SECURING AGAINST INTRUDERS AND OTHER THREATS
THROUGH A NFV-ENABLED ENVIRONMENT

[H2020 - Grant Agreement No. 700199]

Deliverable D4.1

Specifications, design, and architecture for the usable information driven engine

Editor B. Gastón (I2CAT)

Contributors C. Fernández, C. Dávila (I2CAT), A. Litke, D. Papadopoulos, N. Papadakis (INFILI), G. Gardikis, K. Tzoulas (SPH), T. Batista, R. Preto (UBI), A. Pastor, J. Núñez (TID), M. De Benedictis, A. Lioy (POLITO), O. Segou (ORION), D. Katsianis (INCITES), M. Terranova (AGID), L. Jacquin, H. Attak (HPE), E. Trouva (NCSRD), V. Carela (TALAIA)

Version 1.0

Date May 31th, 2017

Distribution PUBLIC (PU)

Executive Summary

Following the work done in D2.1, where the requirements of the SHIELD platform were elicited and the high-level design and architecture of the platform was exposed, a detailed study of the different components has been done in order to obtain the low-level architecture and design (subcomponent granularity), the specifications (transformation of the user requirements into technical requirements/specifications) and the implementation guide (technologies to use). This work has been divided into the two technical development work packages of SHIELD namely WP3 and WP4. Hence, this deliverable covers the two components developed within WP4, i.e. the data analysis and remediation engine (DARE) and the security dashboard.

From the point of view of the low-level architecture, we expose some changes in subcomponents from the technical architecture exposed in the DoA of the project. Firstly, we have simplified some subcomponents with the objective to avoid using technologies with specifications that SHIELD does not use. For example, the Enterprise Service Bus (ESB) has been replaced by a more simple streaming service, since it is only needed to communicate information from the vNSFs to the central DARE and hence, no bidirectional flow or multi-cast is needed. It is worth to mention that SHIELD already identified several phases for the information treatment and valorisation, the main task of the WP4. These phases are defined in detail from the bottom (data ingestion) to the top (data visualisation) of the data valorisation stack. Between them, we define the data analysis phase to detect anomalies in the network and classify them, and the cybersecurity topologies phase, which recommends specific remediations to the detected threats in the form of new network services (sets of vNSFs) or reconfiguration of existing ones.

In terms of design, we have identified the five user stories behind the three use-cases of SHIELD, namely: i) vNSFs deployment, ii) vNSF withdrawal, iii) anomaly detection, iv) recommendation deployment and v) monetisation definition. From these user stories, we have exposed the workflow between subcomponents and between the subcomponents of the WP4 and the components of WP3. These workflows identify the flow of information and tasks that define and compose every user story.

One of the main aspects exposed in this deliverable is the transformation of the user requirements into specifications or technical requirements. The requirements identified in D2.1 were classified between platform requirements, non-functional requirements and service requirements. Firstly, we have identified which of these requirements have implications in every phase and subcomponent (D2.1 already did this work but only at a component level). Secondly, we have translated the requirements from the business language used in D2.1 to the technical language needed for the developments.

Finally, we have done a detailed research on the top technologies in the fields of cybersecurity, big data and data visualisation to be able to choose the ones that better address the exposed specifications. Our conclusion is that Apache Spot [1] is the best choice for the DARE. Firstly, because it covers several aspects needed by SHIELD: i) it has a data ingestion framework based on big data architectures, ii) it provides a probabilistic algorithm for network anomaly detection and iii) it exposes well documented APIs between every subcomponent. Secondly, because it is an open source project supported by a large community, an aspect that goes perfectly in line with the objectives of SHIELD in terms of open-source contribution, reuse of technologies and

contribution to on-going projects. Thirdly, because it is a mature project which goes that will help to achieve the expected TRL of SHIELD. Fourthly, because it is supported by big companies like Intel, which assures that the efforts of SHIELD are tied to the market. Fifthly, because the solution proposed by Spot is realistic and uses the topmost technologies in the state-of-the-art. Finally, because it permits SHIELD to focus the efforts in the anomaly classification, behaviour prediction and rapid remediation that conform the main objective of the project. In the context of SHIELD, Spot will be use as the starting point for the DARE developments. We have already identified several extensions, which will be needed to the core Spot platform in order to fulfil SHIELD requirements, mostly related to functionalities such as: mitigation capabilities; near-real-time operation; classification of threats; optimised operation in an NFV environment; and enhancement of the data model to support for more types of information.

Table of Contents

1. INTRODUCTION

This document presents the low-level architecture, design and specifications of the components involved in the usable information-driven engine, within WP4. The document also summarises the work done in the first iteration of task T4.1. This deliverable starts from the high-level architecture, design and requirements presented in D2.1; and provides specific details of the components' design, definition and their adequateness regarding the SHIELD requirements.

SHIELD, as a use case driven project, aims to cover the functionality required by the following three use cases (already defined in D2.1 but briefly recalled here for the sake of completeness):

- *Use case 1*: An Internet Service Provider (ISP) using SHIELD to secure its own infrastructure. This UC involves the ISPs deploying vNSFs in their network to detect incidents (Figure 1).

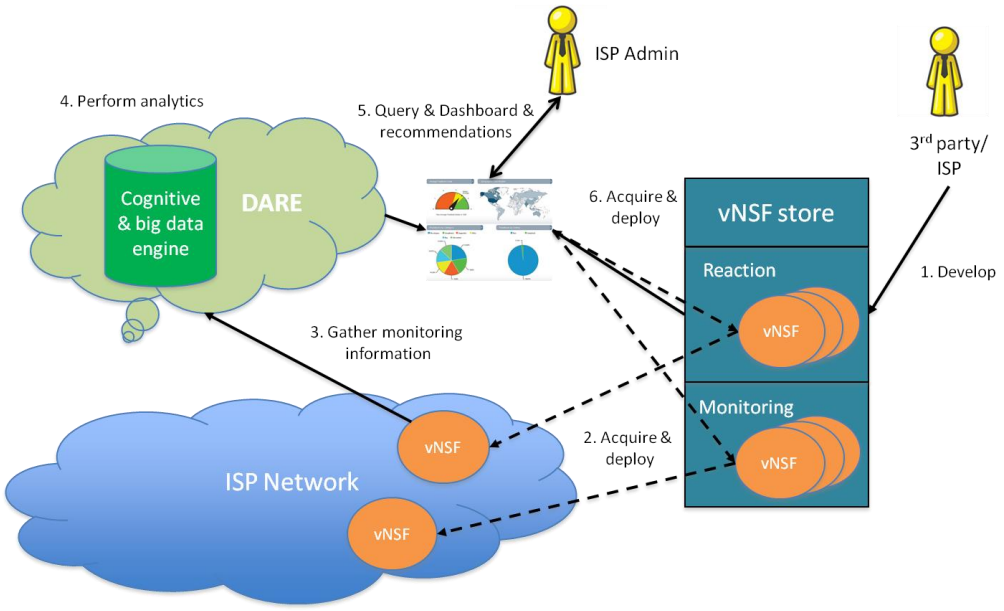

Figure 1: High-level picture of the use-case 1

- *Use case 2:* An ISP leveraging SHIELD to provide advanced SecaaS services to customers. This UC assumes that network security services (consisting of vNSFs), along with real-time incident detection and management, are offered as-a-Service to ISP clients, such as enterprises, public bodies, etc. (Figure 2).

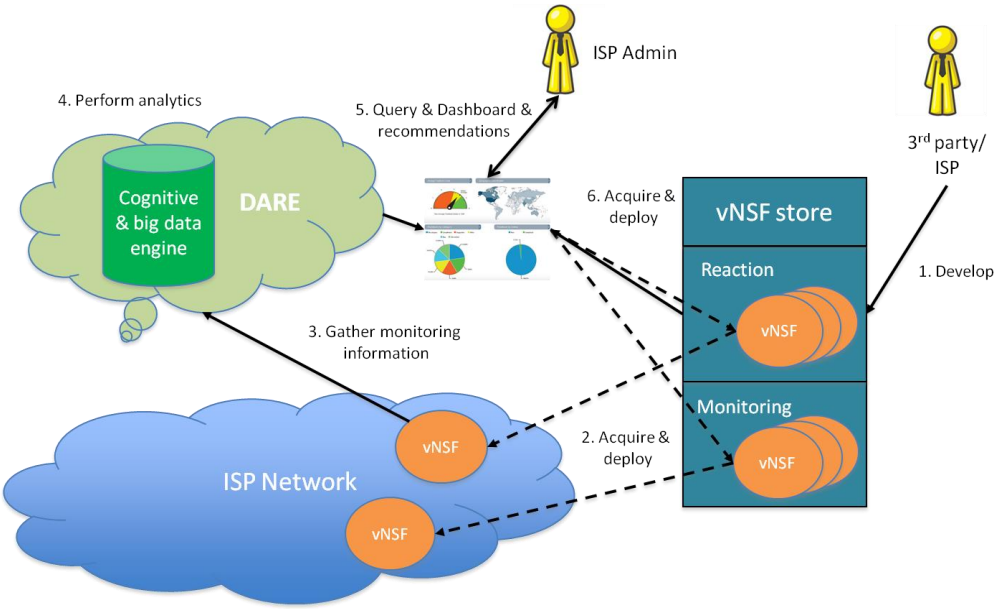

Figure 2: High-level picture of the use-case 2

- *Use case 3:* Contributing to national, European and global security. This UC assumes that incident information is exposed, in a secure and private manner, to public cybersecurity authorities (Figure 3).

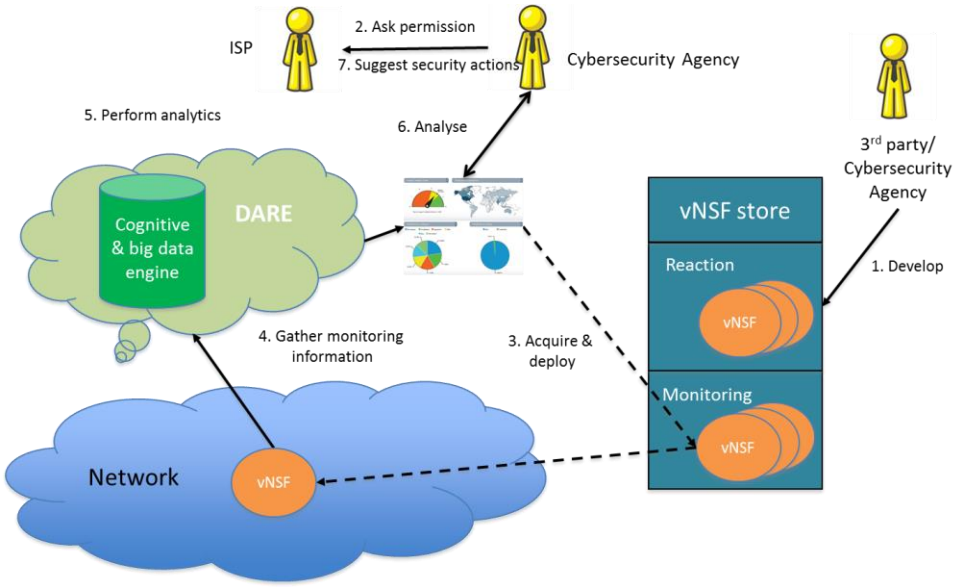

Figure 3: High-level picture of the use-case 3

Although the three use cases act as the basis of the analysis, the resulting architecture, design, specifications and implementation have been elaborated to produce a unified and universal solution i.e. a single cybersecurity solution that can be used for multiple purposes. To this intent, the SHIELD platform enables the actors in the different use cases with different views and roles on the network. For example, while an ISP (use case 1) can view the big picture of the cybersecurity analysis and can deploy a vNSFs in any location of the network, the ISP client (use case 2) only has access to a limited vision of the cybersecurity picture (information that is

offered by the ISP and/or paid by the client) and can deploy vNSFs in specific places of the network (i.e. to its gateways).

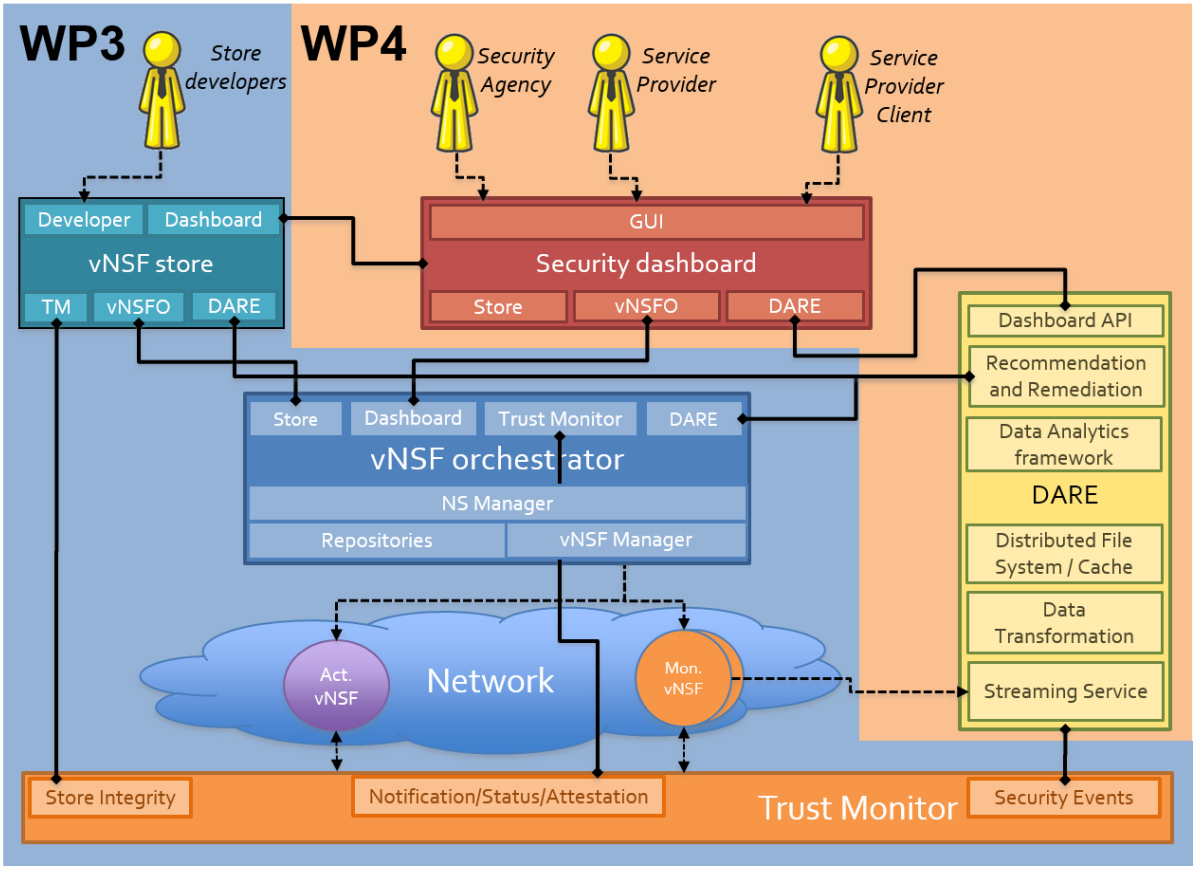

Figure 4: High-level architecture of SHIELD, with components per WP

Based on these use cases and the requirements highlighted in Deliverable D2.1, the designed high-level architecture for the SHIELD platform is articulated around six different components, illustrated in Figure 4 (vNSFs, Trust monitor, vNSF Orchestrator, vNSF Store, Security overview dashboard and DARE). The components corresponding to WP4 are described at a low level in this deliverable. From the point of view of the usable information-driven engine (WP4), the DARE stores and analyses the security logs and events provided by the network services (NSs) and vNSFs running in the network and these results are presented to the operator in the Security overview dashboard. These components collaborate with the vNSF ecosystem (WP3), specifically with i) the vNSF Store, which holds a registry of NS and vNSF-related information; ii) the vNSF Orchestrator, which deploys and manages the lifecycle of the NSs and vNSFs; iii) the monitoring vNSFs which produce the information to detect the threats; and iv) the Trust Monitor, which verifies that both NSs and vNSFs, as well as other nodes from the infrastructure, are trusted at all times.

The high-level design presented in D2.1 states that the network infrastructure provides a trusted environment for supporting the execution of vNSFs. For attestation purposes, the network infrastructure interacts with the Trust Monitor in order to authenticate the integrity of each network component. The network infrastructure is interconnected with the vNSF

Orchestrator through the vNSF Manager Engine. This interaction allows the deployment of vNSFs, the vNSF lifecycle management and the collection of monitoring information. Moreover, in case of attestation incidents, the Trust monitor can act on the network to solve the issue and inform to the DARE of the arisen situation. Monitoring vNSFs inspect captured data and provide valuable information to the DARE. The network status is reported periodically since events, not detectable by individual vNSFs, are inferred by centralising all the information in the DARE. Then, the data analytics framework analyses all the heterogeneous network information previously collected via monitoring vNSFs and Trust monitor. It features cognitive and analytical components capable of predicting specific vulnerabilities and attacks. Finally, the remediation engine provides recommendations in the form of new network services (sets of vNSFs) or medium level policies (configurations of existing VNSFs) to remediate the detected threats. These recommendations and the attack information is given to the intuitive and appealing graphical user interface implemented in SHIELD, which allows authenticated and authorised users to access SHIELD's functionalities. From this dashboard, operators have access to monitoring information showing an overview of the security status. Furthermore, this dashboard allows operators as well as tenants to take actions and react to any detected vulnerability.

