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

[H2020 - Grant Agreement No. 700199]

Deliverable D4.3

Information-driven engine ready for experiments

Editor D. Papadopoulos (INFILI)

Contributors G. Gardikis, A. Kapodistria, G. Kolonias, S. Pantazis (Space Hellas), C. Fernandez, B. Gaston (i2Cat), A. Litke, N. Papadakis (INFILI), G. Dimopoulos (Talaia), M. De Benedictis (POLITO), A. Pastor (TID), R. Preto (Ubiwhere), O. Segou, C. Xilouris (ORION).

Version 1.0

Date November 29th , 2018

Distribution PUBLIC (PU)

Executive Summary

This deliverable documents the conclusions of the technical development work of SHIELD related to WP4. It covers the final architectural design, the development and deployment of the core components involved in this work package, namely the information-driven Data Analysis and Remediation Engine (DARE) and the Security Dashboard.

In terms of design, this deliverable provides the final functional layout of the DARE platform and of the Security Dashboard, along with any functional or architectural updates that have been introduced since D4.2. These updates -which involve all WP4 components- mainly include the addition of new algorithmic implementations, as well as of extensions and enhancements of the existing ones, and are sufficiently documented within the context of D4.3. Specifically, details of new or updated functionalities are provided for the Data Acquisition module, the Cognitive and Security Data Analytics Modules, the Cybersecurity Topologies and the Security Dashboard components.

An aggregated version of the final mapping of the requirements (Platform Functional, Non-Functional, Service Functional and Ethical & Regulatory Compliance) is also collected on this deliverable.

Finally, instructions are supplied for the set-up of the scalable storage and computing environment, as well as for the installation and deployment of the individual DARE components and of the Security Dashboard.

Table of Contents

1. INTRODUCTION 5
1.1. SHIELD project overview 5
1.2. Scope of this document 5
1.3. Organisation of this document 6
2. FINAL FUNCTIONAL LAYOUT 7
2.1. DARE architecture 8
3. UPDATES SINCE D4.2 10
3.1. Data acquisition and storage 10
3.2. Data analysis phase 11
3.2.1. Cognitive Data Analysis Module 11
3.2.2. Security Data Analysis Module 17
3.3. Cybersecurity topologies phase 17
3.4. Dashboard 19
4. REQUIREMENTS MAPPING 20
5. ENVIRONMENT SETUP GUIDE 22
5.1. CDH framework installation and configuration 22
6. INSTALLATION GUIDE 25
6.1. Open Source status 25
6.2. Apache Spot 25
6.3. Distributed Collector and Streaming Worker 27
6.4. Cognitive DA module 28
6.4.1. Anomaly Detection DL 28
6.4.2. Anomaly Detection ML 28
6.4.3. Threat Classification ML 29
6.4.4. Threat Classification DL 29
6.5. Security DA module 30
6.6. Recommendation and Remediation Engine 30
6.7. Dashboard and APIs 31
7. CONCLUSIONS 32
7.1. Present status 32
7.2. Future work 32
8. REFERENCES 33

LIST OF FIGURES 35
LIST OF TABLES 36
LIST OF ACRONYMS 37
ANNEX A. REGULATORY COMPLIANCE 39

1. INTRODUCTION

1.1. SHIELD project overview

The SHIELD project aims to deliver virtualised security services to protect the flow of data across the infrastructure controlled by the SHIELD platform and also to ensure that such infrastructures are properly attested and secured.

To do so, several technologies are combined. The Network Function Virtualisation (NFV) and Software-Defined Networking (SDN) environments allow the deployment of virtualised services (NSs); whilst the Trusted Computing (TC) provides means to attest and verify whether the infrastructure is secured or trusted. Finally, the Big Data (BD) Analytics and Trusted Computing (TC) provide remediation suggestions to inactivate or destroy threats in any virtualised or physical node running on the SHIELD-managed infrastructure.

1.2. Scope of this document

The technical work of WP4 ("Usable information-driven engine") is dedicated to the creation of a reliable and scalable cybersecurity framework, which offers Security-as-a-Service by exploiting the insights of network data analysis. This involves: (a) the development of acquisition and storage capabilities for data associated with cyberattacks; (b) the development of data analytics capabilities for anomaly detection and threat classification, by employing machine learning and deep learning techniques; (c) the development of adaptive cyberattacks mitigation capabilities, providing remediation policies to be deployed in specific places of the network; and (d) the development of a graphical user interface with operations and management capabilities, making SHIELD a cybersecurity.

This document (D4.3, "Information-driven engine ready for experiments") provides the last status regarding the DARE and Security Dashboard components. During M20-M27, SHIELD has continued the deployment, adjustment and configuration efforts on streaming data acquisition and storage, novel anomaly detection and classification models for near-real-time analytics and updated recommendation and remediation recipes for the DARE. Regarding the Dashboard, updates include integration with the vNSF Orchestrator, support of multi-user NS lifecycle management, as well as additional GUI features such as real-time notifications, NS inventory and catalogue views and visualizations for vulnerabilities attestation

D4.3 inherits and extends the following deliverables:

- D2.2 "Updated requirements, KPIs, design and architecture", which gives the final, updated version of D2.1.
- D4.2 "Updated specifications, design and architecture for the usable information-driven engine", which contains the second iteration of design and specifications for the DARE platform and for the Security Dashboard, which was initially provided in D4.1.

1.3. Organisation of this document

This document is organised as follows:

- Section 1 (present section) serves as a basic introduction to this document and its scope;
- Section 2 provides the overview of the final functional layout of the DARE platform;
- Section 3 details the updates at the different levels (architecture and design, development) per WP4 component since D4.2;
- Section 4 provides a summarized view of the requirements per WP4 component, along with explanations regarding the satisfaction of the agreed requirements.
- Section 5 lists the instructions to perform the setup of the environment that supports the operations of the DARE platform and of the Security Dashboard;
- Section 6 includes the guide for the installation of the WP4-related components;
- Section 7 concludes the document, summarizing the WP4 work during the project and suggesting next steps for improvements;
- Annex A provides the Ethical Regulatory Compliance requirements for the WP4 components.

