey.org/

[41] PALANTIR, "Public repository for the SCs" https://github.com/palantir-h2020/sc-secaas

[42] PALANTIR, "Public repository for the SO" https://github.com/palantir-h2020/sco-so

[43] ETSI, "ETSI GS NFV-SOL 006 V2.6.1, Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on YANG Specification" https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/006/02.06.01_60/gs_NFV-SOL006v020601p.pdf

[44] ETSI, "Open Source MANO's documentation ANNEX 3: OSM Information Model" https://osm.etsi.org/docs/user-guide/11-osm-im.html

[45] ETSI, "Open Source MANO's documentation: Information Model Trees" https://osm-download.etsi.org/repository/osm/debian/ReleaseTEN/docs/osm-im/osm_im_trees/etsi-nfv-vnfd.html

[46] ETSI, "ETSI's NFV SOL006 Repository: sample data in XML" https://forge.etsi.org/rep/nfv/SOL006/blob/master/example-data/nfv.xml

[47] The Linux Foundation, "SigDev: new standard for signing, verifying and protecting software" https://www.sigstore.dev/

[48] PALANTIR, "Public repository for the SCC" https://github.com/palantir-h2020/sco-scc

7. Annex A: SCs design, implementation, usage

7.1. Intra-component interfaces

As mentioned in the design section, particularly in section 2.1, two internal interfaces have been considered in the SC component, as pictured in Figure 1.

Before proceeding to the explanation of the internal interfaces, it is important to understand the purpose of the external interface (VeNf-Vnfm). This interface provides a way to connect the SO with the SCs. It is worth noting that, disregarding the indirect connectivity between the SO and the SCs (such as the SO being requested to terminate or isolate a given SC, or even the SO and SCs writing and/or reading from similar Kafka topics), this interface only acts on already established and running SCs: that is, once the SC has been properly instantiated and is reachable to the SO. This said, there can be at least two types of communication through this interface:

(1) Any kind of life-cycle management action that is fulfilled through the NFVO. It does not matter here whether the requested operation may be considered built-in (start, stop, restart, reboot, upgrade) or custom (*e.g.,* "`sc-config`", "`sc-healthcheck`"), or whether these are day0, day1 or day2. This is because, in the end, all these operations, whether explicitly described as primitives in the VNF descriptor or not, must have their implementation available in the Juju charm, as Juju actions. Such actions get executed into the VNF through a specific mean, defined in the Juju charm's layer configuration. In particular, the SO would trigger a configuration action with the specific MSPL statement (a more generic request to apply inside the SC). This would reach the SEM, where MSPL would be transformed to LSPL (a low-level request that can run directly) and right afterwards this would enforce the LSPL to the expected VNF through the VNFi-Mgmt interface.

(2) Operations that may directly execute from the SO to the SC, for instance if the logic of such action was not implemented as a Juju action or is not a built-in operation. An example of this could be direct monitoring of a given VNF through SSH or an interface exposed within the SC over HTTP(S).

The aforementioned explanations help describing the internal interfaces.

The first internal interface (VNFi-Mgmt) communicates the received Juju actions from the SEM to the internal VNFs, mainly to realise the actions requested by SO.

The second internal interface (VNFi-Data) provides the main interaction between the SEM and the VNFs in order to collect the information generated from the VNFs.

The PALANTIR platform understands, in many cases, the SCs as sensors allocated in the SME client. Due to this, the data produced by the SCs need to be exposed to the rest of the PALANTIR platform. In the future, D3.2 will present, as needed, the different interactions identified for this interface.

7.2. User guide

In this case, the PALANTIR repository for the Security Capabilities [41] provides the information related with the deployment and use of the different Security Capabilities. The README.md contains the explanation of the repository structure. Besides, each folder contains specific information, commands, and configuration files to use. Initially, each SC offers Docker and Kubernetes deployment scripts, as well as any other possible specific commands and configurations needed for the correct functioning.

8. Annex B: SO design, implementation, usage

8.1. Intra-component APIs

As stated in the implementation section, the SO follows a hybrid architecture that resembles the miniservices architecture.