In D3.1, a detailed view of the vNSFs ecosystem components is provided. However, with the aim to provide self-explained deliverables, we summarise their main functionalities:

There are two types of vNSFs in SHIELD, monitoring and reacting vNSFs. Monitoring vNSFs are configured to send, in an efficient manner, the collected traffic to the DARE, while reacting vNSFs are configured to stop an ongoing attack or to remediate a detected vulnerability. However, this classification is not strict; many vNSFs are exposing both capabilities (i.e. monitoring and reacting).

The vNSF Store acts as a nexus between the vNSF Orchestrator and third-party vNSF providers/developers, who can register and manage vNSFs to be available through the SHIELD platform. The Store handles all the vNSF data related with the service, the software images and the information required to validate the integrity of itself.

The vNSF Orchestrator, is responsible for managing the lifecycle of vNSFs. Among others, this allows to deploy (instantiate and place) vNSFs in specific points of the network infrastructure. The vNSF Orchestrator interacts with each of the other modules to obtain data on the vNSFs, to receive deployment requests or to convey information of specific vNSFs to enable analysis processes.

The Trust Monitor is the component in charge of monitoring the trust of the SHIELD infrastructure. This is achieved by a combination of authentication and integrity: each node joining the infrastructure must be properly authenticated and provide also a proof of the integrity of its software stack, by leveraging Trusted Computing (TC) mechanisms.

2. DESIGN AND ARCHITECTURE

In this section, the design and architecture of SHIELD are presented for those components defined within WP4, i.e. the DARE and the Security dashboard. This description is more detailed than D2.1, as it specifically addresses low-level details, as the subcomponents for the vNSF environment, their detailed workflows and relation between these and the other components in the SHIELD platform.

2.1. Low-level Architecture

As explained in Section 1, the high-level design elaborated in D2.1 has stated that SHIELD is composed by 6 components. Four of them belong to WP3 (vNSFs, Orchestrator, Store and Trust monitor), while two of them belong to WP4 (DARE and Dashboard). Although D2.1 was only exposing high-level architecture and design, there were some references to the envisioned subcomponents. In particular, the high-level architecture was exposing a Data Services Centre for data format transformation, an Enterprise Service Bus for data transportation, etc. However, as D2.1 was high-level architecture, these subcomponents, which are actually part of the low-level design, were not described in detail.

During T4.1 the consortium has studied conscientiously the requirements elicited in D2.1 together with the specifications needed to fulfil them. Moreover, several technologies that are currently used for threat detection and remediation have also been considered. The results of this detailed study extensively explained in Section 3, imply some adjustments in the low-level architecture of the modules and hence, some changes in the envisioned components from D2.1.

2.1.1. The DARE

The Data Analysis and Remediation Engine (DARE) is one of the three central innovation pillars of SHIELD, together with the vNSF ecosystem and the hardware attestation (both in WP3). The DARE centralises the management information of SHIELD and exchanges information with all the other components of the solution, as described below:

- the vNSFs, since the DARE centralises the information obtained from the monitoring vNSFs;
- the vNSF Orchestrator, since the DARE needs information of the network per tenant (e.g. ISP, ISP clients using SecaaS or a Cybersecurity agency) in order to provide accurate and complete recommendations;
- the Trust Monitor, since the DARE needs to know if a vNSF, or even a complete node, has been compromised;
- the Dashboard, since the DARE notifies to the dashboard the detected network anomalies and one or more recommendations of network services (set of vNSFs) with their appropriate high-level policies for configuration;
- the Store, since the DARE needs to know the vNSFs and NS for deployment or reconfiguration.

The data flow diagram of the DARE is shown in Figure 5.

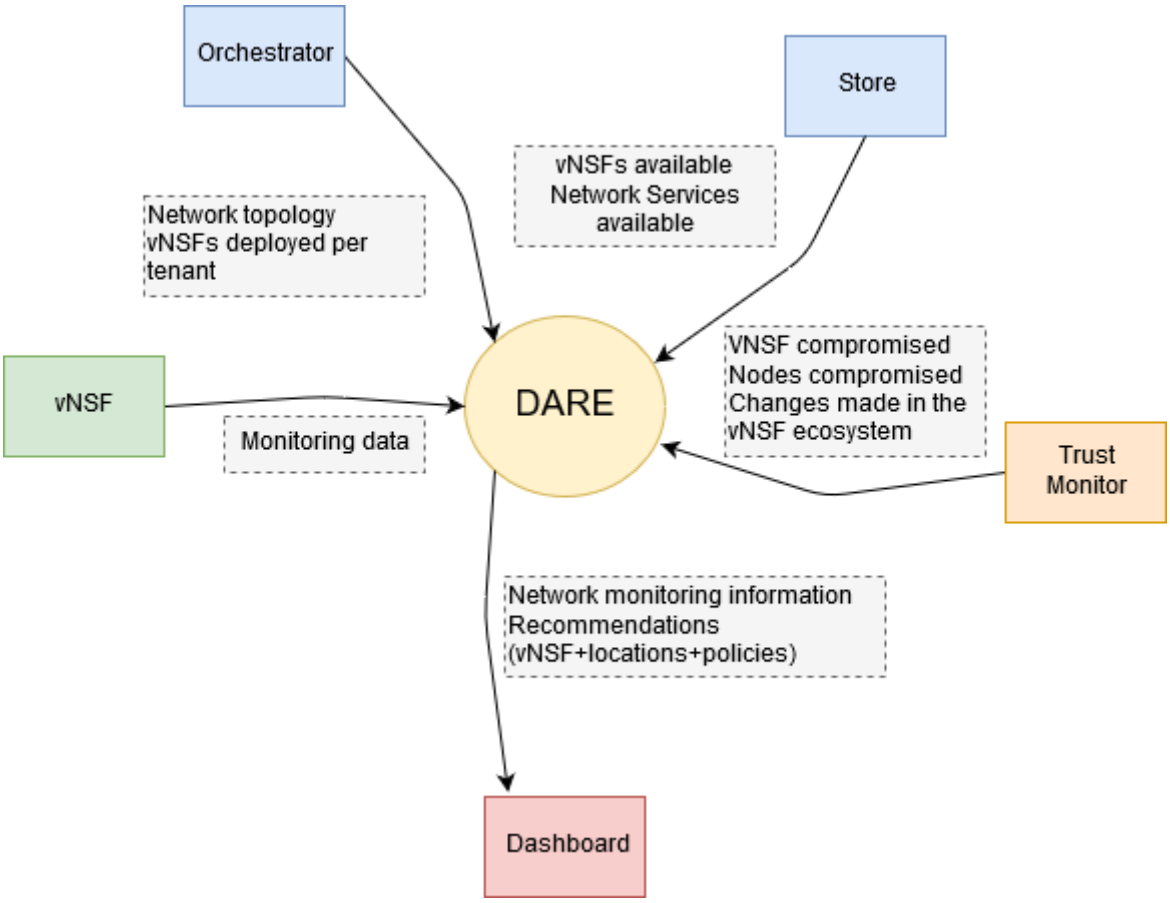

Figure 5: Data flow diagram of the DARE

The DARE will be composed by a central data analytics engine and a distributed set of data collection components. It is worth mentioning that it has been designed following a big data approach where the data value elicitation is divided into three different phases, as shown in Figure 6:

1. Data acquisition and storage
2. Data analysis
3. Cybersecurity topologies

In this section, each one of the subcomponents of the DARE will be explained and the differences with the ones envisioned in D2.1 will be detailed.

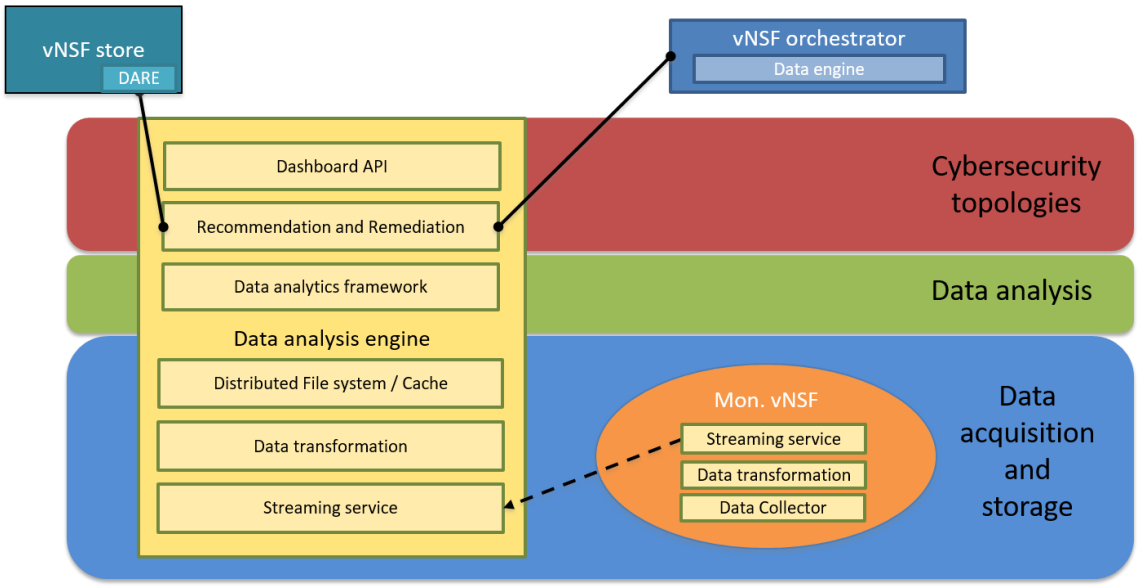

Figure 6: Architecture of the DARE

Data Collector: it is responsible for acquiring the data generated in a monitoring vNSF, using the specific format of the technology provided by such vNSF. The collector is part of each monitoring vNSF and integrated into the diagram for clarification purposes.

Data transformation: it is responsible for transforming the format-specific data into a generic format.

Streaming service: it will send the information from the monitoring vNSF to the data analytics central engine, assuring reliability on the communication.

Distributed File System / Cache: it will be responsible for storing the collected data for both, batch (i.e. hard disks) or real-time (i.e. cache) processing.

Data analytics framework: it is responsible for classifying the traffic for anomaly detection using machine learning techniques.

Recommendation and remediation: it will propose, given a specific anomaly or threat detected, a set of vNSFs with the appropriate policies to be deployed in specific places of the network.

Dashboard API: it will push all the generated information to the Dashboard.

2.1.1.1. Updates since D2.1

Hybrid architecture

In D2.1 the DARE functionalities were completely centralised with the envisioned Enterprise Service Bus (ESB) managing the communication between the central DARE and the vNSFs. According to the research done during T4.1, it has been considered that some subcomponents were missing from the architecture. The data collector module is a good example of this situation. This module works with specific formats in the sense that it accepts data from different formats (for example data from DNS and data from a proxy). Hence, one collector must be developed for each format accepted in SHIELD. Moreover, to avoid flooding the network, we consider the possibility that some uncompressed format specific files can be transformed into generic formats which can be compressed before being sent to the central

engine. Hence, the Data transformation subcomponent (which replaces the Data Services Centre, a name that could lead to confusion) can either be distributed in the case of specific not compressed formats, or centralised for specific formats that already have minimum size.

Moreover, to avoid confusion with commonly defined terms and because the data has been already transformed, the storage module is no longer called “Staging” (which commonly refers to raw and heterogeneous data storage module) but is now called “Distributed File System”.

A streaming service

The envisioned ESB has been replaced by a simpler streaming service. The reason is that SHIELD does not need some of the advanced services provided by ESBs like multi-tenant communications (more than one endpoint for each channel), synchronisation services or bidirectional flows. Instead of this complex system, a streaming service which only deals with reliable message delivery is considered to be enough.

A more complete data analytics framework

The Complex Event Processor (CEP) as well as the processing area have been incorporated in the data analytics framework since it is precisely the machine learning algorithms provided by the data analytics framework the ones that will classify the traffic and manage the events.

A recommendation and remediation engine with a global view

The SHIELD consortium considers that this subcomponent must use not only the information given by the data analytics framework, but also the global view of the system. Hence, the recommendation and remediation engine will consider all the variables needed to recommend or remediate a cybersecurity vulnerability:

- The threats and attacks detected by the data analytics framework.
- The different types of vNSFs provided by the Store and the policies that can be applied to them.
- The vNSFs already deployed in the network.
- The network topology.

Using this central position, it will recommend network services (sets of vNSFs) together with the policies and the specific deployment locations with the objective to remediate the detected attack.

2.1.2. The Security Dashboard

The Dashboard has been designed to be the unique interface with the users of the platform. Hence, the Dashboard must unify all the needs of the users for all the SHIELD use-cases (ISP, SecaaS, and cybersecurity agency).

The low-level architecture presents the different subcomponents that compose the Dashboard as shown in Figure 7.

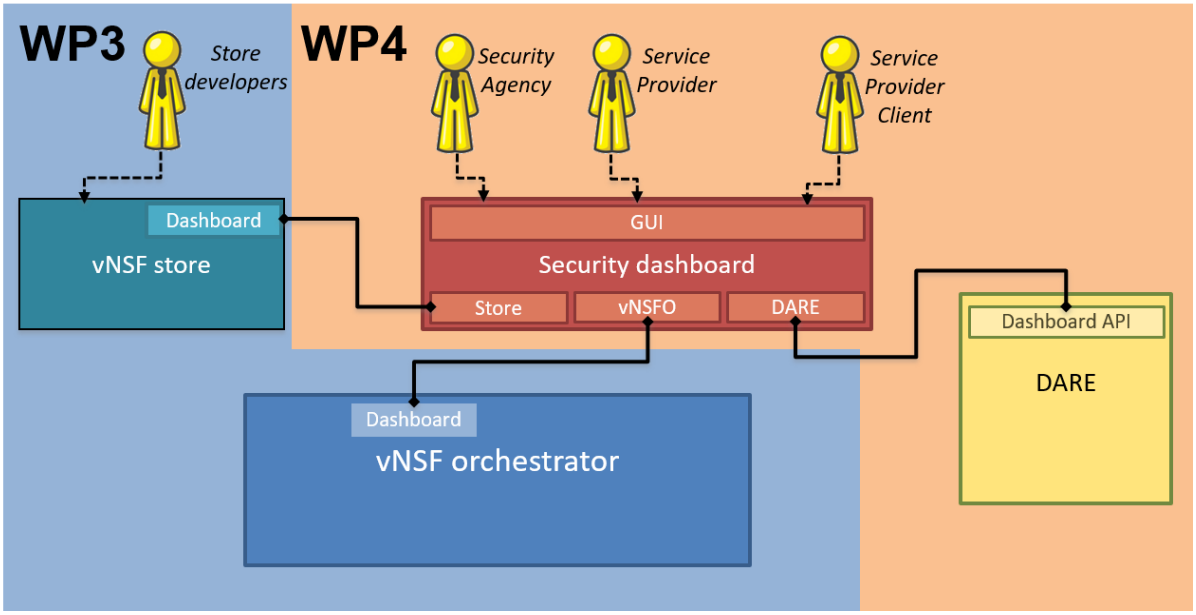

Figure 7: Subcomponents of the Security Dashboard

2.1.2.1. Updates since D2.1

The main difference in respect to D2.1 is the unification of the interfaces for the three use-cases. As explained in detail in Subsection 2.2, the difference between the three use-cases is mainly the level of access of the different users. Hence, it is not needed to have one interface per use-case (user) but a single interface that controls the rights of each user.

2.2. Design

SHIELD is committed to solve three relevant use-cases. Firstly an ISP which uses SHIELD to secure its own infrastructure. Secondly, an ISP using SHIELD to provide SecaaS to their clients. Thirdly, a cybersecurity agency collaborating with an ISP to research on attacks and vulnerabilities.

This means that the SHIELD must take into consideration the following user interactions:

1. Allow users to deploy network services (after payment, if necessary).
2. Allow users to withdraw network services.
3. Provide insights about what is happening in the network to detect anomalies.
4. Provide a system to display and dispatch DARE recommendations in the form of network services including (completely or partially):
    a. The detailed set of vNSFs recommended.
    b. The description of the specific places of the network where they will be deployed.
    c. The policies that will be used to configure the vNSFs.
5. Allow privileged users to implement monetisation methods.

These features can be modelled as user use-cases (called user cases to avoid confusion with the SHIELD use cases), showing also the components that are related to each one of them (Figure 8). Note that the three SHIELD use-cases share most of the envisioned features. The difference between them is the level of access (e.g. the ISP will have access to all the infrastructure and vNSFs while the ISP client has access only to their subnetwork and to the paid services).