2. FINAL FUNCTIONAL LAYOUT

Based on the SHIELD use cases and the requirements highlighted in Deliverable D2.2 [1], the designed high-level architecture for the SHIELD platform is articulated around six different components, illustrated in Figure 1 (vNSFs, Trust monitor, vNSF Orchestrator, vNSF Store, Security overview dashboard and DARE). The low detail architecture and the design of the components corresponding to WP4 where defined in D4.1 [2] and D4.2 [3]. These components (DARE and Dashboard) are the ones analysed in this deliverable where we expose their final state and their functionalities. The flows of data between these components is shown in Figure 2.

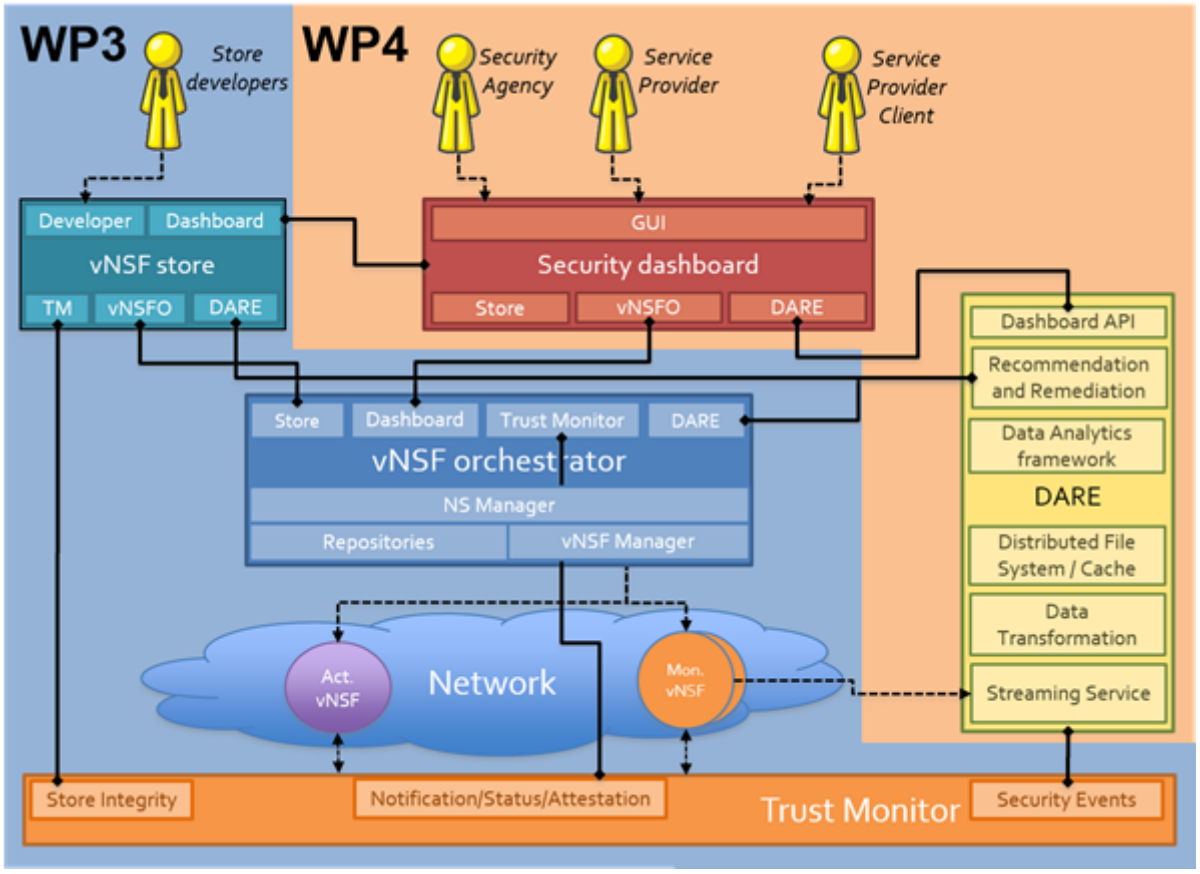

Figure 1: The SHIELD architecture

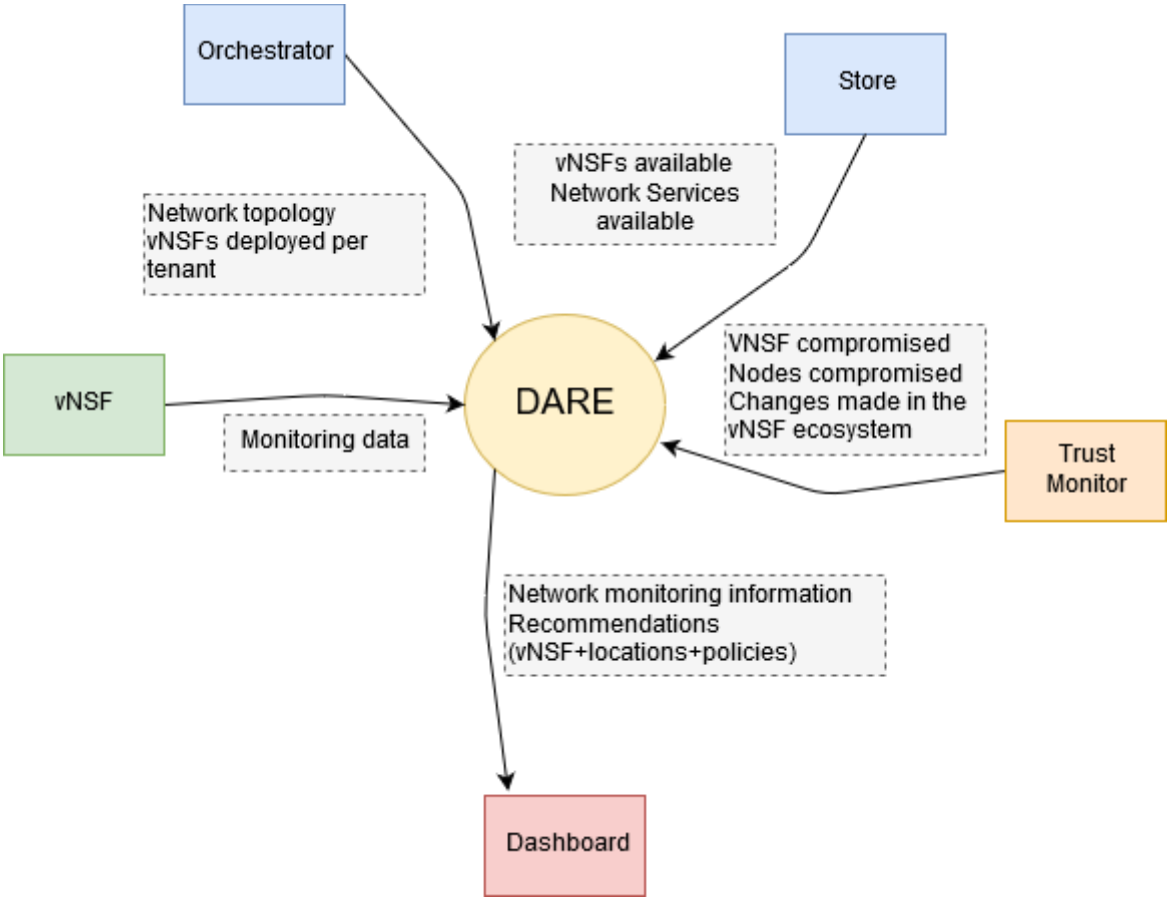

Figure 2: Data flow diagram of the DARE

2.1. DARE architecture

The DARE is 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 3:

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

Following, we describe these subcomponents:

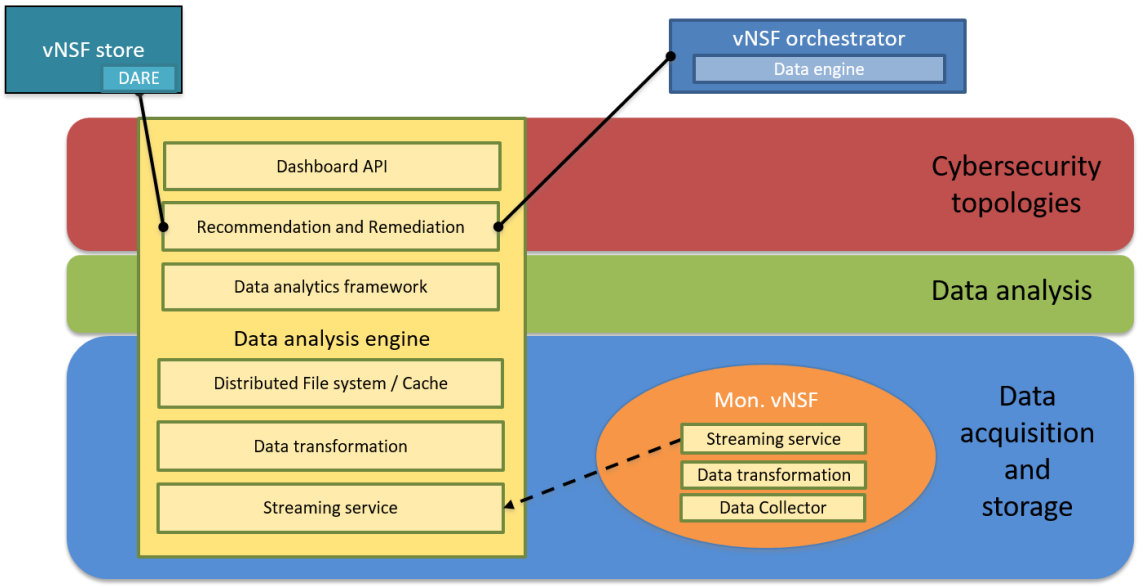

Figure 3: Architecture of the DARE

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 sends the information from the monitoring vNSF to the data analytics central engine, assuring reliability on the communication.

Distributed File System / Cache: it is 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 proposes, 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 pushes all the generated information to the Dashboard.