This approach considers a number of miniservices interacting with each other, as well as common or transversal layers that can group data or logic that is in use by the rest of modules. Both of them may expose interfaces to others, in this case, RESTful APIs.

Here, the API and CFG modules, as well as the DBL transversal layer will be ingesting or aggregating data from others, being a consumer entity that does not expose interfaces.

Table 6 provides the current draft, indicating the current design status. This schematic and simple approach ignores lower-level details like GET parameters in order to filter provided resources. It is important to indicate that this is work in progress and thus subject to changes by D3.2.

Table 6: description of APIs internal to SO.

Endpoint	Resource	Methods	Request body
Module: AAC			
/authn	--	POST	{“mod-name”: “...”, “cert”: “...”, “cert-req”: “...”}
(1) If a module does not have a certificate of its own, it will request the generation of a certificate (possibly signed by an internal self-issued CA) to valid modules requesting access. (2) When the certificate is available, and a given module is requested for an operation, the module will authenticate itself first and then the external requesting client.			
/authz	--	POST	{“mod-name”: “...”, “cert”: “...”, “cert-req”: “...”}
Checks the provided certificate for an external requesting client and identifies whether it can request a given resource.			
Module: ATR			
/runtime /runtime/	-- node-id	GET GET	-- --
Requests all the available information for the totality or a specific node, whether an infrastructure physical node or a virtualised instance of a SC.			
/attestation/	node-id	POST	{“node-id”: “...”, “status”: “...”, “reason”: “...”}
Communicates a new attestation incident report to the SO, by uniquely identifying the ID of the failed node that was first attested by the AE, as well as the reported attestation status and the reason.			

Module: LCM			
/ns /ns/	-- ns-id	GET GET	-- --
/vnf /vnf/	-- vnf-id	GET GET	-- --
Requests all available information for the totality or a specific NFV-related resource (*i.e.,* the SCs) that is currently either available for instantiation or already instantiated. This is the case for the internal abstractions (NS and VNF) that are used by each SC.			
Module: MON			
/vim /vim/	-- vim-uuid	GET POST	-- --
/vnf /vnf/	-- vnf-uuid	GET GET	-- --
Requests all available telemetry data for the totality or a specific resource (such as the VNFs that are running or the underlying VIMs). Metrics can come from a pre-defined pool or defined by the operator.			
/vnf/metric /vnf/metric/ /vnf/metric	-- metric-id --	GET GET POST	-- -- {"name": "...", "vnf-uuid": "...", "cmd": "...}
GET Requests information on the operator-defined, custom metric requests to be enacted at a specific VNF. Either the totality or a specific metric request can be consulted. **POST** Submits the operator's monitoring request on a specific VNF, which is already running, to extract a specific metric, defined by a Unix-like command defined by the operator itself. Once submitted, the requested telemetry data will be polled in a periodic fashion and stored locally.			
Module: PKG			
/ns-pkg /ns-pkg/ /ns-pkg	-- pkg-id --	GET GET POST	-- -- {"disk-path": "..."}
/sc-pkg /sc-pkg/ /sc-pkg	-- pkg-name ---	GET GET POST	-- -- {"disk-path": "..."}
/vnf-pkg /vnf-pkg/ /vnf-pkg	-- pkg-id --	GET GET POST	-- -- {"disk-path": "..."}
GET Requests all available information for the totality or a specific SC package (*i.e.,* the VNF and NS			

package contents that lie within the SC), which is currently registered in the SCC and in the internal NFVO registry. **POST** Registers a new SC package into both the SCC and the internal registry of the NFVO, effectively onboarding the contained packages.			
Module: POL			
/cfg /cfg/ /cfg	-- cfg-id --	GET GET POST	-- -- {“vnf-uuid”: “...”, “cfg”: “...}
GET Requests all registered configuration policies for the totality or a specific VNF running instance. **POST** Registers a new configuration policy (in the form of an MSPL statement) for the running instance of an NFV. The LCM module is the one transmitting this.			
/mon /mon/ /mon	-- mon-id --	GET GET POST	-- -- {“name”: “...”, “metric-id”: “...”, “threshold”: “...”, “vnf-uuid”: “...”, “cmd”: “...”, “web-hook”: “...”}
GET Requests all registered monitoring policies for the totality of the VNF running instances or for one of them. **POST** Registers a new monitoring policy for a given VNF running instance, indicating the threshold to monitor for a particular metric monitored within the VNF (*i.e.,* which was previously registered as a monitoring request on the same VNF). It indicates the threshold value such metric would surpass before triggering a specific command and/or sending a notification to a specific webhook URL.			