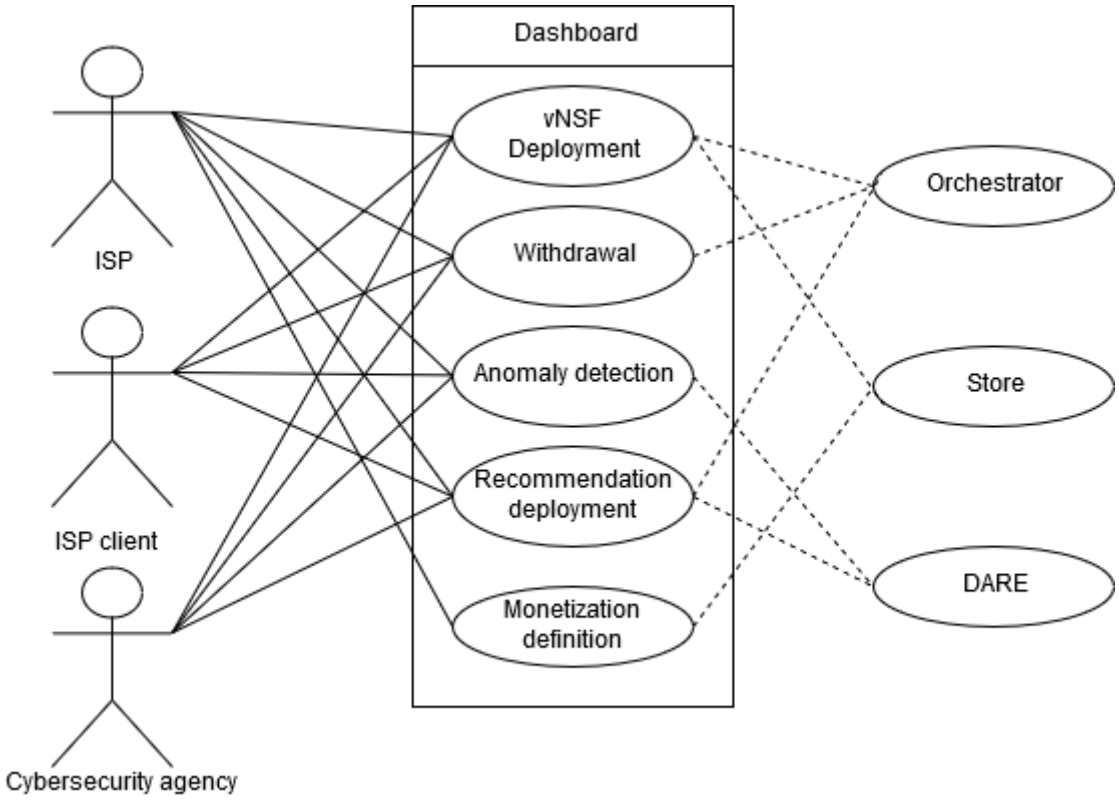

Figure 8: UML user case diagram.

In this section all these user cases will be explained in detail and the differences between the SHIELD use cases will be also stated.

2.2.1. vNSF Deployment and Withdrawal

The processes of deployment and withdrawal of vNSFs are extensively explained in D3.1 since most of the tasks are done by the components developed in that WP. However, the dashboard also plays a role in this process and hence, we will explain them only from the WP4 perspective.

All the SHIELD use-cases require these functionalities:

- An ISP must be able to deploy or withdraw any available vNSF in any available location in the Network.
- An ISP client must be able to deploy or withdraw paid vNSFs in the gateway that connects the client's subnetwork with the general ISP network.
- A Cybersecurity agency must be able to deploy or withdraw specific vNSFs (allowed by the ISP) to specific locations of the network (also allowed by the ISP).

From the WP4 perspective, the process is very simple and it is shown in Figure 9.

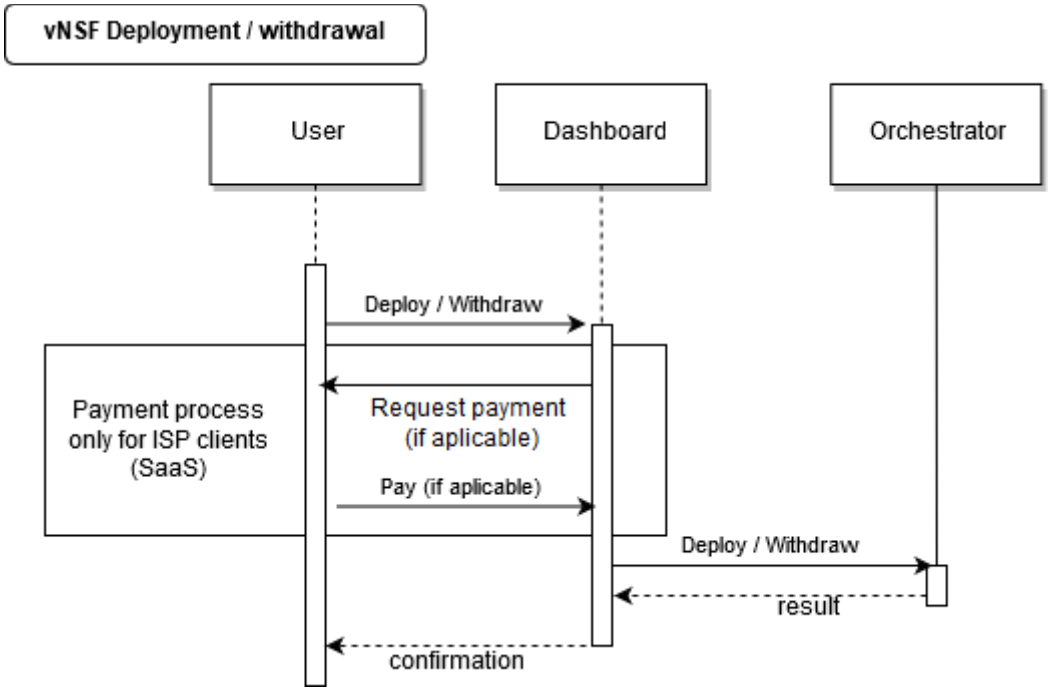

Figure 9: Flow diagram of the vNSF deployment / withdrawal task.

2.2.2. Anomaly Detection

The process of anomaly detection is divided in two different phases. In phase one, information is collected from the monitoring vNSFs. This information is stored in the Distributed File System subcomponent for further analysis. In the case of real-time analytics, the information will be loaded into cache instead of stored in hard disks, but the workflow is identical. As already explained, the Data Transformation subcomponent can either be distributed, where data is transformed to a generic format (e.g. CSV) before being sent to the central engine (Figure 10); or centralised, so specific formats (e.g. PCAP files) are sent to the central engine (Figure 11).

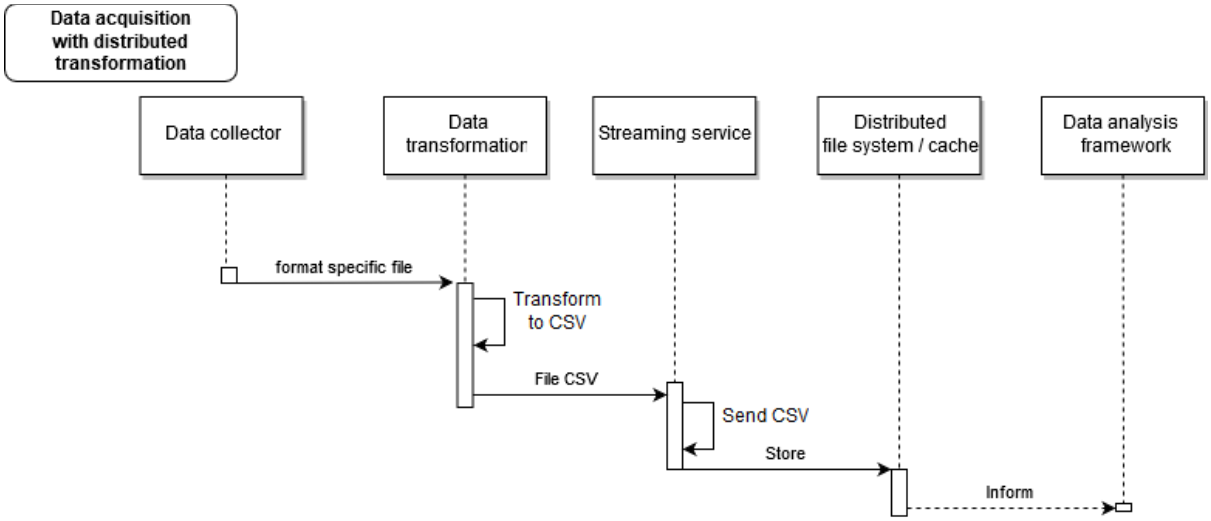

Figure 10: Flow diagram of the data acquisition phase with distributed transformation.

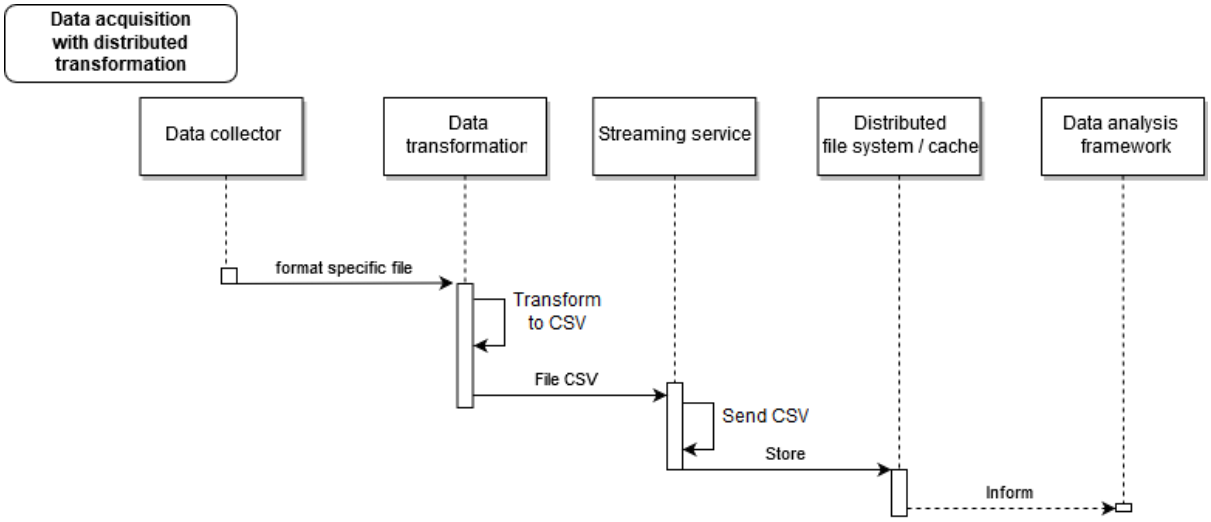

Figure 11: Flow diagram of the data acquisition phase with distributed transformation.

Once the data has been loaded into the distributed file system (either in batch or real time) the phase two (Figure 12) implies that the data analysis framework will use the machine learning algorithms to detect anomalies and inform the remediation and recommendation subcomponent. At its turn, this subcomponent will use this information, together with the information gathered from the vNSF Orchestrator, to provide recommendations to the user in the form of specific network services with the configurations and the locations to be deployed.

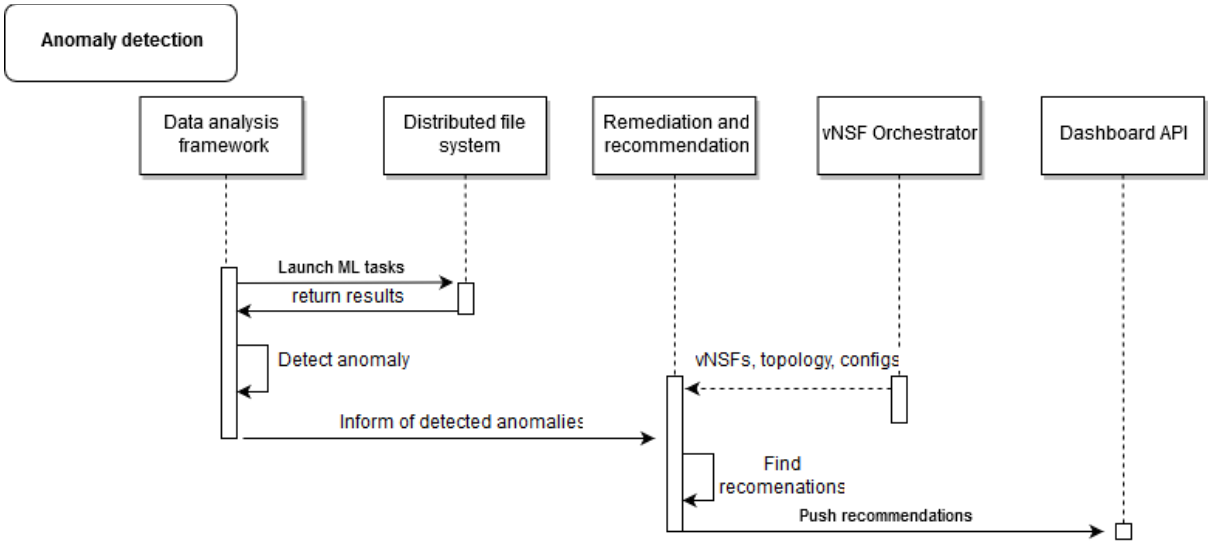

Figure 12: Flow diagram of the anomaly detection phase.

2.2.3. Recommendation deployment

The process of recommendation deployment is driven by the user. Firstly, it has to decide if automatic remediation will be allowed (so that the system applies automatically the most accurate recommendation) or if every recommendation must be explicitly deployed by the user. If no automatic remediation is chosen, the user has to decide which one of the provided recommendations will be deployed. The process is shown in Figure 13.

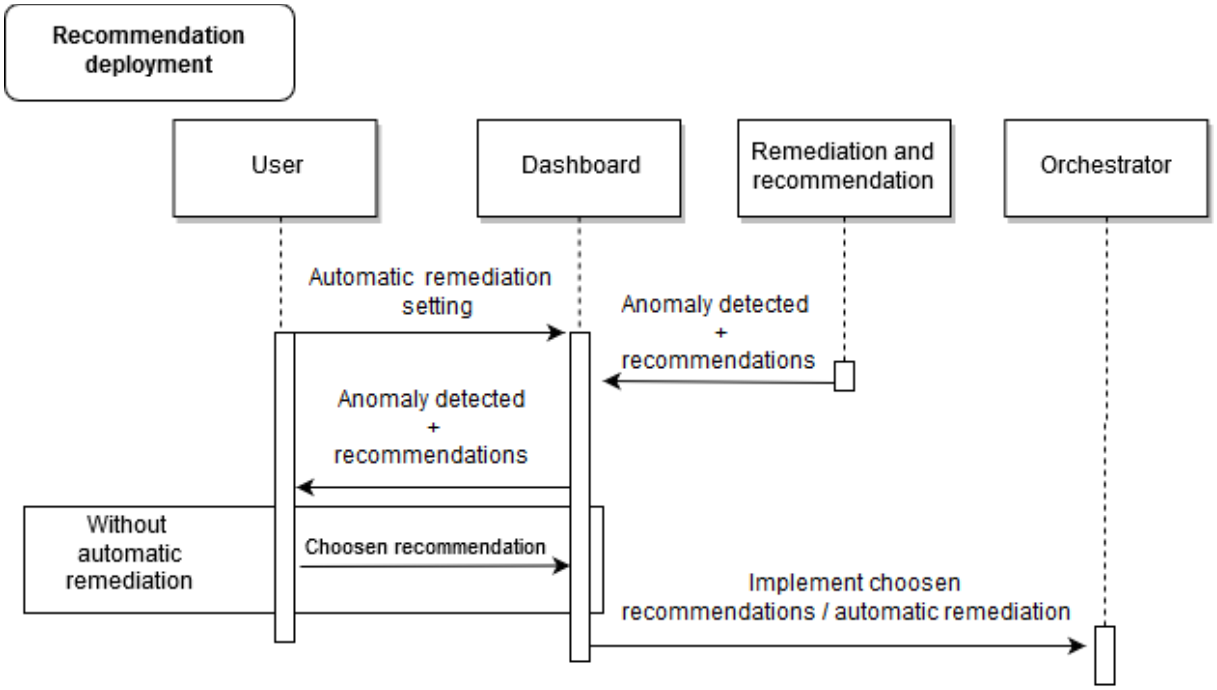

Figure 13: Flow diagram of the recommendation implementation.

2.2.4. Monetisation definition

In the SecaaS SHIELD use-case, the ISP client can buy specific vNSFs to be deployed in the gateway that joins the client network with the ISP network. It is important to note that, apart from the direct access to the information gathered from its own monitoring vNSFs, the ISP may allow the system to provide recommendations to the user based not only on the user information but also on a more general view of the network.

The monetisation system will be variable. It can be based on a "pay per use", price per vNSF, or "flat rate" that includes all the needed monitoring and mitigation vNSFs to solve a security incident. Business model will be decided later in D6.2, but it is important to include as part of the design some metadata required. Several parameters must be included. Price, license types, terms of use, period of validation are some potential options. All this parameters will be supported as optional and included in the metadata of the vNSF, within its package. Some examples can be the price defined by the vNSF developer and added as metadata in the vNSF Store (WP3). Another example would include the price defined by the ISP, who can charge the user by adding fixed price to a vNSFs and/or define a subscription system for using the SecaaS. The workflow of this task is shown in Figure 14, BSS is an acronym for Business Support Service, an API used to implement the business models.