3. UPDATES SINCE D4.2

In this section, all updates at the different levels (architecture and design, development) per WP4 component since D4.2 are listed, in order to better keep track of the development work that has been done to extend the engine's capabilities.

3.1. Data acquisition and storage

The Data Acquisition phase is responsible for the efficient and reliable capture and storage of various heterogeneous data. It involves mechanisms and methods 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.

As described in detail in D4.2, the distributed data acquisition module was implemented following a decentralized schema (Figure 4). This decentralized format is time efficient as far as read/write from/to HDFS [4] is concerned and since only relevant info is sent, network traffic is reduced inside infrastructure.

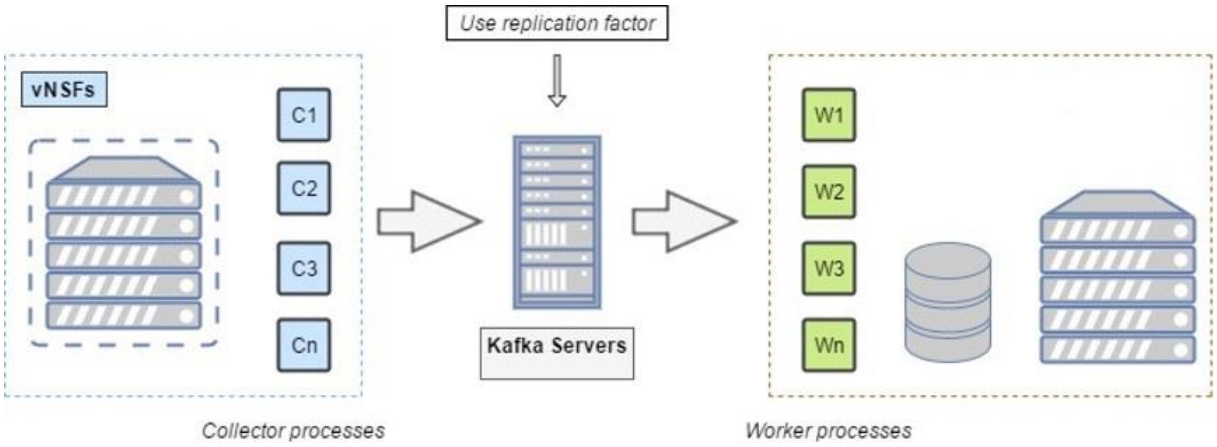

Figure 4: Distributed data acquisition and storage architecture

Distributed Collector

Distributed collector was deployed to solve scalability issues of Spot's original collector. It is fully integrated with vNSFs and supports netflow, DNS, and proxy traffic, security event and data metrics. Essentially what collector does is to monitor the file system and to detect new files. Then these raw files are converted into comma separated files and are published in Kafka topics [5].