8.2. Technology selection

The SO subcomponent follows principles that maximise or minimise specific conditions.

On the one hand, it targets the maximisation of (i) modularity between internal modules, (ii) flexibility on its deployment, (iii) availability of libraries, bindings and tools.

On the other hand, it aims to minimise the (i) complexity of the code, (ii) decoupling of similar logic indifferent subcomponents, and (iii) maintenance costs.

To this end, a simple programming language was chosen (Python), such that the readability is increased and the development and maintainability costs are reduced. In its simplicity, it heavily leverages on the JSON format, which is a commonly used format that can match well with unstructured collections of data (thus being able to use non-relational and, in a way, more flexible database engines such as MongoDB) and that can straightforward translate to the YAML format.

This language offers a wide range of both built-in libraries (like `multiprocessing`, `subprocess` or `json`), as well as other custom made (like `requests`, `pyyaml`, `paramiko` or `uvicorn`). It also provides a huge selection of bindings for commonly used frameworks as tools. Some examples are in virtualisation of computing instances (like OpenStack, Docker or Kubernetes), inter-component communication (Kafka) or NFV orchestration (OSM).

The availability of so many tools will be leveraged in a way that it can provide further flexibility to operators with alternative options to deploy it through different methods.

8.3. User guide

The repository for the SO [42] includes the documentation for deployment and the access to the exposed APIs. It also includes the structure with the modules described in the previous sections.

The README.md file at the root level indicates the location of each module, whereas inside each module, the README.md indicates the URL and port where the specific service is exposed, and hwo to access the content through the specific REST API endpoints, implemented per module.

The repository also contains deployment scripts that use the Docker and docker-compose tools by default; yet also a draft version for the Kubernetes deployment.

9. Annex C: SCC design, implementation, usage

9.1. Data model

PALANTIR has a single, centralised and searchable catalogue that stores SCs, which are essentially security VNFs with accompanying security descriptors. The SCC allows for registration and search of security capabilities, so that developers can create and offer new security features to enterprises, in order for them to protect their networks or extend existing security functionality without making changes to the core network topology or the network elements in operation. The SCC acts as a repository of SCs on offer.

In order to operate, the SCC requires a data model to describe each SC. This facilitates the registration, validation and onboarding of a SC by a developer, as well as the search and selection of a SC by a network operator or an enterprise manager. The data model incorporates aspects relative to (i) the NS and VNF description; (ii) metadata considered necessary for the operation of PALANTIR's innovative features like threat detection and remediation; (iii) which parameters are required for searchability; (iv) billing data; (v) privacy metadata; as well as (vi) integrity metadata, which have to do with both the SC itself and the developer.

Following are the tentative collections of metadata to include in the SCC model.

9.1.1. VNF Descriptors

In PALANTIR, OSM is the currently chosen NFVO in use for the SO. The SCs stored in the SCC must therefore comply with the packaging guidelines supported by OSMr10. In previous versions, OSM adopted the ETSI SOL006 standard [43][44]. Consequently, the SC package should comply with these standards. It is worth noting that the supported OSM release may be more up-to-date throughout the project, provided it does not introduce major changes and there is agreement between the involved partners.

The SCC is not to include all fields mentioned in the OSM models for NS and VNF in the SC entry. However, there are fields that shall be included in the SC entry, most of which should be searchable. The rest of the descriptors that are static, are to be included in the package. Table 7 explains these fields.

Table 7: description of data model for the VNF descriptor.