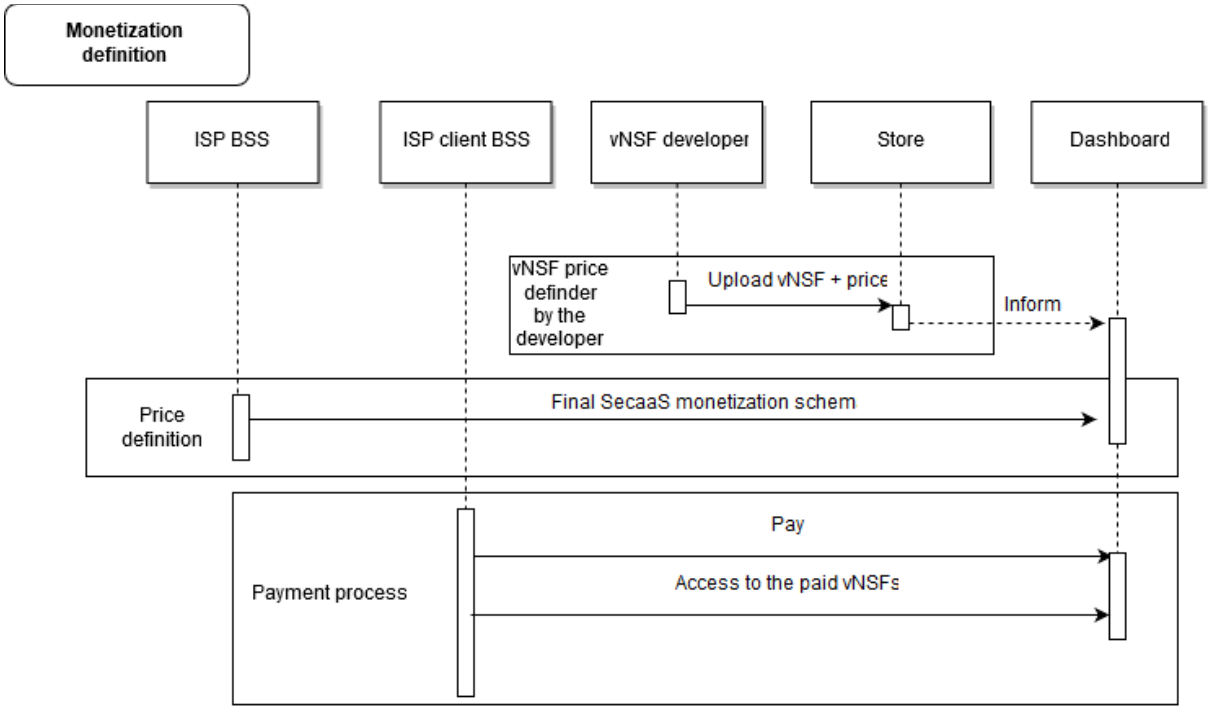

Figure 14: Flow diagram of the monetisation definition.

2.3. Data acquisition and storage Phase

The Data Acquisition phase is responsible for the efficient and reliable capture and storage of various heterogeneous data. It will involve mechanisms and methods in order to capture and transfer files generated by network tools to the central data analytics engine. This phase is of high importance for ensuring the integrity of the data and their quality in further processing steps.

Heterogeneous network information is captured via specialised vNSFs, which collect overall networking events that are relevant to threat detection. This information is transferred to the central data analytics engine, where it is stored for further processing. This phase will gather, transform and store the acquired network data to a format that can be processed by analytics components.

There are two (2) options for describing the low-level architecture of the Data Acquisition and Storage phase:

- Option 1 – Centralised architecture (Figure 15): only the collection of the data is distributed, while all the other functionalities are centralised in the Data analysis phase.
- Option 2 – Distributed architecture (Figure 16): the data collection and the data transformation are distributed per vNSF and hence, the data is sent to the central engine in a standard format (e.g. CSV).

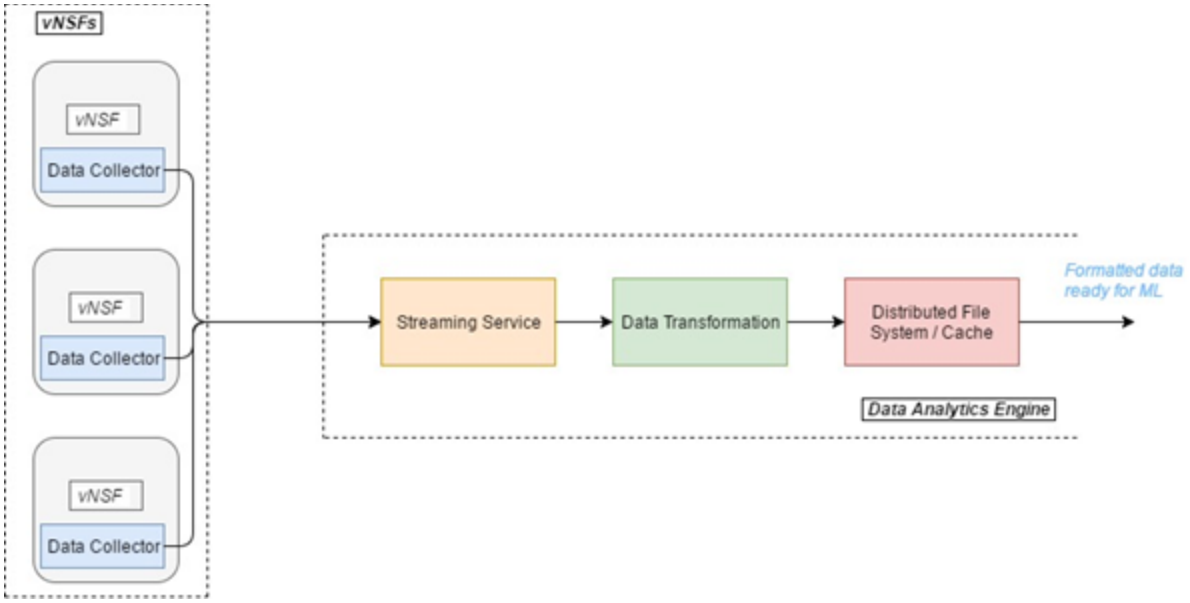

Figure 15: Centralised low-level architecture.

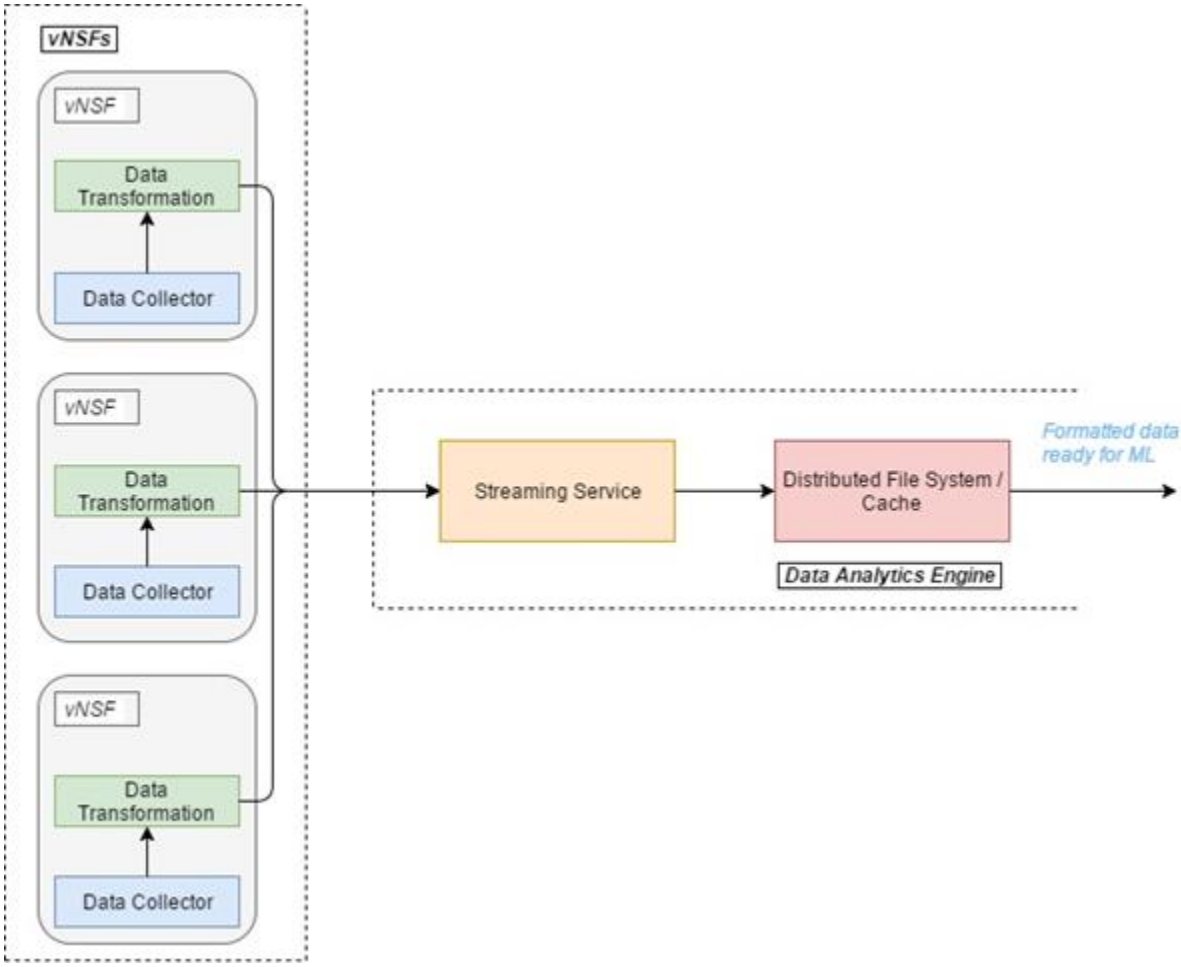

Figure 16: Distributed low-level architecture.

The objective of the data acquisition phase is to gather all the network information produced by the vNSFs, transform it into a generic format, ingest it into the data analytics central engine,

and store it for further processing. The low-level architecture of the Data Acquisition and Storage phase is divided in four main subcomponents.

Data Collector

The data collector subcomponent is the only one which is distributed (one collector per vNSF) in both options. Each vNSF uses a daemon, called data collector, which is responsible for monitoring the vNSF and detecting new files produced by it.

Streaming Service

The acquisition of network data is achieved via a distributed streaming service that splits the network data into smaller specific topics and smaller partitions, while creating a data pipeline for each topic. It has to be reliable and fault tolerant for ensuring the integrity of the data and their quality in further processing steps.

Data Transformation

This is the subcomponent that determines the chosen option for the low-level architecture. If the data transformation is centralised, the architecture will be considered to address option 1, while if it is distributed, the architecture will be considered to address option 2.

In the case of option 1, each pipeline created by the streaming service transfers the stored data to specific daemons which exist inside the central data analytics engine. These daemons are subscribed to a specific topic and partition of the streaming service, and transform the raw network data into a human-readable format, by using dissection tools. They are tasked with reading, parsing and storing the data in a specific distributed format to be consumed by the machine learning algorithms.

In the case of option 2, each network data file that is captured by the Data Collector, is sent to specific daemons which exist inside each vNSF (distributed). These daemons transform the raw network data into a human-readable format, by using dissection tools.

Distributed File System / Cache

Once the network data has been transformed, the input is stored in a distributed file system in both the original and modified formats (in the case of option 1) or only the modified (in the case of option 2). The distributed file system is responsible for storing the collected data and making them available, so that it can be accessible by search queries.

2.4. Data analysis phase

The data analysis phase is composed only by one subcomponent, the data analytics framework. This subcomponent features cognitive and analytical functionalities capable of detecting network anomalies that are associated with specific vulnerabilities or threats, offering batch and streaming incident detection. The processing and analysis of large amounts of data is carried out by using Big Data analytics and machine learning techniques. By processing data and logs from vNSFs deployed at specific strategic locations of the network, the data analytics framework is able to link traffic logs that are part of a specific activity in the network and detect any possible anomaly. In case malicious activity is detected, it informs the remediation and recommendation engine.

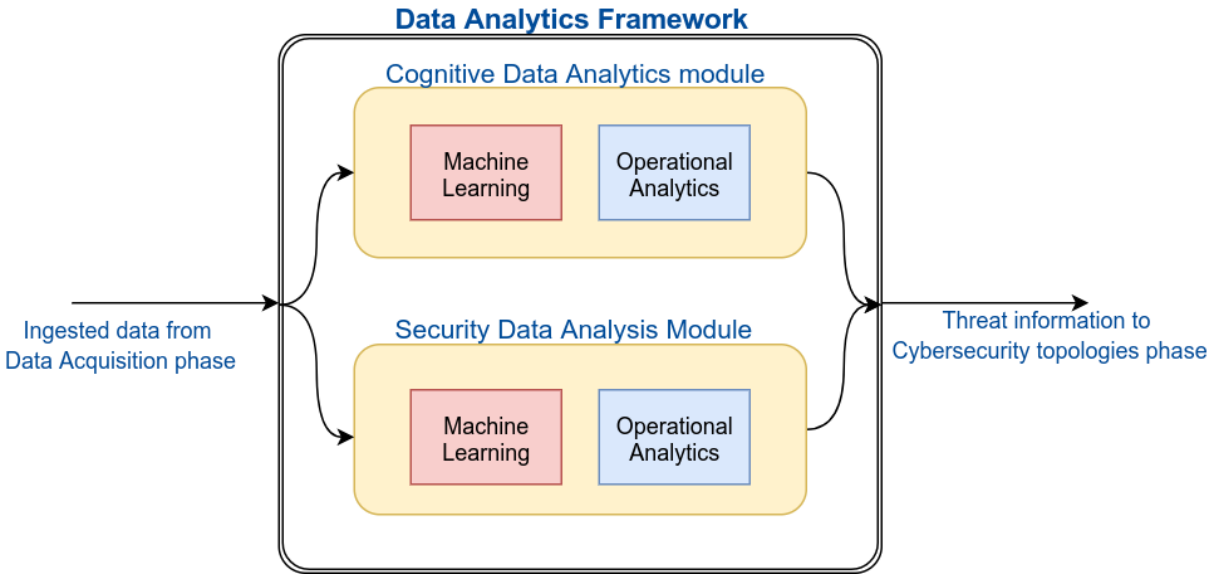

Figure 17: Data Analytics Framework overview.

The Data Analysis Engine leverages two different data analytics modules (Figure 17) (while remaining open for the inclusion of others in the future), that use a wide range of complementary detection techniques along with open source frameworks and solutions, a cognitive Data Analytics module based on open-source technologies and a proprietary Security Data Analysis module. The individual functionalities of each module are described in detail.

Since the network data provided by the Data Acquisition phase is required for the functionality of both the cognitive Data Analytics module and the proprietary Security Data Analysis module in the same distributed format, the accessing method is shared between these two modules.

Below we describe the two main Data Analytics modules that comprise the DARE, namely the **cognitive Data Analytics module** and the **Security Data Analysis module**.

Cognitive Data Analytics module

The cognitive Data Analytics module is an entity of elements that is able to produce packet and flow analytics by using scalable machine-learning techniques. To this end, it involves state-of-the-art Big Data solutions as well as the latest distributed computing technologies to allow batch and stream processing of large amounts of data, scalability and load balancing, utilisation of open data models (ODM) and concurrent running of multiple machine-learning applications on a single, shared, enriched data set. The above technologies will ideally allow for tailor-made security analytics and will lead to predicting attacks by correlating network anomalies to specific threats. The cognitive Data Analytics module consists of two main entities that comprise the overall detection procedure. These discrete entities are shortly referred to as machine learning and operational analytics and are configured either as separate computational nodes (physical or virtual machines) or as a part of a larger distributed computing system. A description of each entity is given below:

- **Machine learning**
  The machine learning entity (Figure 18) is responsible for the detection of anomalies in network traffic that will lead to the prevention or mitigation of potential threats. For this purpose, DARE uses a combination of open-source tools to run scalable machine learning algorithms. The machine learning entity works not only as a filter for

separating bad traffic from benign, but also as a way to characterise the unique behaviour of network traffic. It contains routines for performing suspicious connections analytics on flow, DNS, proxy logs, security event data and metrics gathered from the Data Acquisition phase and the built-in Distributed storage system subcomponent. These analytics consume a collection of network events in order to produce a list of the events that are considered to be the least probable, and these are considered the most suspicious. Below are listed the main types of analytics that are utilised by the machine learning component inside a cluster computing framework.

Anomaly Detection algorithms: The statistical model being used for discovering incongruences or rare behaviours by examining network traffic.

Additional algorithms: The machine learning entity is open for the inclusion of additional algorithms to enhance the overall detection capabilities of the platform as well as allow the correlation between the detected anomalies and specific threats by classifying the results of the anomaly detection algorithms.

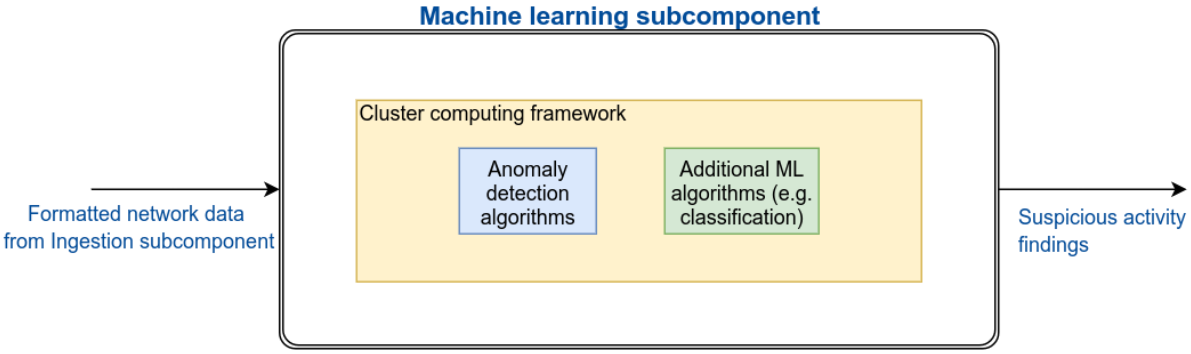

Figure 18: Machine learning entity overview.