Streaming Worker

Streaming Worker was developed to register the processed data into corresponding Hive tables. Kafka topics and partitions are created and files read from each topic are stored to HDFS. Furthermore, records are inserted into Hive tables. Use of Streaming Worker was proven to be more time efficient in terms of time consumed for storage processes.

3.2. Data analysis phase

The DARE leverages two different data analytics modules that export their findings to a shared Remediation engine, to produce optimal results. The Cognitive Data Analytics module is based on the Apache Spot platform [6], an in-development, and open-source project for network telemetry and anomaly detection. The Security Data Analysis module implements a version of Talaia's proprietary network visibility solution based on the SecaaS architecture. The aforementioned modules are heavily modified and functionally enriched, with respect to the fulfilment of the SHIELD's requirements.

3.2.1. Cognitive Data Analysis Module

New Anomaly Detection Algorithms

One of the limits identified in the Apache Spot Framework was the lack of variety of machine learning algorithms. In the case of anomaly detection, only LDA [7] was supported. To overcome this limitation, several algorithms have been introduced by the SHIELD consortium, based on different Python frameworks and libraries. Depending on the selected framework, there was a relative impact on the performance compared to non-distributed processing environments, especially during the algorithms' training phase. To mitigate this problem, the training phase has been removed from the DARE timeline, and introduced as maintenance algorithm lifecycle process (see Re-Train lifecycle capacity).

Autoencoder

Deep learning techniques are starting to see an application in the field of anomaly detection, due to their capability to extract complex features from raw data. For anomaly detection we have used autoencoders, which rely in reconstructing the input signal after going through a compressive path [8]. This type of networks are composed of two main parts, the encoder and the decoder (Figure 5). The encoder's task is to compress the input data into a low dimensionality vector, while the decoder uses this vector as input, and tries to reconstruct the data with minimum losses. After being trained, the autoencoder will have adjusted its parameters to optimally reconstruct data similar to the one it was trained with. However, anomalies will present a high reconstruction error after being forwarded through this architecture. Then, the reconstruction error can be used to label a certain data point as anomaly or not.

For SHIELD, we have created an Autoencoder composed of three layers. The first one containing 16 neurons corresponding to the following variables of the netflow traffic:

- Protocol: has been encoded using on-shot technique producing a vector of 5 variables.
- Flags: have been encoded using one-shot technique producing a vector of 6 variables.
- Duration of the flow.
- Origin and destination ports.
- Number of packets and number of bytes per flow.

The inner layer is composed by 12 neurons and the outer layer contains again 16 neurons. The reasons of the choice of this simple network are mainly two. On the one hand, the number of variables is relatively small (16) and on the other, we want to avoid false positives (normal traffic labelled as malicious).

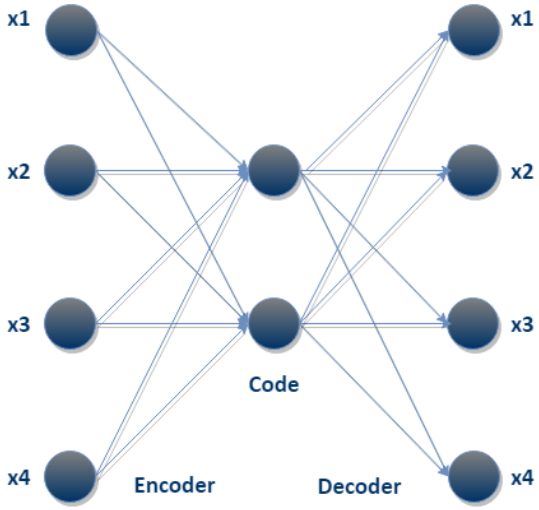

Figure 5: An autoencoder. The phase between the input and the code is called encoder and the phase between the code and the output is called decoder.

One class Support Vector Machines

One class SVM [9] belongs to the unsupervised algorithms group using novelty as detection method. It is based on the applicability of Support Vector Machines (SVM), commonly used in classification supervised learning processes, but simplifying the problem to identify if one data point belongs to one anomaly class or not (one class). The algorithm is estimating the support of input's distribution by identifying regions where most of the cases lie (Figure 6). This algorithm implies two main aspects when we think in the cybersecurity threat landscape. First, being an unsupervised method, it does not require any type of labelling process for the training phase, and second the novelty detection implies that the traffic must be clean in the training phase. These two properties can be very useful in order to work with some types of network cyber-attacks, such as zero-days attacks, where no training dataset or labelling process will be possible. Our implementation, extracts and normalizes relevant data from the netflow protocol, focusing in the relevant features of the network traffic, such as protocol family, ports, IPs, flow duration, bytes and packets per flow or number of similar flows.

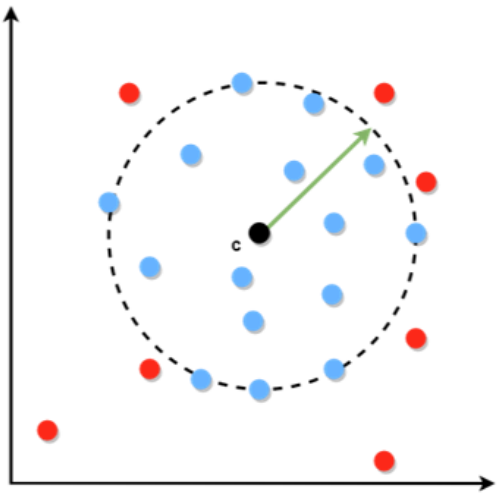

Figure 6: One-class SVM. The points within the decision boundaries are considered normal cases.