Name	Technical name	Purpose, description	Contributors	Ext. availability	Search ability	Example value
id	id	**ID** to reference the VNF of the SC, and therefore the same as SC ID.	Dev	Y	Y, select single	ASA
provider	provider	**Designer** of the appliance	Dev	Y	Y	My company
version	version	**Version** of the **VNF** appliance. VNF	Dev	Y	Y	1.1

		versioning scheme.				
software version	software-version	**Version of the contained software**, following its own versioning.	Dev	Y	N	9.8
product name	product-info-name	**Name** of the SC	Dev	Y	Y	My company ASAv
product descriptio n	product-info-description	**Description** of the SC	Dev	Y	Y	Single FW and router
Security group rules	security-group-rule	**Security group rules** (list, as in VNFD) for protocols and ports of VNF	Dev	Y	N	(As in [45] vnfd/securit y-group-rule)
Software images	sw-image-desc	**Software images** needed for VNF	Dev	Y	Y, some	(As in [45] vnfd/sw-image-desc)

The table above captures several fields specified in the NS descriptor in [44]. The example fields are inspired by [46]. The selection of parts to be included is still to be determined and will be continuously refined through the integration process.

9.1.2. Billing and SLA descriptors

Table 8 describes a list of billing and SLA descriptors. The following aspects are taken into account:

- Billing of SCs usage.
- Assessment of the SLA of the SC.

Table 8: description of data model for the billing and SLA descriptors.

Name	Technical name	Purpose, description	Contributors	Ext. availability	Searchability	Example
Billing model	billing_model	Reference the supported	Ops	Y	Y	hourly

		billing models. (list)				
Subscription billing	subscription_billing	Price to access a SC at it subscription	Ops	Y	Y, range	20.0
Instance billing	instance_billing	Fees of a SC at its instantiation	Ops	Y	Y, existence, range	1.0
Hourly cost billing	hourly_billing	Fees	Ops	Y	Y, range	0.1
Tolerated SLA	sla_downtime	Amount of tolerated downtime per year	Ops	Y	Y, range	8h
SLA violation fees	sla_violation_fee	Discount on the non-tolerated downtime per hour	Ops	Y	Y, range	0.1

Note: **The above data fields are heavily dependent on other tasks**, and are work in progress at the time of writing this document.

9.1.3. Security-enhancing descriptors

A list of security-enhancing fields is defined below. The following aspects are taken into account:

- Qualification of the SC to respond to a specific threat and based RAF analysis/outcome/alert.
- The generic behavior of the SC.
- The types of threat the SC helps protect against.
- The types of assets the SC helps protect.
- Support for multiple deployment models for a SC.

Note that most of the fields in Table 9 are usually populated as a list of elements, as there are many times in which more than one value is applicable.

Table 9: description of data model for the security-enhancing descriptor.

Name	Technical name	Purpose, description	Contributors	Ext. availability	Searchability	Example
Supported deployment models	deployment_model	Supported PALANTIR deployment models	Dev	Y	Y	Cloud/MEC /vCPE

Detection method	detection_method	Detection method of the protection method implemented by the SC	Dev	Y	Y	network_flow_monitoring
Mitigation method	mitigation_method	Mitigation method of the protection method implemented by the SC	Dev	Y	Y	network_flow_filtering
Security control data type	control_data_type	Type of the data generated by the control feature	Dev	Y	Y	netflow_configuration
Security monitor data type	monitor_data_type	Type of the data consumed by the monitoring feature	Dev	Y	Y	pcap
Security threat protection	threat_protection	Unique name of threat that can be averted by the SC **(Optional)**	Dev	Y	Y	Kind of attack like "ddos" or specific malware or threat name like "wannacry"

Note: **The above data fields are heavily dependent on other tasks**, and are work in progress at the time of writing this document.

9.1.4. Integrity descriptors

The integrity descriptors are dependent on the SC package integrity guidelines agreed upon by partners. The goal is to assert integrity of the package, and to certify that the SC is created by a developer that is certified in the PALANTIR ecosystem, in order to avoid tampering and to avoid malicious registration of SCs.

It is worth noting that for the SCs there is no need for encryption of the contents of the SC package. However, the legitimacy of the submitted contents must be assertable. As such, the developer that adds the SC should be visible and identifiable; and the developer, the software image and the metadata are accompanied by a hash or a (most probably X.509) certificate.

The descriptors from Table 10 are to be used in the validation within the SCC, as well as for other components, like the AE (T4.4) under TAR. The technology to be used for code and assets integrity and the accompanying PKI can also be based on an open-source project, like the SigStore project [47].