- **Operational Analytics (OA)**

  The OA entity (Figure 19) consists of a number of context enrichment, noise filtering, whitelisting and heuristics processes to produce a list of the most likely patterns which may comprise security threats. It provides utilities to extract and transform data, by loading the results into output files. It supports basic data types such as flow, DNS and proxy logs that correspond to the most common types of network threats. The output of the OA entity can be used in the Remediation Engine, to provide recommendations to the users or to optionally activate task-specific countermeasures in the form of security functions from the vNSF Store. It also offers a combined view of the above information in the form of information to be pushed to the Dashboard for better visualisation purposes. A description of each element is given below.

  Threat interaction tools: An interactive tool that allows for a comprehensive interaction with the network anomalies detected by the machine-learning component.

  Ingest summary: It presents the amount of network data that has been ingested on the cluster.

  Filtering tools: A set of tools that provide the ability for customised results based on time, source/destination, severity, type etc.

  Whitelisting tools: A convenient set of tools to exclude some of the detected anomalies from the results, thus dealing with potential false-positives.

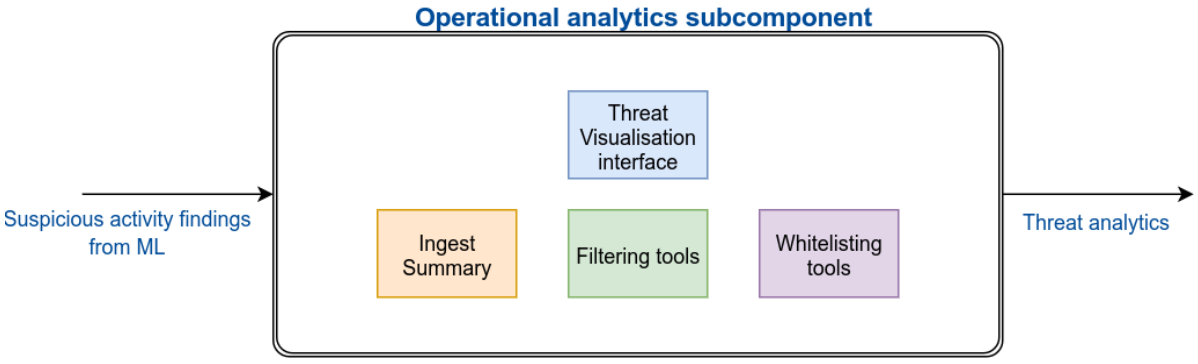

Figure 19: Operational analytics subcomponent overview.

Security Data Analysis module

The Security Data Analysis module (Figure 20) is an entity based on a combination of Big Data analytics and machine learning techniques that can efficiently process and analyse a vast amount of network data online and automatically discover and classify cybersecurity threats. This engine follows the architecture designed by the commercial network anomaly detector commercialised by Talaia Networks, which has received very good feedback from customers around the world because of its comprehensive detection of network attacks and its low false positive rate.

The Security Data Analysis module has four subcomponents that are described in more detail later. Briefly, this module receives the network data from the distributed storage system /cache subcomponent. The main input used is flow aggregated data (e.g., NetFlow), however, it also can utilise other sources of information (e.g., DNS, network logs). This data is provided to the Event Clustering and the Service Profiling subcomponents that compute several measurements and statistics that are used in the Alarm Correction subcomponent to detect anomalous behaviours related to security issues. Once an anomalous behaviour is detected the Anomaly Classifier subcomponent is in charge of classifying it among different network attacks (e.g., DDoS, network scan). All this information is then outputted from the Security Data Analysis to the Remediation module.

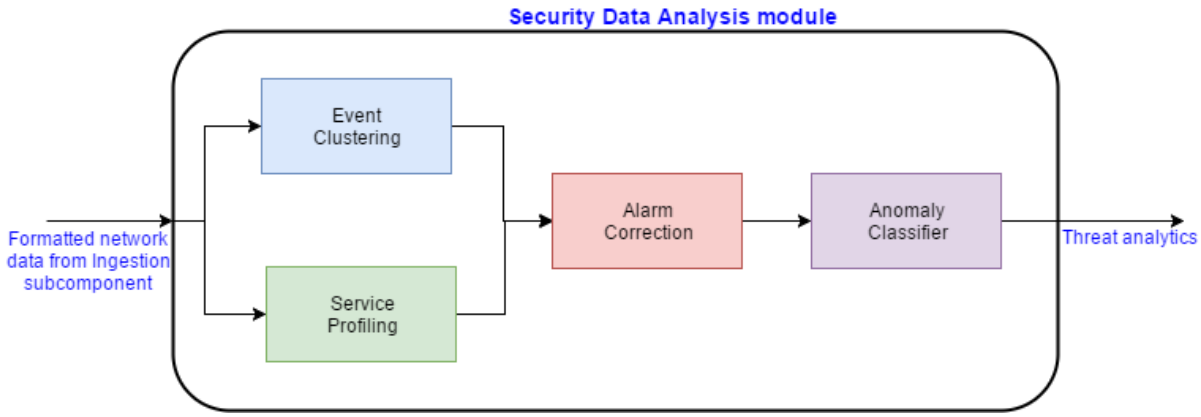

Figure 20: Security Data Analysis module overview.

- **Event Clustering**
  The Event Clustering entity consists of the adaptation of data mining techniques able to discover multi-dimensional patterns of network usage in the data provided. This entity

detects clusters of events in the data which are frequent and share a specific behaviour. The detected clusters, although they can be related to benign traffic, are suspicious of being anomalous traffic that can be related to security issues. The data involved in the cluster and the specific statistics and measurements shared by the events of the cluster are sent to the Alarm Correction in order to decide if the detected cluster is actually related to a security issue.

- Service Profiling
  The Service Profiling entity performs a thorough and detailed analysis of the data provided in order to create profiles of all the services contained in such data. To perform this analysis in real-time, the engine is relying on extremely efficient data structures combined with cutting-edge data mining techniques. The objective of this entity is to identify and understand the behaviour of the different services in the data. In addition, this entity is also able to discover the services that are prone to be affected by networks attacks. Similarly to the Event Clustering entity, the information related to the services identified is then forwarded to the Alarm Correction entity for further analysis.
- Alarm Correction
  The Alarm Correction entity receives as input the output from the Event Clustering and the Service Profiling entities. By comparing and enriching this information with a proprietary methodology, this entity is able to identify anomalous behaviours in the traffic and discern with very high accuracy between benign and malign behaviours related to security issues. The malign behaviours detected and the data related to them are forwarded to the Anomaly Classifier entity.
- Anomaly Classifier
  Once a malign anomalous behaviour is detected by the Alarm Correction entity, the Anomaly Classifier entity is in charge of identifying the type of the anomaly. To that end, the Anomaly Classifier entity uses the measurements and statistics as input for a machine learning technique that is able to classify the anomaly between different volumetric attacks (e.g., DDoS) and zero-day attacks. All the resulting information is then provided to the Remediation module, that based on the characteristics and the anomaly type is able to better provide accurate counter-measurements to mitigate the security issues.

2.5. Cybersecurity topologies phase

The DARE includes two subcomponents on top of the data analytics framework which are in charge of defining the remediation and recommendations actions to be presented to the final user via the Dashboard. A Remediation action consists in a cybersecurity topology responsible for addressing a specific network security threat. These actions will ultimately be translated into a set of vNSFs with proper configuration and deployment location, therefore allowing its instantiation in the secure network environment. Configurations are specified as a set of high-level, technology-independent policies with a uniform description regardless of the targeted vNSF type or implementation.

The recommendation and remediation subcomponent is aware of the current state of the network infrastructure in order to optimise the security impact of the vNSFs of the different Network Services. This awareness is built upon information regarding running instances for both vNSFs and NSs retrieved from the Orchestrator per tenant. A cybersecurity topology will

be generated by a detected threat, which is converted into a high-level abstraction of a remediation recipe. However, the actual remediation is not to be performed directly in this subcomponent, which is not be in charge of directly modifying the status of the infrastructure, but to be proposed to the SHIELD operator via the Security dashboard (using the Dashboard API), that is in charge of accepting or declining it. If accepted, the remediation action is applied in the network infrastructure of the tenant through the Orchestrator, which also forwards the request for the translation of policies to low-level configurations, carried out within each vNSF.

2.5.1. The recommendation and remediation subcomponent

The high-level block diagram representing the architecture of the recommendation and remediation engine and its interaction with other SHIELD components is depicted in Figure 21. The engine's internal subcomponents are described as follows.

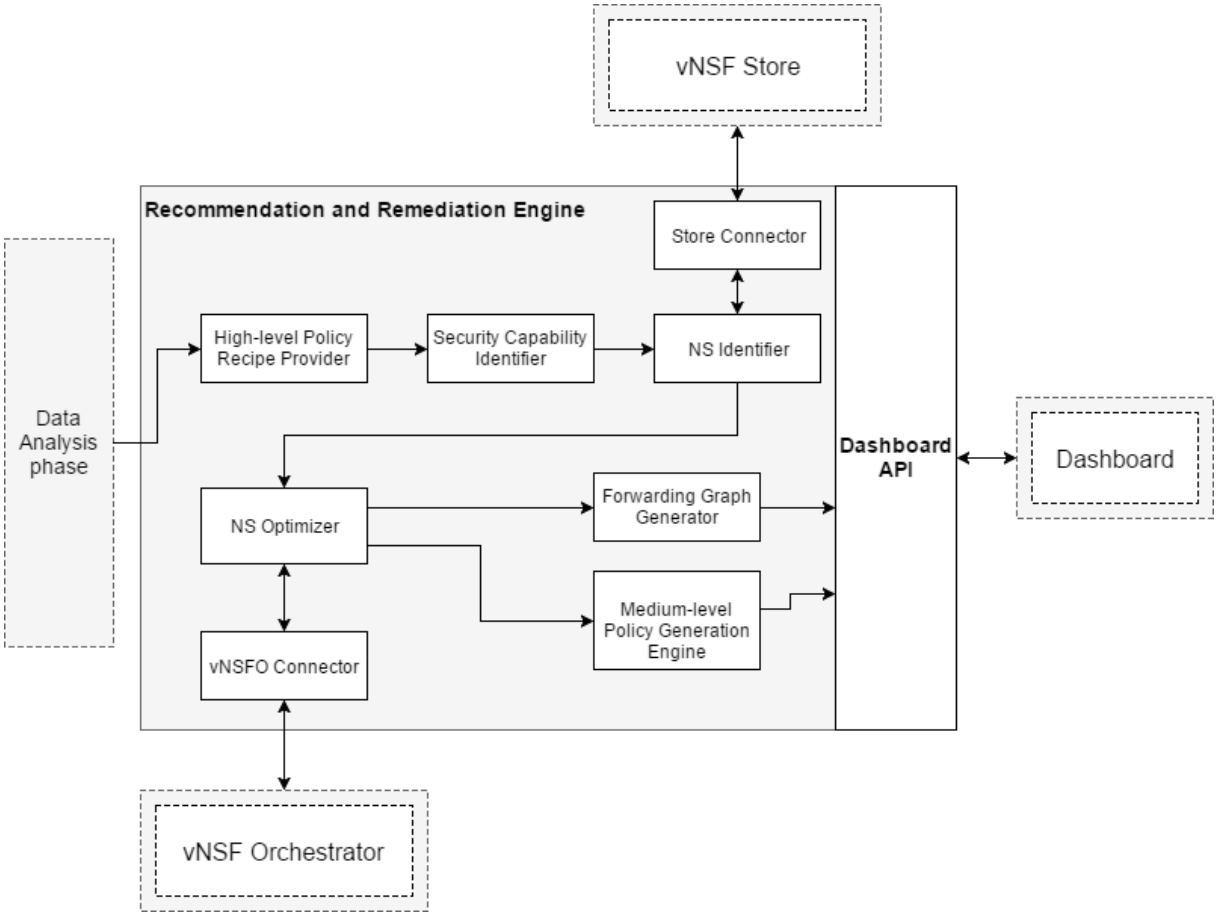

Figure 21: The recommendation and remediation subcomponents.

High-level Policy Recipe Provider

This module receives an event from the data analytics framework of the DARE, containing alerts and contextual information related to an occurring threat. Starting from it, this module will define a "recipe", consisting of a set of security requirements specified in a high-level policy

abstraction targeting the mitigation of the detected threat. The recipes will be stored in an internal repository, in order to allow their update/addition/removal by a security analyst.

Security Capability Identifier

Using the information created by the previous module, this module will be responsible for mapping each remediation high-level recipe to a set of security capabilities. A capability is defined as a basic feature that can be configured to enforce a security action (e.g. address translation, authentication, data protection, authorisation, routing, resource protection, resource analysis).

NS Identifier

This module is in charge of selecting the sets of vNSFs (the NSs) that match the required security capabilities. To do so, it requires the knowledge of the security capabilities offered by each of the NS in the NS Catalogue, hence it directly interacts with the vNSF Store.

Store Connector

This module is in charge of connecting the subcomponent with the vNSF Store, by consuming its API in order to retrieve the information about the available NSs in the catalogue.

NS Optimiser

This module will be in charge of selecting the best NS that matches the required capabilities, according to an optimisation criterion. In addition, this module will be able of verifying if a NS has been already deployed for the tenant, in order to optimise the deployment of the remediation by providing only the updated configuration.

vNSFO Connector

This module is in charge of connecting the subcomponent with the vNSF Orchestrator, by consuming its API in order to retrieve information about the already deployed NSs for the tenant.

Forwarding Graph Generator

This module is in charge of describing the list of vNSFs into a topological arrangement that would allow the vNSF Orchestrator to deploy them into the network infrastructure.

Medium-level Policy Generation Engine

This module generates the medium level policy abstraction starting from the Forwarding Graph and the list of capabilities identified to address the security threat. Each vNSF capability is associated to a policy, expressed in an application-independent syntax.

2.5.2. The Dashboard API subcomponent

This subcomponent consists of an interface in charge of presenting the final result of the recommendation and remediation decisions to the Dashboard. The information to be provided will consist of an optimised cybersecurity topology and a set of application-independent rules to implement the mitigation.

2.6. Security Dashboard

The Security Dashboard (Dashboard from now on) is SHIELD's topmost component being in charge of enabling users and third party applications to use SHIELD's internal features. Dashboard is therefore the entry point of SHIELD solution, seamlessly encapsulating the access and use of all its information and features in this component. Being the only point of access for the SHIELD users facilitate the integration and builds a more secure application, since the access control is more robust and protected. Besides integrating with all SHIELD's components, Dashboard is also responsible for the implementation of a set of support features. It will provide user and tenant management features, billing and monetisation capabilities as well as a remediation subcomponent responsible for persisting and dispatching (upon validation by authorised users) all SHIELD's remediation suggestions.

Displayed in Figure 22, is provided the currently envisioned Dashboard's internal subcomponents specified taking into consideration the requirements and responsibilities associated with it.

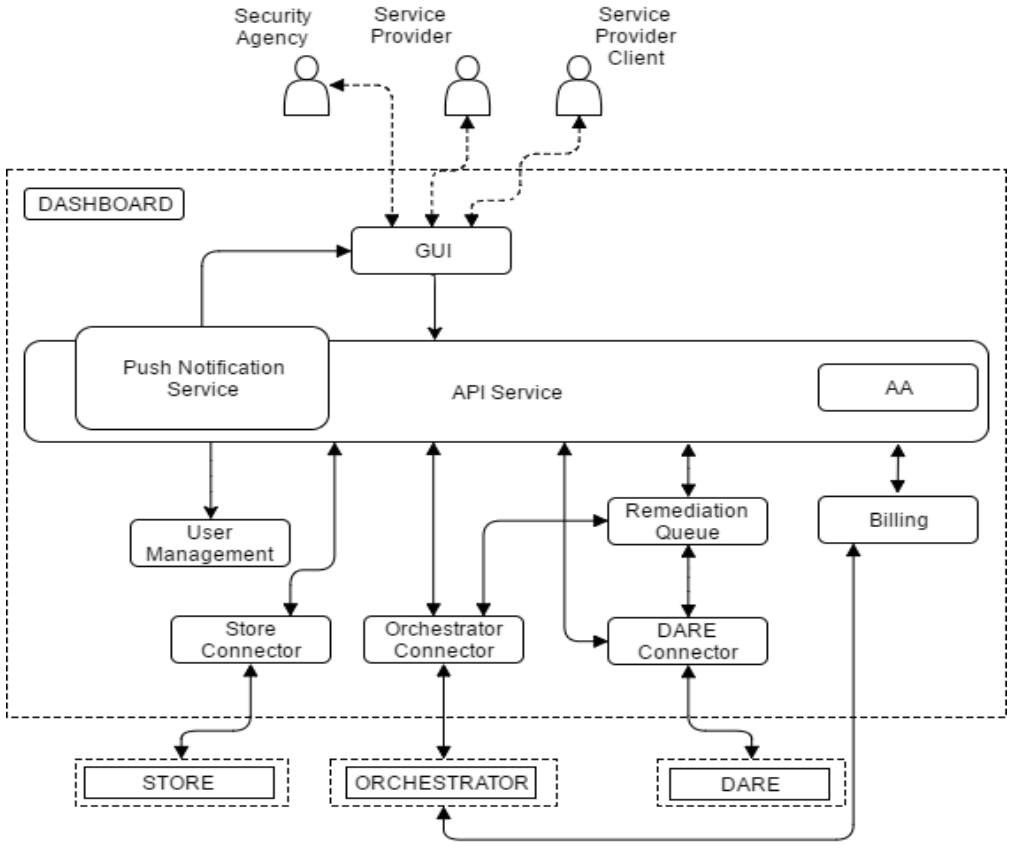

Figure 22: Dashboard internal subcomponents.