Isolation Forest

IForest [10] provides an anomaly detection algorithm, exploiting the concept of "isolated" observations after applying a random forest of decision trees. The reasoning is simple, anomaly observations are easy to isolate because they will show a significantly shorter path length (Figure 7). IForest is suitable for huge amount of datasets and shows an acceptable memory usage [11] [12]. Precisely, these properties make IForest a very promising technique to apply in anomaly detection for cybersecurity incidents based on huge network flows, such as Telco or big corporations with massive traffic. Also, it's worth mentioning that the training process can be achieved with normal and anomalous traffic in the same dataset, thus making it valid for production environments.

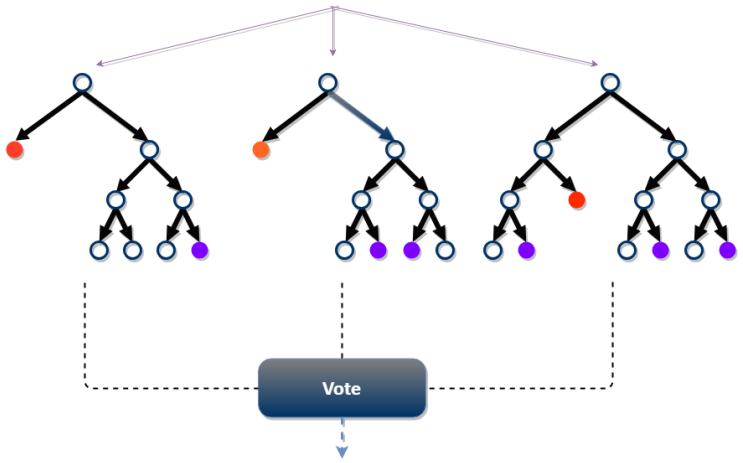

Figure 7: Isolation Forest. Outliers (red) are less frequent than regular observations and require less splits (closer to the root of the tree).

Local Outlier Factor

LOF [13] is another anomaly detection algorithm based on outlier detection, where data points are seized using local density deviation with respect to their k-nearest neighbours. Regions of similar density correspond to normal data points, while points that have a substantially lower density than their neighbours can be considered to be outliers (Figure 8). This algorithm has shown promising results in some type of cybersecurity attacks, such as network intrusion [14], but suffers from a high computational resources demand.

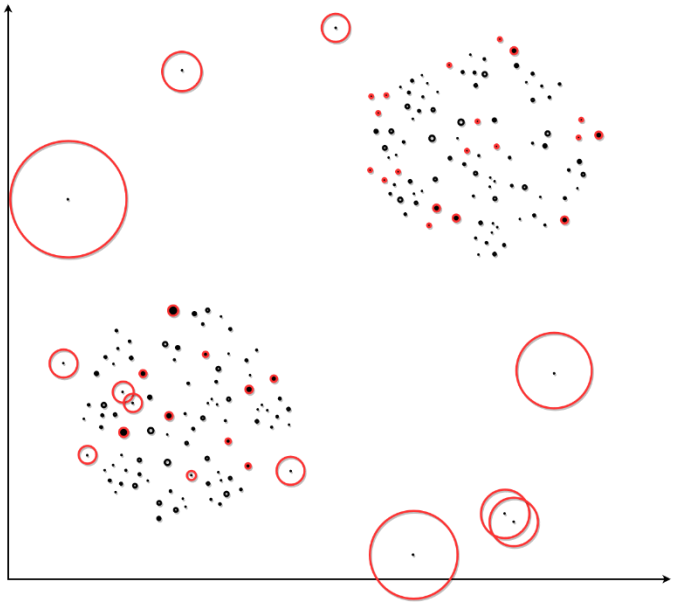

Figure 8: Local Outlier Factor. Points that have a substantially lower density than their neighbours can be considered to be outliers.

New Threat Classification algorithms

While anomaly detection focuses on distinguishing outliers of network activity from normal traffic, threat classification is a classification technique that assigns these outliers to one of several predefined threat classes. Classification algorithms (also called classifiers) are supervised learning methods, meaning that they are trained with labelled datasets to recognize distinctive patterns, in order to classify all activity based on this information. From an intrusion detection perspective, classification algorithms can work in parallel with anomaly detection or in addition to it, to characterize anomalous network activity as malicious, benign, scanning, or as any other threat category of interest, using information like source/destination ports, IP addresses, and the number of bytes sent during a connection. The following threat classification algorithms have been implemented for the DARE Cognitive DA module:

Random Forest

Random forest is an ensemble supervised machine learning method used for classification. It constructs a multitude of decision trees at training time and outputs the mode of the classes (the most repeated value) of the individual trees as the final class [15]. Essentially, each tree's prediction is counted as a vote for one class and the final label is predicted to be the class which receives the most votes (majority vote) (Figure 9). The algorithm applies the general technique of bootstrap aggregation (or bagging) to tree learners, leading to a better performance model by decreasing the variance, without increasing the bias [16]. Random forest is considered one of the best-performing ML algorithms [17], mainly because of its ability to remove decision trees' habit of overfitting the training set (being too much dependant of the training set and not performing so well in the testing set) [18] and of its unmatched classification accuracy compared to current algorithms. In the case of network traffic classification, the datasets are usually unbalanced since the majority class (normal traffic) is usually orders of magnitude

higher than the minority classes (attack flows). Therefore, classifiers are overwhelmed by the dominating class and tend to ignore the flows related to malicious activity. Random forest is of no exception, thus techniques like cost-sensitive learning and oversampling of the minority class are leveraged to tackle this issue.

For our implementation, the scalable Spark MLlib framework [19] was used to design a Random Forest model of 50 trees, which was trained and evaluated using the datasets described later. Since Spark provides APIs in non-JVM languages such as Python, many data scientists use the latter, as it has a rich variety of numerical libraries with a statistical, machine-learning, or optimization focus. A parameterization grid was set to select the optimal values for the maximum tree depth (length of the longest path from a root to a leaf) and feature subset size (number of features to consider for splits at each node).