Table 10: description of data model for the integrity descriptor.

Name	**Technical name**	**Purpose, description**	**Contributors**	**Ext. availability**	**Searchability**	**Example**

SC image hash (certificate ?)	sc_img_hash	Hash for the SC VNF Software image	Dev	N, only integrity check	N	<hash>
SC support metadata hash (certificate ?)	sc_meta_hash	Hash for the supporting SC metadata kept in manifests and DB	Dev	N, only integrity check	N	<hash>
Developer ID	dev_id	A unique ID for the developer, corresponds to KeyCloak ID of user	Dev (indirectly)	Y?	Y, select one	<unique id>
Developer name	dev_fullname	Developer's name	Dev (indirectly)	Y		Stelios Tsarsitalidis
Developer email	dev_email	Developer's e-mail	Dev (indirectly?)	Y		stsarsitalidis@ubitech.eu
Developer organisation	dev_org	Developer's organisation	Dev (indirectly?)	Y		Ubitech Ltd.
Developer certificate	dev_cert	Certificate for the developer	Dev (indirectly)	N, only integrity check	N	<X.509 certificate>

9.1.5. Privacy descriptors

The SCs are supposed to comply with the GDPR, and so the catalogue should express the kinds of personal data a SC may manipulate. The developer defines the necessary information to comply with the GDPR framework, and the kind of private data the SC will have access to or manipulate.

Table 11 is a work in progress at the time of writing, and will be updated based on feedback from partners with GDPR experience and T4.3.

Table 11: description of data model for the privacy descriptor.

Name	**Technical name**	**Purpose, description**	**Contributors**	**Ext. availability**	**Search ability**	**Example**

Interfaces and formats	interfaces_descr	Explains the nature of (i) input (ii) output data from the appliance and (iii) their nature.	Dev	TBD	N	Ingress and egress traffic from customer appliances. Netflow traffic.
GDPR applicability	gdpr_applicability_descr	Identify (i) manipulated personal data, their (ii) category and (iii)	Dev	TBD	Y, some	IP addresses, structure, without identification
Data storage	storage_descr	Indicate which data are stored, if they are encrypted, pseudonymise and anonymise.	Dev	TBD	N	No storage
Data processing	processing_descr	Detail the nature of the conveyed processing, and their modalities, such as (i) the profiling (ii) the monetisation (iii) the responsible entity of the data processor (iv) the DPO in charge (v) the data controller (vi) a reference to the consent process (vii) and the lawfulness.	Dev	TBD	Y, some	Block traffic from blacklisted IP addresses.
Data sharing	sharing_descr	Indicate to which entity, external to PALANTIR platform, the data are communicated, such as (i) third parties (ii) law enforcement (iii) foreign entities (iv) a CERT/CSIRT.	Dev	TBD	Y, some	No third party involved
Data subject rights	subject_right_descr	Indicate how stakeholders can exert their right on retained data, such as (i) the right to access (ii) the right of rectification (iii) the right to be forgotten (iv) the applied restriction (v) the notification and (vi) data portability, if applicable.	Dev	TBD	Y, some	No identification data are retained.

Open Internet	open_internet_descr	Indicate if the SC can influence the respect on the Open Internet norm.	Dev	TBD	Y	No traffic classification
Non-discrimination	non_discrimination_descr	Explain how data are used for discriminating users, and the consequences of misuse.	Dev	TBD	Y, some	Not applicable
ePrivacy	eprivacy_descr	Detail the compliance with the ePrivacy framework (*e.g.*, protection of the communication and the user tracking through cookies to retain preferences)	Dev	TBD	Y, some	

9.2. User guide

The PALANTIR repository for the SCC subcomponent [48] includes the documentation for deploying and accessing the exposed APIs. The README.md file at the root level explains how to spin up the SCC in developer mode, and how to package or run the SCC, and even how to create compiled native executables. The repository also contains deployment scripts that use the Docker and docker-compose tools. Finally, after spinning up the SCC in developer mode, the SCC REST API can be inspected and tested through Swagger UI, which is accessible at `SCC_service/q/swagger-ui`.