As depicted in the previous figure, Dashboard's architecture can be divided into multiple subcomponents, each one with a well-defined scope implementing its set of features. Each one of these subcomponents is described below allowing the understanding of both how this component interacts with other SHIELD components as well as the internal workflows of the Dashboard.

GUI

The Dashboard user interface will provide an appealing and intuitive web-based interface exposing the security-related features to end users. Different permission levels will be attributed to different users, thus allowing the graphical user interface to adapt the provided features to the ranking of the logged user. By using a web browser, end users will then be able to see SHIELD's monitoring system thus perceiving an overview of the security status of their services. Moreover, this interface will also display detected vulnerabilities as well as remediation suggestions that allow to mitigate each detected vulnerability. The end user will be able to analyse the proposed actions of a remediation and decide whether to apply or not the suggestion. Tenant, user and billing features will also be present in the security dashboard graphical user interface (taking into consideration each user's permission level) allowing to control the access to SHIELD's features as well as the monetisation of SHIELD's tools. This subcomponent, as illustrated in the previous figure will interact only with API Service as well as Push Notifications Service subcomponent providing an entry point for SHIELD's end users as depicted in the previous figure.

API Service

API Service provides an abstraction from the GUI as well as potential third party applications aimed to interact with SHIELD solution. It interacts with all Dashboard's internal subcomponents forwarding a given user request to an entity responsible either for implementing a feature or forwarding the request to another component. Following what was previously described in the current section, the authorisation of the type of request performed by the user will firstly be validated through authorisation and authentication features provided by AA subcomponent.

Push Notification Service

This subcomponent is responsible to push notifications/events from the bottom layers of SHIELD to either the graphical user interface or third party applications. The current subcomponent will therefore enable information visible by end users to be updated without the need of the user to request its refresh, therefore allowing interested and authorised end users to always perceive the security status of their services.

AA

AA subcomponent stands for Authentication and Authorisation providing a set of features regarding these two scopes. This subcomponent therefore ensures that SHIELD's available resources are only accessible to users that have the needed rights to access them. In order to

do so, this subcomponent will firstly validate the authentication and in a second phase the authorisation of each request. Authorisation encompasses the steps needed to identify the user responsible for a given request and will be responsible to check if a previously correctly identified user/token is able to access a given resource, therefore enabling that different sets of operations are accessible to different user rankings. Only if both, authentication and authorisation are assured, a specific request will be forwarded and access to Dashboard's internal components will be provided.

User Management

As previously mentioned, the access to SHIELD's internal features will be controlled in each request made to the API Service, taking into consideration the authorisation and authentication level access of the responsible user. Hence, there is a need of a subcomponent that is responsible for the management (creation, edition, deletion) of users in the scope of SHIELD. User Management subcomponent will be responsible to implement this management features ultimately providing and defining the access level of each user in the solution.

Remediation Queue

This subcomponent will be responsible for persisting and providing the suggested actions originated in the DARE's remediation subcomponent. As mentioned before, end users will be responsible for the validation/rejections of these recommendations (unless automatic remediation is selected). Moreover, if a validation is received regarding a given recommendation, this will then be forwarded to the Orchestrator for application through the Orchestrator Connector, therefore allowing SHIELD to act upon and solve a specific security thread.

Billing

Billing subcomponent is responsible for providing a set of features that enable all the SHIELD's monetisation features. Following what will be defined in SHIELD's business model, this subcomponent will allow SHIELD operators to charge for instance a "pay per use", price per vNSF, or "flat rate" whenever a new NS, vNSF or remediation action is applied.

Store Connector, Orchestrator Connector, DARE Connector

Since Dashboard component is envisioned to interact with all SHIELD's components, an abstraction layer of these connections was added in the Dashboard's architecture to minimise the implementation's dependencies across Dashboard's internal subcomponents. These three subcomponents (Store, Orchestrator and DARE connectors) will allow different SHIELD's Dashboards to interact with these three mentioned components.

3. SPECIFICATIONS AND IMPLEMENTATION

In order to drive and properly guide the development of the platform, apart from the overall architecture and design presented in the previous section, there is a need of transforming the requirements elicited in D2.1 into technical specifications for each one of the phases. As stated in D2.1, there are three types of requirements: i) platform requirements (PF), ii) non-functional requirements (NF), and iii) service requirements (SF). As i) and ii) have different implications in each phase in the data value chain (ingestion, analytics, decision and visualisation), they will be transformed into specifications per phase. For iii) we analyse them from a general point of view since they have implications in a general way into the components of the WP4. This transformation is shown in Table 1.

Table 1: General service specifications and fulfillment of service requirements.

Numbering	Title	Description
SF01	Content filtering	A security service COULD provide URL filtering based on different configurable categories (e.g. political, violence, sex, social networks, etc.) in the internet web browsing.
S_SPEC_1	The platform may be enriched with topic modelling algorithms that will provide the necessary insight for the Remediation Engine to proactively filter specific content, depending on the desired configuration provided by the user through the dashboard.	
SF02	Detect/Block access to malicious websites	A security service SHALL control access to malicious websites, such as phishing servers, malware spreading, C&C servers, etc. The user must be alerted and the access to the site could be blocked/allowed depending on the configured policy rule.
S_SPEC_2	The platform will incorporate algorithms that will detect malicious connections and will provide relevant information to the Remediation Engine and the Dashboard. The Remediation Engine will recommend specific medium level policies (MSPL – Subsection 3.3) and/or vNSFs to stop these connections.	
SF03	Security assessments	A security service COULD provide continuous vulnerability assessment on the network, hosts or applications.
S_SPEC_3	The platform may include automated security check routines that will scan the network for vulnerabilities (e.g. open ports) and remediate the threats by recommending the deployment of medium level policies and/or new vNSFs to remediate the vulnerabilities.	

SF04	L4 traffic filtering	A security service SHALL monitor traffic based on configuration rules. Traffic packets are filtering and specific traffic is either allowed, rejected or blocked based on a predefined set of rules (usually based on source IP, destination IP, destination port, etc.). Commonly called firewall.
S_SPEC_4	The platform will monitor network connections by deploying cognitive analytics and will provide feedback about the level of suspicion of each connection. The Remediation Engine will recommend medium level policies to apply to the vNSFs or directly the deployment of new vNSFs to stop these connections.	
SF05	Central log processing/SIEM	A security service COULD collect and correlate security logs from different legacy user sources and generate alerts. This service is intended to provide the user with a way to process its security logs that are not generated by a vNSF in SHIELD.
S_SPEC_5	All the logs of the system must be centralised ingested into the system and tagged as log information. This log information can be analysed in batch at best effort. Using probability of an anomaly and anomaly classification new alarms can be generated.	
SF06	Malware detection	A security service COULD detect (and optionally clean) files with malware downloaded from Internet.
S_SPEC_6	The platform will incorporate a threat classification algorithm that will correlate network anomalies to specific threats, including malware detection. Remediation Engine can recommend new vNSFs or new policies to existing vNSFs to stop connections from the infected machine and isolate it from the network. Later on, the malware may be cleaned or user will be requested to manually clean it.	
SF07	Spam protection	A security service SHALL protect against unwanted emails, based on source reputation lists and content analysis.
S_SPEC_7	The platform will incorporate a threat classification algorithm that will correlate network anomalies to specific threats, including spam activities. Network connections can be stopped by using middle policies and/or vNSFs recommended by the Remediation engine.	
SF08	DoS Protection	A security service SHALL protect against volumetric Denial of Service attacks. Detect the DoS attack and

		divert the traffic for filtering. Forwarding the good traffic flows to the destination.
S_SPEC_8	The platform module will incorporate a threat classification algorithm that will correlate network anomalies to specific threats, possibly including volumetric DoS attacks. Remediation Engine can recommend the deployment of new middle policies (e.g. specific IPSs to be isolated) and or new vNSFs (e.g. firewalls).	
SF09	Intrusion Detection/Prevention System	A security service SHALL detect attacks with a wide range of techniques such as network flow or behaviour analysis and deep packet inspection. Allow traffic flows according to IPS rules. Monitor traffic network traffic at OSI layer 7 and generate alerts for security policy violations, infections, information leakage, configuration errors and unauthorised clients.
S_SPEC_9	The platform will analyse the main types of ingested network traffic (netflow, DNS, proxy) and will combine the detected results with those provided by network monitoring vNSFs. The Remediation Engine will be able recommend to the user either to deploy new IDPS vNSFs or middle policies to the already deployed ones.	
SF10	Honeypots	A security service COULD provide a Honeypot service that simulates or impersonates specific services (e.g., Windows computer, Web server, IoT or SCADA device, etc.) in order to detect malicious behaviours in the network.
S_SPEC_10	The platform will incorporate a threat classification algorithm that will correlate network anomalies to specific threats, possibly being able to distinguish incidents that would require the deployment of a Honeypot service. The Remediation Engine will be able to recommend the deployment of a Honeypot vNSF in a specific PoP and redirect the traffic there through the deployment of L4 or L7 filtering vNSFs or the configuration of already existing ones using medium level policies.	
SF11	Sandboxing	A security service COULD provide a sandbox service for executing and analysing programs. Must provide the possibility to install different OSs.
S_SPEC_11	The platform will incorporate a threat classification algorithm that will correlate network anomalies to specific threats, possibly being able to distinguish incidents that would require the deployment of a sandbox service.	

SF12	VPN	A security service COULD provide a secure tunnel service in order to connect the branch of a client with users in Internet or other branches.
S_SPEC_12	The platform will incorporate a threat classification algorithm that will correlate network anomalies to specific threats, possibly being able to distinguish incidents that would require the deployment of a secure tunnelling service.	

Apart from the transformation of the requirements into specifications, this section also defines the specific technologies that will be used to develop every subcomponent. Although each phase exposes several technologies that can be used to fulfil the specifications (and hence the requirements), Apache Spot [1] has been selected as the main technology for the entire DARE. Apache Spot is an in-development, open-source platform for network telemetry and anomaly detection. Apache Spot provides tools to accelerate the ability to expose suspicious connections and previously unseen attacks using flow and packet analysis technologies. It also features a built-in ingestion subcomponent that is responsible for handling and transferring the raw network data into the data analytics engine. Spot is built over very mature technologies like:

- Cloudera CDH for data ingestion and storage (which uses Hadoop HDFS and Apache Hive). CDH is an Open Source platform distribution that helps to perform end-to-end Big Data workflows.
- Spark for machine learning and streaming. Apache Spark is a fast and general engine for large-scale data processing.
- ReactJS and Flux for the web components. ReactJS is a Javascript framework for building user interfaces. Flux is the application architecture that Facebook uses for building client-side web applications.
- IPython for the Spot virtualisation server. IPython is a command shell for interactive computing in multiple programming languages.
- GraphQL for query data from HDFS Parquet files. GraphQL is a query language for custom API's, and a server-side runtime for executing queries by using a user-defined type system.
- Hadoop for distributed file system. Apache Hadoop allows for the distributed processing of large data sets across clusters of computers using simple programming models.
- Hive for data storage. Apache Hive facilitates reading, writing, and managing large datasets residing in distributed storage using SQL.

Using Spot, SHIELD will be benefited in several aspects:

- Leveraging the on-going efforts of the community,
- Taking advantage of the dissemination potential offered by Spot,
- Increasing the exploitation possibilities of SHIELD,
- Collaborating with experts on cybersecurity to provide better solutions,
- Following a methodology and good practices in code development.

On the other side, Spot will be also benefited by the SHIELD developments, since they will help the project to achieve a more mature state. The choice of an integral, open-source and active cybersecurity solution like Spot goes perfectly in line with the concept an Innovation Action. It must be stressed out, at this point, that Spot is still far from suitable to be adopted "as-is" in SHIELD; in the context of SHIELD, Spot will be just used as the starting point for the DARE developments. The SHIELD team has already identified several extensions, which will be needed to the core Spot platform in order to fulfil SHIELD requirements, mostly related to functionalities such as: mitigation capabilities; near-real-time operation; classification of threats; optimised operation in an NFV environment; and enhancement of the data model to support for more types of information. These extensions are described in detail in the following sections.

In Figure 23, the subcomponents envisioned in the architecture are shown together with the technology that will be used to develop them.

It is worth to mention that Spot is not "yet another IDPS" but a platform that offers to the clients the possibility to develop their own machine learning engines. This approach allows SHIELD to focus the effort on the innovative algorithms that will be used for threat detection and mitigation.

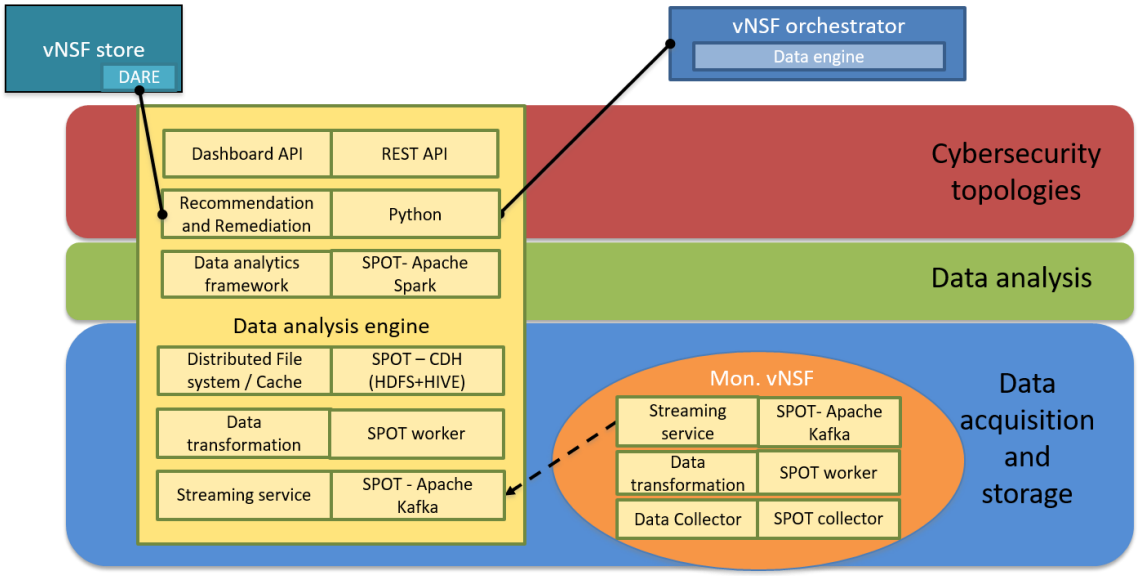

Figure 23: List of subcomponents and used technologies

3.1. Data acquisition and storage

The Data Acquisition and Storage phase involves mechanisms and methods in order to capture and transfer heterogeneous network information, from the monitoring vNSFs, into the central data analytics engine. This phase is described by two different low-level architectures, the centralised and the distributed (Section 2).

Specifications

The requirements elicited in D2.1 are here transformed into specifications from the point of view of the Data acquisition and storage phase (Table 2).

Table 2: Specifications of the Data Acquisition phase and fulfillment of requirements.

Req. number	Requirement Name	Requirement description
PF04	Security data monitoring and analytics	The platform SHALL be able to collect and analyse metrics and logs from the vNSFs in real time in order to detect security incidents
I_SPEC_01	The Data Acquisition and Storage framework utilises collector daemons to detect, capture and transfer heterogeneous network data from vNSFs into the central data analytics engine through a streaming service in real time. Worker daemons are also used, in order to transform the collected data into human readable format before storing it to a distributed file system. After being stored in the distributed file system, the data is available for further processing analysis.	
PF16	Historic reports	The platform SHALL generate reports of past incidents based on historic data.
I_SPEC_02	Report generation is supported by using the abstraction layer of the Data Acquisition and Storage framework, in order to run queries on tables, where the historic data is stored.	
NF01	Response time	The platform SHALL report the incident within a relatively short time (in the order of seconds).
I_SPEC_03	The Data Acquisition and Storage framework has a distributed architecture, which allows fast data access and processing and returns results in a relatively short time (in the order of seconds).	
NF04	Data Volume	The platform SHALL be able to handle data in the order of Terabytes.
I_SPEC_04	The Data Acquisition and Storage framework is fully scalable and can handle increasing data volume by adding more data nodes to the infrastructure.	
NF05	Impact on perceived performance	The platform SHALL be able to handle data in the order of Terabytes.
I_SPEC_05	The impact of data volume on perceived performance can be handled by the scalable architecture of Data Acquisition and Storage. When performance is degraded due to increasing data volume, adding more data nodes to the infrastructure can improve the system's performance significantly.	

Implementation details

The centralised architecture of the Data Acquisition and Storage phase (refer to Figure 15) will feature a built-in ingestion framework of the Apache Spot platform, which will be responsible

for handling the raw network data that will be transferred directly to a specific path of the data analytics engine. Ingestion framework consists of a number of edge nodes, running on Linux OS and handling the incoming network traffic. It also supports Apache Kafka [2] as a streaming platform, for handling all the real-time network data feeds.