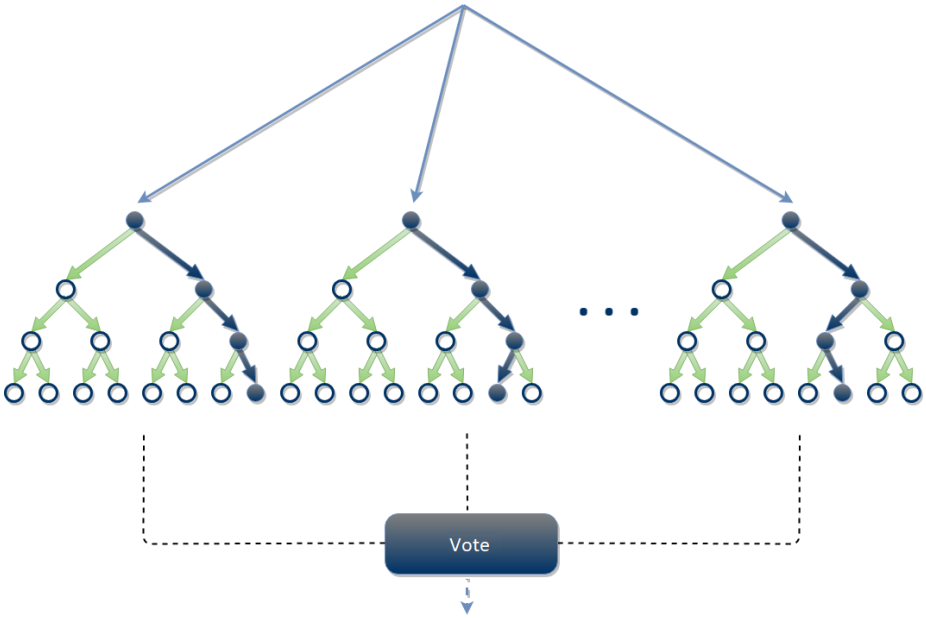

Figure 9: Random Forest. It constructs a multitude of decision trees at training and outputs the mode of the classes of the individual trees.

Multi-Layer Perceptron (MLP)

MLP is a class of feedforward artificial neural networks, consisting of at least three layers of nodes (input, hidden, and output layers). In MLP, each neuron unit calculates the linear combination of its real-valued inputs and passes it through a threshold activation function (Figure 10). Learning occurs iteratively, by changing connection weights after each piece of data is processed, based on the amount of outputted errors compared to the expected result (backpropagation). The use of non-linear activation functions in the neural nodes can be implemented to reproduce a nonlinear function mapping, allowing to solve non-linearly separable problems, such as network anomaly classification [20] [21]. Using a carefully chosen set of features of the Netflow protocol as input signal, we were able to train and compare several MLP architectures to classify multiple normal and anomalous states, using the Deep Learning Studio platform [22] which leverages the open-source Keras neural network library [23].

The proposed architecture involves an input layer, a batch-normalisation layer, two hidden dense layers consisting of 36 and 12 nodes respectively and the output layer. The rectified linear unit (ReLU) is chosen as the activation function of the hidden dense layers, while Softmax is used for the output layer. The model was trained for 10 epochs using the Adagrad optimizer and categorical cross-entropy as loss function. The selected MLP model was integrated in the Cognitive DA module of the DARE, and it was further compared to the Random Forest classifier in terms of speed, robustness, and accuracy in capturing the essence of this system.

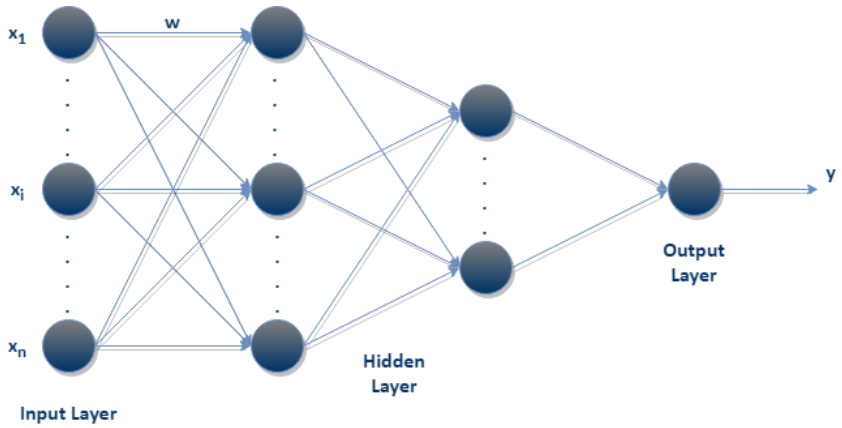

Figure 10: MultiLayer Perceptron. Each neuron unit calculates the linear combination of its real-valued inputs and passes it through a threshold activation function.

Re-Train lifecycle capacity.

Cognitive models re-train lifecycle management allows the DARE, to re-train different algorithms over production environments with fresh datasets obtained during data acquisition and storage phase at specific intervals. This new capability of the Cognitive DA module is represented in Figure 11 and it can be deployed as a parallel process to avoid delays in the detection phase due to training.

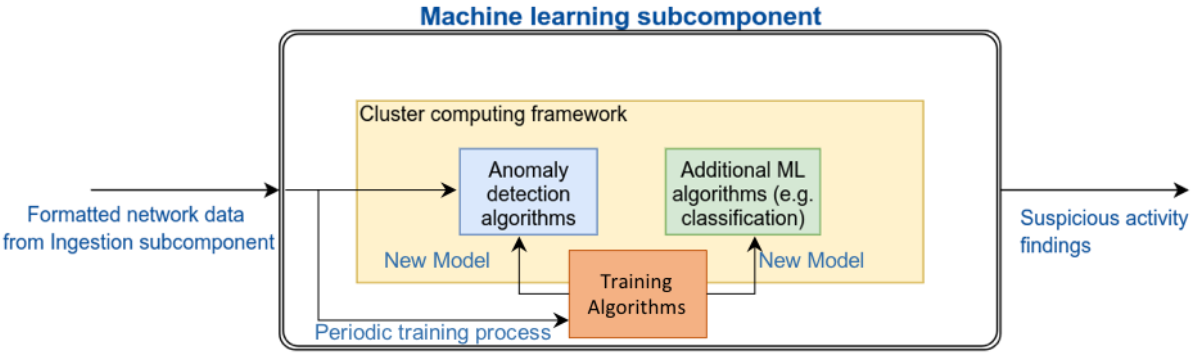

Figure 11: Periodic re-train lifecycle management of the DARE modules

The re-training process can be adjusted to occur periodically, and for a specific time period, e.g. 24h. It utilizes the network data from the ingestion subcomponents to produce new models that substitute the original ones. This allows us to update the algorithms over time, accounting for any differences in the network patterns, i.e. traffic increase or user behavioural changes. This is particularly useful for newly added anomaly detections algorithms. To this end, our new