Inside each vNSF there are daemons, called data collectors, which monitor the vNSF for new files with network data generated by it. In the case of centralised architecture, once new data files are detected, collectors capture them from local file system and publish them to Apache Kafka. Apache Kafka splits the raw network data into specific topics and smaller partitions, while creating a data pipeline for each type of data. Each pipeline sends the network data, stored by data collectors, to specific daemons, called workers. Workers are running in the background inside the data analytics engine, as part of the data transformation subcomponent. Each worker is subscribed to a specific topic and partition of Apache Kafka. It reads raw network data from the partition, decodes and translates it into comma-separated files (CSV), by using dissection tools. Once the data has been transformed, worker stores the input in HDFS with both the original and human-readable format, making it available to HIVE tables, so it can be accessible by SQL queries.

The Data Acquisition and Storage phase also supports a distributed architecture (refer to Figure 16). The main difference between the centralised and the distributed architecture is that in the centralised one, workers are running inside the data analytics engine, while in the distributed one, there is one worker per vNSF. So, in the distributed architecture, when a vNSF generates a new data file, the collector of the vNSF will detect and drive it to the corresponding worker of the vNSF. Workers will decode and translate the raw network file into a comma-separated file with specific structure, and will send it to the Apache Kafka streaming platform. Kafka will transfer the text file to the data analytics engine, where the file will be imported directly into Hive tables. Unlike the centralised architecture, which stores both the original and the human-readable formats inside the data analytics engine, only the transformed version of the files is stored in the distributed architecture.

As the Data Acquisition and Storage phase will be based on the Apache Spot platform, the heterogeneous network information that will be captured via specialised vNSFs shall be compatible with the following structure and format:

- Netflow: data files which contain network traffic as it enters or exits from a monitoring interface. By analysing netflow data, a network administrator is able to determine things such as the source and destination of traffic, class of service and the causes of congestion.
- DNS: PCAP files (packet capture) out of DNS servers. PCAP files contain network packet data, created during a live network capture. By analysing PCAP files, vital information can be retrieved regarding the monitored network and its characteristics.
- Proxy: popular proxy format logs will be supported like the bluecoat format.

Apart from the above, it is possible to take advantage of other data types produced by the vNSFs, such as possible alerts, so as to collect extra information which could be helpful in further processing by the machine learning algorithms (e.g. data obtained from the hardware attestation). Moreover, along with the raw network data, measurements could also be sent on the load and usage of each vNSF, in order to prevent machine failures or data loss.

By default, Apache Spot uses specific open-source decoders in order to transform raw network data into comma-separated files. According to the data type, a different dissection tool is being used. For flow traffic, a modified version of nfdump is used to dissect the flow packets into comma-separated files. For PCAP files (packet captures) the TShark tool is used with a combination of options so as to generate the comma-separated files. The proxy log files are parsed using Apache Streaming before being inserted into Hive tables.

Currently Apache Spot's ingestion framework supports netflows, DNS PCAP files and the bluecoat proxy log formats. In order to take advantage of other data types produced by the vNSFs, such as alerts and metrics as described earlier, significant modifications need to be made to Spot's ingestion mechanism under SHIELD's scope. New collectors need to be implemented to accommodate capture of new network data formats and Apache Kafka's configuration needs to be extended, in order to handle the new data feeds. Additionally, new workers will be required, in order to read and translate these new data feeds. These will also be the required changes in case the centralised architecture is used, which is Spot's default architecture. However, if the distributed approach is followed for the ingestion process, a redesign of current Spot architecture will be needed. Note that then, workers will be no more centralised but distributed among vNSFs, working closely with collectors and performing the feeding of transformed network data to Apache Kafka. This is quite a different view on the current Spot architecture, which is nevertheless closer to SHIELD's general distributed architecture.

Note also that a collector will be implemented inside the Trust monitor consuming the API provided by it.

Comparison with similar technologies

Both centralised and distributed architectures use Apache Kafka as streaming service. Kafka is a unified platform for handling all the real-time data feeds. It supports low latency message delivery and guarantees fault tolerance in the presence of machine failures. Kafka is distributed, partitioned, replicated and can scale easily without any down time.

Another streaming service that could be used is Apache Storm [3], a free and open-source, distributed, real-time computation system. Storm is designed to process vast amount of data in a fault-tolerant and horizontal scalable method. It is a streaming data framework that has the capability of high ingestion rates. However, Storm is mostly a computation unit, meaning it can execute all kind of manipulations on real time data in parallel. In our implementation, all the computations are taking place inside the data analytics engine and not on streaming process. In addition, Storm does not have the ability to share data with multiple systems, something that may be useful in the future, since Kafka can provide the same data to multiple storage systems (e.g. a new data analytics engine that does not use HIVE but uses PIG).

Storage is done in HIVE [4] tables over Hadoop Distributed File System (HDFS). Using HIVE, it is possible to launch SQL queries over a Distributed File System like HDFS which facilitate the work of accessing the data. A possibility is to use Spark directly, which works over HDFS without the need of any abstraction layer like HIVE. However, this will imply that access to data will be more difficult since SQL like language cannot be used. Apache PIG [5] is another abstraction layer on top of HDFS that facilitates the task of accessing the data. PIG is flow-based, instead of table-based like HIVE. Although PIG is a possibility, SPOT uses HIVE because of the convenience of developers that are used to SQL.

3.2. Data analysis phase

The DARE will leverage two different data analytics modules that will export their findings to a shared Remediation engine, in order to produce optimal results. The Cognitive Data Analytics module will be based on the Apache Spot platform, an in-development, and open-source project for network telemetry and anomaly detection. The Security Data Analysis module will implement a version of Talaia's proprietary network visibility solution based on the SecaaS architecture. The aforementioned modules will be modified and functionally enriched, with respect to the fulfilment of the SHIELD's requirements. The Remediation Engine will exploit the Data Analytics Engine's output to detect security incidents and will recommend the triggering of actions to mitigate the threats, utilizing open-source technologies and implementing the security policies defined in the next section. Next follows a specifications and implementation subsection for each module involved in the DARE.

Specifications

The requirements elicited in D2.1 are here transformed into specifications from the point of view of the Data analysis phase (Table 3).

Table 3: Specifications of the Data Analysis phase and fulfillment of requirements.

Req. number	Requirement Name	Requirement description
PF04	Security data monitoring and analytics	The platform SHALL be able to collect and analyse metrics and logs from the vNSFs in real time in order to detect security incidents
A_SPEC_01	The platform will include worker nodes that will utilise the ingested network traffic to implement batch (historic) and streaming (real-time) machine-learning algorithms in order to detect security threats.	
PF08	PF08. Platform expandability	The SHIELD platform offers well-documented APIs and interfaces as well as SDKs and guidelines so that third parties can easily develop new security functions and services.
A_SPEC_02	The developed platform will be developed using open-source and each module will be isolated through APIs. Several analytics engines can be used. Specifically, two analytic engines will be developed. One open source and based on cognitive principles and another one privative and based on pattern discovery techniques.	
PF13	Mitigation	The platform SHALL be able to trigger, in the case of an event, proper actions in order to mitigate the threat.
A_SPEC_04	The platform will provide information of the detected threats to the Remediation engine, in order to initiate the mitigation procedure.	

PF16	Historic reports	The platform SHALL generate reports of past incidents based on historic data.
A_SPEC_05	The detected threats from this phase will be stored in the distributed file system and will be available for processing and reporting.	
PF17	Interoperability	All interfaces of the vNSFO, the vNSFs and the DARE are publicly documented and compliant to open standards to the maximum possible extent.
A_SPEC_06	The Cognitive Data Analytics module is based on open-source state-of-the-art technologies that comply to open standards and industry's best practices. The proprietary engine is used as a “black box” that will implement the given APIs in order to connect it to the platform.	
PF18	Service composition	The platform SHALL be able to compose security services by combining one or more of the available vNSFs.
A_SPEC_07	The platform will communicate with a number of vNSF collectors in order to ingest network traffic required for performing security analytics	
NF01	Response time	The platform SHALL report the incident within a relatively short time (in the order of seconds).
A_SPEC_08	The platform will incorporate state-of-the-art streaming processing technologies to process the ingested data.	
NF05	Impact on perceived performance	When network traffic is proxied or analysed, the user experience SHALL not be degraded.
A_SPEC_09	The platform will be based on a distributed computing framework that will feature scalable storage and processing, load-balancing and resource management functionalities.	

Cognitive Data Analytics module

The Cognitive Data Analytics module infrastructure consists of a cluster of nodes running on open-source OS and virtual machines. Each node will perform a number of operations and will be orchestrated by cluster and resource management technologies, specifically designed for big data applications. The specifications of the Cognitive Data Analytics module include:

- A cache system for real time analytics.
- Unified services for orchestration, operations and resource management.
- Machine Learning modules that provide scalable ML algorithms for network traffic filtering, as part of a cluster-computing framework.

- An Operational Analytics module that allows for the implementation of whitelisting and filtering techniques that will help reduce false positives and offer remediation recommendations to the users.

The module will consist of a number of nodes (physical or virtual machines), each of them performing a designated task. The worker nodes receive the ingested network traffic from the Data Acquisition Phase and are responsible for the operation of the machine learning entity. The Operational Analytics nodes execute the filtering and whitelisting functionalities and are intended to operate as the final editing step, before pushing the detection results to the Remediation Engine.

Implementation details

The cognitive Data Analytics module performs anomaly detection analytics, following the Spot architecture, by deploying a collection of state-of-the-art technologies (Hadoop [6], Spark [7], Kafka [8] etc.) in the form of an integrated ecosystem, the Cloudera Distribution for Hadoop [9]. CDH is an Apache-licensed open-source framework that delivers the core elements of scalable storage and distributed computing, along with a Web-based UI and enterprise capabilities and is considered the most popular distribution for Apache Hadoop and related projects. Spot utilises CDH as a general dependency package for the development of its three main parts: ingestion (Subsection 3.1), machine-learning and operational analytics.

- Machine Learning: The machine learning entity is responsible for the detection of anomalies in network traffic and the prevention or mitigation of potential threats. Apache Spot already contains routines for performing anomaly detection on netflow, DNS and proxy logs and there is the intention to develop additional algorithms that will handle security event data and metrics gathered from the vNSFs. These routines consume a collection of network events and produce a list of the events that are considered to be the least probable, these being considered the most suspicious. The statistical model that is currently used for discovering abstract topics of these events and ultimately discovering normal and abnormal behaviour is a topic modelling algorithm called Latent Dirichlet Allocation [10]. LDA is a generative probabilistic model used for discrete data that is applied to network traffic by converting network log entries into words through aggregation and discretisation, in order to discover hidden semantic structures. Spot executes LDA routines using a Scala Spark implementation from MLlib, Apache Spark's scalable machine learning library. It should be noted that Spot's current capabilities do not include any anomaly classification algorithms that would interpret the detected outliers as specific threats/attacks, thus such an algorithm will be originally developed to meet this requirement. The module will exploit Spot's existing batch processing capabilities, coupled with the development of streaming analytics functionalities currently missing from Spot, in order to achieve real-time (or near real-time) visibility for threat detection.
- Operational Analytics: The operational analytics entity will exploit Spot's built-in results' editing capabilities, as well any additional features that will occur as a result of the developed machine-learning features. Spot uses the Jupyter/iPython notebook, a server-client application that allows editing and running notebook documents via a web browser, in order to apply filtering and whitelisting services thus providing a more accurate view of the overall anomaly detection procedure, by reducing false-positives.

Comparison with similar technologies

The Cognitive Data Analytics module was initially planned to be developed by leveraging machine learning techniques (e.g. Naive Bayes classification, Support Vector Machine) to analyse events and network data. This would include the selection, configuration and deployment of a number of state-of-the-art frameworks for big data analysis (e.g. Apache Mahout, Scala frameworks). The Apache Spot framework can be considered a superset of the above frameworks, as it features a combination of distributed computing (Hadoop), data lake management (Hive) and machine learning (Spark) frameworks along with auxiliary services to ingest, analyse and present the detected anomalies in network traffic. It being maintained by a large community is a strong indication that Spot will continue to evolve its cybersecurity capabilities, while the open-source nature of the project will allow for contributions regarding all of its main components.

A similar framework that could be deployed for network analytics within the scope of SHIELD is Apache Metron/OpenSOC [11]. Metron is a cybersecurity application framework that provides the ability to ingest, process and store diverse security data feeds at scale in order to detect cyber anomalies. Metron and Spot have similar objectives and approaches from an ingestion, storage and user interface perspective. The key differences relate to the Open Data Model that is utilised only by Spot and the fact that Spot leans towards a machine-learning approach to provide results, while Metron is currently more focused on traditional deterministic (rules/signatures/patterns) analytics.

Other open source/commercial products not very close to the network traffic analysis that try to simplify the cognitive analysis closing the algorithms available or lock to their own sources (Splunk, Machine Learnings, ToolKit app) or on the contrary just offer wide open Machine Learning tools (Google TensorFlow, Amazon ML) where the client must setup the dataset injection model and create their algorithm from scratch. Apart from the fact that created trained algorithms code is not exportable, these products are not specialised in network security, such as Apache Spot.

Finally, Spot uses Spark instead of Hadoop. Spark is an extension of Hadoop in the sense that for batch processing it uses Hadoop, but it adds the possibility to work with cache and streaming. This will be absolutely necessary for processing real-time traffic.

Security Analysis Module

The Security Analysis module is derived from the security module of Talaia's commercial product and will be integrated into the SHIELD platform as a stand-alone component. That module will be extended and integrated within the SHIELD framework. To this end, and similarly to the Cognitive Data Analysis module, the specification of the Security Analysis module includes a functionalities for real-time detection and classifications of network performance anomalies with a low false positive ratio.

The different entities inside this module can run in a distributed infrastructure or in a stand-alone node (physical or virtual machine) depending on the scenario requirements. The operation will start with the reception of the data by the Data Ingestion framework that will transform and adapt the data so it can be understood by the Security Analysis module. Then, the data is stored in a distributed storage system in order to assure no data is lost while the Security Analysis module is processing them.

Implementation details

From the implementation point of view, the Security Analysis module is a stand-alone black box integrated into the SHIELD framework. As previously mentioned, the Security Analysis module is derived from Talaia's commercial product. Because of this, some specific details related to the exact implementation cannot be published.

The original Security Analysis module only uses Netflow-derived data to perform the detection and classification of network performance anomalies (e.g., DDoS, network scans). In this project, this module will be extended to use the data provided by the Apache Spot framework (e.g., DNS, proxy logs) and by the monitoring vNSFs developed within the SHIELD project. All the data available will then be transformed so the different entities in the Security Analysis module can take advantage of the new information. As a result, the data mining and machine learning techniques used in the four entities of the Security Analysis module will enrich and improve the detection and classification of network anomalies. In addition, the new information provided will also allow the detection and classification of new anomalies not detected in the original version.

Once new anomalies are detected and classified, the Security Analysis module will adapt its output (e.g., anomaly detected, traffic involved in the anomaly) so the Dashboard can represent it and the Remediation Engine can actuate on it.

Comparison with similar technologies

The technologies used in the Security Analysis module consist of a set of cutting-edge data mining and machine learning techniques from the literature combined with proprietary methods.
On the one hand, as part of a commercial product, similar solutions can be found in Talaia's competitors. However, similarly to Talaia, competitors are always very reticent to share the details of part of its core business. Among the different competitors, we can highlight Kentik [12] and DeepField/Nokia [13] because both solutions are software-based solutions commercialised from the cloud and that are currently being adapted to the NFV paradigm. In addition, similarly to the Security Analysis module, these solutions base their classification and detection of anomalies in data mining and machine learning techniques enriched with proprietary methods. There are also hardware-based solutions as the one provided by Arbor Networks [14] that are able to detect network anomalies by using deep packet inspection techniques.
On the other hand, the open-source Apache Spot framework used in this project and the Metron framework are also similar solutions.

3.3. Cybersecurity topologies phase

The policy engine leverages the policy specification and optimisation techniques developed in the scope of the EC-funded project SECURED [15]. The SECURED project aimed at the definition of an abstract configuration for each security capability, which could be consistently transformed into specific settings by the actual implementation of the security control.

Specifications

The requirements elicited in D2.1 are here transformed into specifications from the point of view of the Cybersecurity topologies phase (Table 4).

Table 4: Specifications of the Cybersecurity topologies phase and fulfillment of requirements.

Req. number	Requirement Name	Requirement description
PF02	vNSF lifecycle management	The platform SHALL be able to manage the full lifecycle of vNSFs (on boarding, instantiation, chaining, configuration, monitoring and termination).
T_SPEC_01	The Recommendation and Remediation subcomponent will be in charge of defining configurations, using a high-level, application-independent syntax, for each vNSF. To do so, it will provide pre-defined recipes to address each supported threat. These recipes will define the protection requirements to be implemented by the vNSFs in order to address the network security threat. The recipes will be stored in a database and an API will be provided to interact with it. Starting from the protection requirements, a set of security capabilities will be derived. Each vNSF will have to support one or more capabilities, allowing this subcomponent to select the optimal set of vNSFs (the Forwarding Graph). The Remediation and Recommendation subcomponent will then translate each capability into a high-level policy for configuring the vNSF and will provide the suggested mitigation to the final user via Dashboard, by interacting with the Dashboard API subcomponent.	
PF06	Ability to offer different management roles to several users	The platform SHALL provide domain management with accessibility to the resources of a domain by different users. The admin of a domain has to be able to create management users with different roles.
T_SPEC_02	The Recommendation and Remediation subcomponent will provide mitigation actions to be applied in the user's domain. No information will be gathered regarding other tenants during this operation.	
PF08	Platform expandability	The platform SHALL be easily extended to support new security services.
T_SPEC_03	The Recommendation and Remediation subcomponent will be able to interact with the vNSF Store in order to select the best set of vNSFs to address a security threat with a generic pull interface. In this way, the subcomponent will be able to choose among all the different and evolving implementations for the NSs.	

PF13	Mitigation	The platform SHALL be able to trigger, in the case of an event, proper actions in order to mitigate the threat.
T_SPEC_04	The Cybersecurity Topologies phase of the DARE will implement the complete workflow to derive a mitigation action, consisting in a topology of vNSFs and their configuration (expressed in an application-independent syntax) starting from an occurring threat.	

Implementation details

The policy engine to be developed in the Cybersecurity Topologies phase will leverage the functionalities provided by several modules developed within the SECURED project, but it will aim to extend their capabilities and adapt the workflow to what is expected in the DARE.

The different layers of policy abstraction, as expressed in the specifications for this subcomponent, will be mapped to different languages, namely:

- High-level Security Policy Language (HSPL), suitable for expressing the general protection requirements without specifying configuration rules;
- Medium-level Security Policy Language (MSPL), suitable for expressing configuration rules in an application-independent syntax.

The policy abstraction layers will be relevant for the SHIELD recommendation and remediation engine, as they will be used to describe the mitigation action starting from the general protection requirements to the configuration rules to be applied by each vNSF. The first will be utilised by the High-level Policy Recipe Provider to describe one or more rules that compose the recipe. The latter will be used to describe configuration rules for each vNSF, whose functionalities will be mapped on the security capabilities. It is to be noted that each vNSF in the catalogue will have to provide this information embedded in its metadata, in order to allow the recommendation and remediation engine to choose the best set of vNSFs for a specific threat.

The SECURED project also designed and developed different transformation tools to ease the configuration of security functions, namely:

- H2M Service for HSPL to MSPL transformation;
- M2L Service for MSPL to low-level configuration transformation.

Hence the transformation from MSPL to low-level configuration will not be part of the recommendation and remediation engine, no integration of the M2L Service is needed in this component.
The recommendation and remediation engine will leverage the functionalities offered by the H2M Service to provide an optimal set of vNSFs with proper MSPL configuration starting from the set of HSPL rules in input.

More specifically, the already available module will be based on a Java 1.7 project, relying on different open source frameworks, such as:

- Drools [16], a Rule Based System (RBS) based around the Rete algorithm which allows the specifications of conditions (using a query language) that trigger specific actions (written as Java code);
- MOEA [17], a framework which is commonly used for multi-objective optimisation.

The already developed modules provide APIs to allow an extension of the architecture, as it is envisioned in SHIELD. The two modules that will be implemented from scratch, namely the High-level Policy Recipe Provider and the Dashboard API subcomponent, will follow the best practices in development, testing and documentation and will leverage the functionalities offered by several frameworks, such as Swagger [18] for the API provisioning and different technologies for No-SQL database (e.g. MongoDB [19], Redis [20], Apache Cassandra [21]), required to store the HSPL-based recipes in a key-value structure (where the key would be the threat "signature"). A REST API with Create, Read, Update, and Delete CRUD capabilities will be provided in order to let a system administrator interact with the recipes' database.

Dashboard API operations

The following table (Table 5) describes the envisioned operations of the Dashboard API.

Table 5: Operations of the Dashboard API.

Operation	Arguments	Description
post_mitigation_recommendations	-	Send the list of MSPL policies to be applied
post_mitigation_location	-	Send the Forwarding Graph (topology of vNSFs) and the locations where to deploy
post_threat		Sends the detected and classified threat
get_traffic	threat	Gets all the traffic related to a specific threat

Comparison with similar technologies

The configuration of vNSFs in an orchestrated, distributed environment has received great attention in literature, but there are not currently open-source, widespread technologies that provide this capability because of the lack of formal representations for the type of data needed. The policy specification engine, as originally designed and developed in SECURED, has been considered as a fitting solution for the DARE's recommendation and remediation subcomponent, as it provides both the two-level abstraction for policies and the functionalities to identify, optimise and configure vNSFs running in a SDN/NFV environment. In addition, the MSPL language is defined as a meta-model, which may be extended to support different security functions, other than the ones defined in SECURED. Other solutions, which will be briefly introduced in this section, are either too specific for a security function (e.g. packet filter) or don't provide a convenient way of describing configurations in an application-independent syntax. Proprietary solutions, if any available, are not described in this section as they would

provide translation mechanisms for vendor-specific network security functions. Firmato [22] is a proposal for translating high-level security requirements into packet filter configurations, based on a entity-relationship model to represent the knowledge base (e.g. the protection requirements and the topology of the network. Its applicability to large and heterogeneous networks, which may be typical for NFV environments, has not been proven, as the technology has been applied only to a network with a single border firewall. FACE [23] is another model for configuring a firewall in an NFV environment. It takes as input the topology and a global high-level policy, and outputs the packet filter rules. MIRAGE [24] is a tool for analysis and deployment of security policies, which is not limited to supporting packet filter's rules, but also configurations for VPN gateways and IDSs. The SECURED's solution is considered to be more generic, as it adopts an extensible capability-based language, and it is also supported by an open-source implementation that has been already tested in a NFV scenario, close to the one envisioned in SHIELD.

3.4. Dashboard

Based on the requirements elicitation available in D2.1 as well as on the specification of the Dashboard component previously provided in the current document, this section will explain the requirements currently addressed by the Dashboard component as well as the implementation details currently envisioned for this component (Table 6).

Specifications

The requirements elicited in D2.1 are here transformed into specifications from the point of view of the Dashboard (Table 6).

Table 6: Specifications of the Dashboard component and fulfillment of requirements.

Req. number	Requirement Name	Requirement description
PF03	vNSF status management	The operator SHALL be able to control the lifecycle via a graphical user interface. The vNSF lifecycle should support events like DEPLOY, START, STOP, MODIFY, DELETE.
D_SPEC_01	Dashboard has connectivity with Orchestrator component (Orchestrator connector subcomponent) allowing lifecycle features to be provided to the end user through the Graphical User Interface.	
PF05	Analytics visualisation	The operator SHALL be able to see the analytics visualised in e.g. a dashboard.
D_SPEC_02	DARE connector subcomponent will allow GUI to expose monitoring information persisted in this component. Furthermore, Remediation Queue	

	subcomponent will stage recommendation sent by the DARE waiting for acceptance/rejection from the end user.	
PF06	Ability to offer different management roles to several users	The platform SHALL provide domain management with accessibility to the resources of a domain by different users. The admin of a domain has to be able to create management users with different roles.
D_SPEC_03	User management and AA subcomponents will provide Authentication, Authorisation and user management features allowing creating and using different roles with different sets of permissions. SHIELD's features and information will be filtered based on the role and permissions of each authenticated user.	
PF07	Service elasticity	The platform COULD provide the mechanism to allow scalability of the vNSFs.
D_SPEC_04	Dashboard will allow triggering this mechanism through its interface, using the Orchestrator connector to ensure this feature.	
PF09	Access control	The platform SHALL provide a secure environment. Authentication mechanisms that should control the access and restrict access only to authenticated users.
D_SPEC_05	User management and AA subcomponents will provide authentication, authorisation and user management features. Only authenticated users will have access to SHIELD's internal features and information.	
PF12	Log Sharing	Sharing logs with a third entity SHALL be allowed. The granularity of the data provided by the logs depends on the severity and type of each attack.
D_SPEC_06	The Dashboard will provide access to APIs that will allow the sharing of logs between the DARE and other entities.	
PF13	Mitigation	The platform SHALL be able to trigger, in the case of an event, proper actions in order to mitigate the threat.
D_SPEC_07	The Dashboard will allow the deployment of vNSFs and policies via the GUI.	

PF14	Multi-tenancy	The platform SHALL accommodate multiple users, with isolated services and secured access to analytics
D_SPEC_08	User management and AA subcomponents will provide authentication, authorisation and user management in a multi-tenant environment.	
PF15	Service store	The store SHALL allow selecting security services from the catalogue.
D_SPEC_09	Store connector subcomponent will allow GUI to expose Store's security service catalogue features.	
PF16	Historic reports	The platform SHALL generate reports of past incidents based on historic data.
D_SPEC_10	Historic reports will be available in the dashboard using DARE connector to fetch this information.	
PF17	Interoperability	The platform SHALL expose openly-defined APIs for information exchange with third parties.
D_SPEC_11	Dashboard will provide an open API allowing third party applications to use it.	
PF20	Billing framework	The platform SHALL implement a billing framework for the use of the security services. The clients should be able to access to the functionalities defined by their payment modality.
D_SPEC_12	Billing features will be assured by Dashboard's Billing subcomponent enabling SHIELD's monetisation features.	

Implementation details

The Dashboard contains a set of subcomponents that together allow providing all the features envisioned for the component. Regarding its Graphical User Interface subcomponent, it will be based on web technologies allowing end users to access it using a common web browser. Hence, standard technologies will be used such as Hyper Text Markup Language (HTML), Javascript and Cascading Style Sheets (CSS). The use of web frameworks based on these technologies such as AngularJS is also envisioned allowing to boost development efficiency, productivity, data handling and maintenance. Regarding visual styling, technologies such as Sass will be incorporated as an extension to CSS enabling new manipulation methods as well as an efficient code structuring. The GUI subcomponent will leverage results achieved in previous and running EC-funded projects namely T-Nova and SELFNET. The features provided by the

Graphical User Interface of each one of these projects will be merged and extended taking advantage of the result of all discussions and conclusions that took place in each project. More precisely, the web interfaces associated with the vNSF catalogue, store, permission management and vNSF orchestration will be based on the conjunct result provided by these two projects. Regarding network topology visualisation, SHIELD's GUI will leverage the work done in both SELFNET and SONATA in an attempt of providing an appealing and intuitive interface for this feature. While SELFNET focused on the physical and virtual connectivity of the virtual machines instantiated across the Network Infrastructure, SONATA implemented a hierarchical visualisation on the resources associated with each Network Service. For the implementation of this visualisation, the use of Scalable Vector Graphics (SVG) is envisioned with the support of D3js framework providing a set of relevant tools allowing its manipulation. The use of both D3js and SVG is enforced by the fact that both SELFNET and SONATA use this combination to implement its topology visualisation features.

The interaction between GUI and API Service is envisioned to be implemented using REST technology. As a consequence, a REST client will be developed in the GUI subcomponent allowing the connection to a REST Service to be developed and provided by API Service. In regard to the real time push of information coming from API Service to the GUI, websockets technology is envisioned allowing the seamless implementation of the envisioned workflow.

As previously mentioned, API Service will be based on REST technology providing a set of endpoints allowing the use of features provided by SHIELD's internal components. These endpoints will interact with a specific Dashboard's internal subcomponent (User Management, Remediation Queue, Billing, Store Connector, Orchestrator Connector, and DARE Connector) responsible for sanitizing data as well as for interacting with the correct connector. Regarding the connector subcomponents, their goal is to abstract the interaction with a given SHIELD component so the use of technologies like real time messaging, advanced message queuing, REST or SOAP technologies will depend on the interfaces and workflows provided by it.

Comparison with similar technologies

Security Dashboards are currently available both in private and open source initiatives. These dashboards are often based in web browser technologies however desktop or mobile applications are also used. Looking at the features commonly provided by both private and open source initiatives the main goal seems to be allowing end users to quickly understand the status of the controlled environment by flagging the last incidents reported as well as the last remediations suggested/implemented. SHIELD's dashboard aims to align with this approach providing an interface that allows authorised users to quickly check the last events detected by the platform as well as remediation action to mitigate each one of them. Regarding open source initiatives, the most prominent solution is Apache Spot, which is providing a security dashboard that enables the quick visualisation of the last threads detected by its framework. Looking into the private sector, one of the most relevant solutions is the Security dashboard of IBM providing a view on the security health, by showing the top 10 attacks/intruders/victims as well as the blocked actions performed by the security tool. SHIELD aims to extend the features provided by these dashboards, allowing not only the end users to have an intuitive and appealing interface for visualizing the last events detected in the network but also to orchestrate its environment as well as control the available mitigation actions through a NS and vNSF catalogue.

4. CONCLUSIONS

Using the requirements and the high-level design of D2.1, T4.1 has performed a detailed analysis and has produced a detailed design and architecture, where all the WP4 subcomponents are explained, their workflows are shown, the communication between them is defined and a study about the implementation technologies is exposed. The results are the present report on the low-level architecture and design of the DARE and the Dashboard (the WP4 components), as well as a transformation from the requirements to the technical specifications, and a choice of the technologies used.

We have realised that due to the work that has been done during these first months of the project, several subcomponents have been redefined, mainly because of changes in the state-of-the-art from the moment when the proposal was submitted and because of the more extensive knowledge gained within the consortium. However, none of these changes introduce a major shift from the overall technical approach of the project, as laid out in the DoA.

Although we expect to acquire new knowledge and get more insights during the development phase (starting from this deliverable), the consortium does not envision major adjustments during the updates of the design deliverables (D2.2 - M17, D3.2-M19 and D4.2-M19). Note that the work exposed in this deliverable perfectly separates the Architecture and Design (blocks and workflows), the specifications (requirements from the technical point of view) and the implementation (the technologies used). This separation isolates the subcomponents in a way that the implications of a change in any of these aspects (architecture, design, specifications and implementation) will be minimised.

As an Innovation Action, SHIELD's vision is to leverage state-of-the-art techniques and try not to reinvent the wheel. To this end, SHIELD has studied the most mature open source technologies and has concluded that Apache Spot will be the main solution to be reused and improved to build the DARE. Apache Spot has some of the most important functionalities needed by the DARE (ingestion, data treatment, extensible analytic framework and a dashboard) however, it is still missing some relevant aspects needed by SHIELD. Firstly, Apache Spot has been built to be a batch solution and although streaming technologies have been considered, the collection of data is completely centralised (workers read a folder for new files). This is not enough for SHIELD since one of the envisioned functionalities is the capacity to process vNSF logs and alerts in real-time. Secondly, the platform offers an anomaly detection algorithm based on probabilities of events however, neither classification of threats is being done nor real-time processing. Moreover, since Spot is completely lacking threat mitigation and recommendation functionalities, these will be originally developed, so that the capabilities of the DARE are in accordance with what was initially envisioned in the DoA. Finally, as SHIELD has to integrate information from multiple sources in the Dashboard (Store, vNSF Orchestrator, recommendations, and results from Security engine and results from Cognitive engine), it will not use the dashboard provided by Spot, but will directly use the API provided by the analytics framework.

With all these aspects in mind, we conclude the first iteration of the project design and we enter into the first iteration of the development phase, having drafted a clear technical roadmap for the months to come.

REFERENCES

[1] Apache Spot - http://spot.incubator.apache.org/ (Accessed May 2017)

[2] Apache Kafka - https://kafka.apache.org/ (Accessed May 2017)

[3] Apache Storm - https://storm.apache.org/ (Accessed May 2017)

[4] Apache Hive - https://hive.apache.org/ (Accessed May 2017)

[5] Apache Pig - https://pig.apache.org/ (Accessed May 2017)

[6] Apache Hadoop - https://hadoop.apache.org/ (Accessed May 2017)

[7] Apache Spark - https://spark.apache.org/ (Accessed May 2017)

[8] Apache Kafka - https://kafka.apache.org/ (Accessed May 2017)

[9] Cloudera Distribution for Hadoop - http://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html (Accessed May 2017)

[10] Latent Dirichlet Allocation (Blei et al, 2003) - http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf (Accessed May 2017)

[11] Apache Metron - https://metron.apache.org/ (Accessed May 2017)

[12] Kentik - https://www.kentik.com/ (Accessed May 2017)

[13] Deepfield - https://deepfield.com/ (Accessed May 2017)

[14] Arbor - http://es.arbornetworks.com/ (Accessed May 2017)

[15] The SECURED project - http://www.secured-fp7.eu/ (Accessed May 2017)

[16] The Drools project - https://www.drools.org/ (Accessed May 2017)

[17] MOEA framework - http://moeaframework.org/ (Accessed May 2017)

[18] Swagger Framework - http://swagger.io/ (Accessed May 2017)

[19] MongoDB - https://www.mongodb.com/ (Accessed May 2017)

[20] Redis - https://redis.io/ (Accessed May 2017)

[21] Apache Cassandra - https://cassandra.apache.org/ (Accessed May 2017)

[22] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, “Firmato: A novel firewall management toolkit,” ACM Transactions on Computer Systems, vol. 22, no. 4, pp. 381–420, November 2004.

[23] P. Verma and A. Prakash, “FACE: A Firewall Analysis and Configuration Engine,” in SAINT05: Symposium on Applications and the Internet, Trento, Italy, February 2005, pp. 74–81.

[24] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda, “Mirage: A management tool for the analysis and deployment of network security policies,” Data Privacy Management and Autonomous Spontaneous Security, vol. 6514, pp. 203–215, 2011

LIST OF ACRONYMS

Acronym	Meaning
DARE	Data Analysis and Remediation Engine
DoS	Denial of Service
DoA	Description of the Action
NS	Network Service
SecaaS	Security-as-a-Service
vNSF	Virtual Network Security Function
PoP	Point of Presence