implementation includes the feature of training with data from one specific date and exporting the trained model to be used for anomaly detection of the future traffic.

3.2.2. Security Data Analysis Module

Traffic Ingestion

The Security DA Module is configured to ingest traffic from the distributed collector. The ingestion is performed by a Python service which monitors the HDFS for new traffic captured by the distributed collector. As soon as the traffic becomes available, the service will fetch it, transform it and send it to the anomaly detection engine to be ingested.

Anomaly Detection

The detection capabilities of the security DA module were extended to allow the detection of cryptocurrency mining activity inside the monitored network. More specifically, the machine learning algorithms that are part of Talaia's anomaly detection engine were trained to be able to identify the Stratum protocol which is the de facto protocol that is currently being used to allow the communication between the mining software and the mining pool server. The detection of Stratum traffic automatically generates a new anomaly notification that includes all the flows that were generated from the mining activity.

The Stratum protocol is used by a wide range of mining software and for mining different cryptocurrencies such as Bitcoin, Ethereum, Litecoin, etc. Therefore, by detecting Stratum traffic, the security DA module is capable of identifying cryptocurrency mining activity independent of the configuration and the type of the mined currency. Moreover, the detection only relies on the analysis of netflow traffic and does not require instrumentation and monitoring of the user equipment nor advanced DPI software to analyse the traffic down to the packet level.

Anomaly Notification

A monitoring service polls the engine frequently for new anomalies and as soon as a new one is detected, it is sending all related logs to the Recommendation and Remediation Engine and SHIELD's Dashboard in the form of RabbitMQ messages [24].

3.3. Cybersecurity topologies phase

The cybersecurity topologies component (Figure 12) is in charge of analyzing the attack data generated by the machine-learning modules of the DARE, decide an appropriate response to mitigate the threat, produce a set of medium-level policies and send them to the dashboard as recommendations, so that an administrator can be noticed and take remediation action.

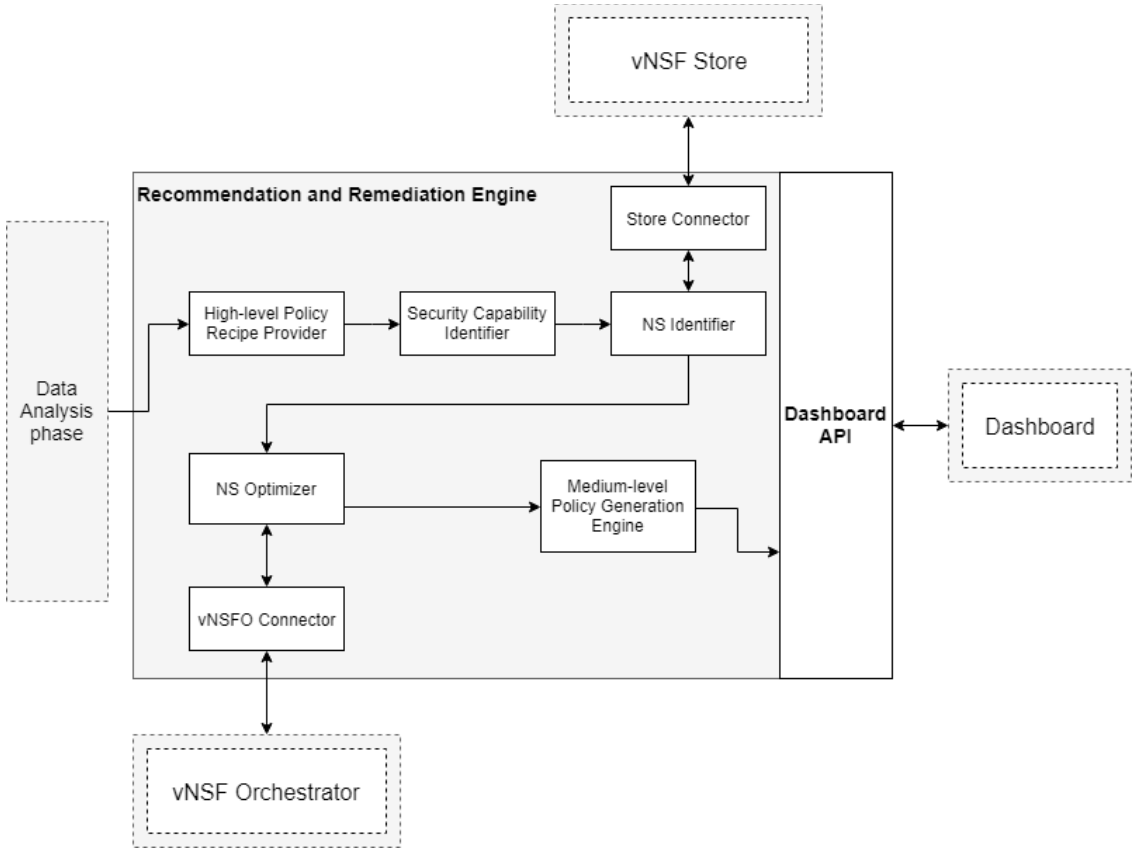

Figure 12: Overview of the Recommendation and Remediation Engine

Full RabbitMQ support

The initial prototype of the cybersecurity topologies module reacted to new attacks by listening to a directory where an external entity had to copy a CSV file containing the attack report and used a RabbitMQ queue only to send the policies to the dashboard.

The updated version of this module now fully supports also a RabbitMQ queue that can be used to receive the attack CSV data instead of copying a file in a specific watched folder. This has two main advantages over the previous approach. First, this method allows multiple attack reports to be sent in parallel, since each CSV file line is sent as a separate message. Second, it makes more homogeneous the interaction and integration between the DARE and the dashboard since now every module uses RabbitMQ queues to exchange the data.

Constraints for the attack destination

The attack remediation recipes can contain zero or more constraints that modify the produced policies. As an example, these constraints can be used to force the use of the “any port” value in the policies (this is useful, for instance, to write node isolation policies).

The initial prototype of the cybersecurity topologies module allowed to place most of the constraints only on the source of the attack, while now it is possible to put these constraints also on the attack destination (i.e. the victim). This allows to write more flexible recipes and in turn to produce better tailored remediation policies.

3.4. Dashboard

The Dashboard component enables users and applications to access SHIELD's internal features, therefore being the entry point of SHIELD solution. The main architecture of Dashboard has not changed since D4.2, however their specified capabilities were implemented, enabling multi-user lifecycle management, real-time notifications, incident summary, attestation and remediation (Figure 13).

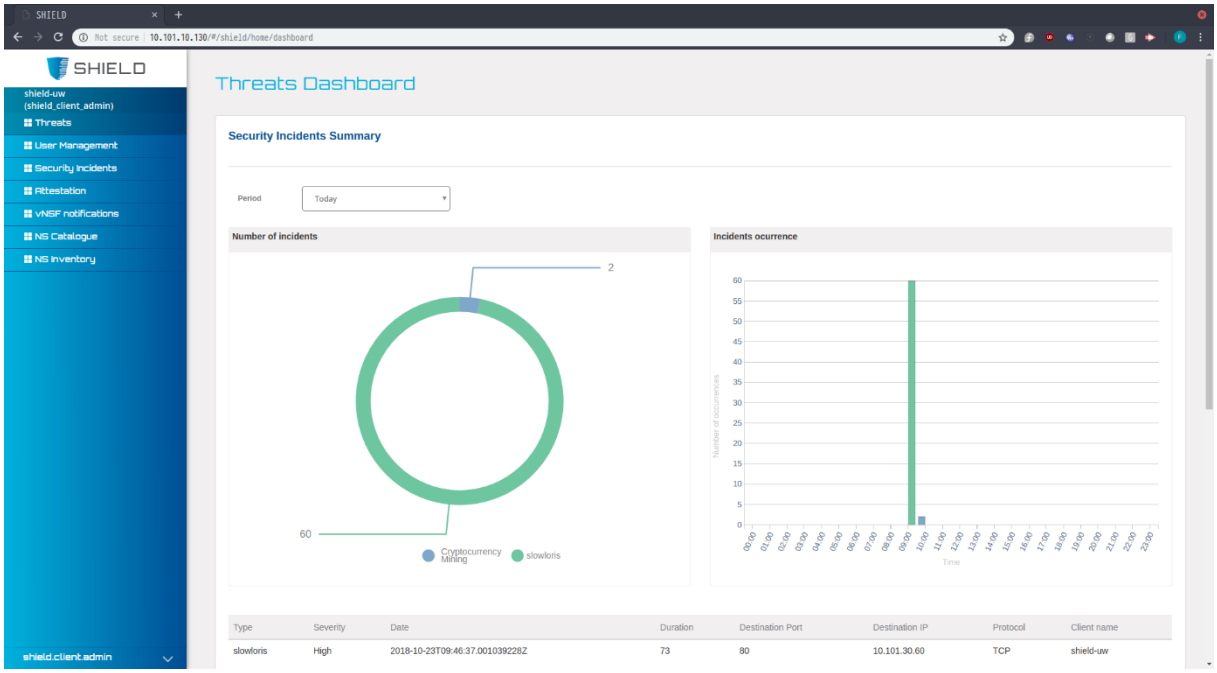

Figure 13: Security Incidents summary of the Dashboard

In the current version of the Dashboard, multiple types of users can be created, namely Developers, SecaaS Clients and Cybersecurity Agents. For each SecaaS client multiple users can be created and associated with it. Each SeccaS client user is tied to a specific role which bounds it to perform a set of restricted actions, for instance a SecasS client admin has the ability to instantiate a service whereas remaining users can only view the instantiated services.

Real-time notifications were enhanced, supporting vNSFO service instantiation status notifications, Trust Monitor notifications and Store onboarding notifications.

The security attestation of vNSFs may result in remediation actions which are reported from the DARE to the Dashboard. From these actions, the client administrator can now choose the remediation and apply it to rectify a particular vNSF.

4. REQUIREMENTS MAPPING

Based on D4.2, we address four types of requirements: i) platform requirements (PF), ii) non-functional requirements (NF), iii) service requirements (SF) and iv) regulatory compliance requirements (ERC). In this section, the compliance of WP4 components to i), ii) and iii) is briefly justified from a general point of view (Table 1). The regulatory compliance of the DARE and the Dashboard is discussed in the Annex of this document.

Table 1: Compliance of WP4 components to PF, NF and SF requirements.

Component	Requirements	Justification
DARE	PF04, PF08, PF12, PF13, PF16, PF18, PF21, PF22	The DARE offers real-time monitoring of network traffic collected by the vNSFs, by leveraging its set of distributed collector and streaming worker modules. It also utilizes machine-learning algorithms optimized for distributed computing frameworks, to analyse the ingested traffic in near-real-time. The platform is expandable, having already showcased that it can support several analytics engines based on open-source principles. When an attack is detected, information is provided to the Remediation Engine in order to construct a recommendation message that is then forwarded to the Dashboard in an encrypted manner. The platform retains historic data stored in its distributed file system, available for processing and reporting.
	NF01, NF02, NF03, NF04, NF05, NF06, NF07, NF08	The platform incorporates SotA scalable, streaming processing technologies (Hadoop, Hive, Spark, Kafka) to process and analyse the ingested data in a relatively short time, regardless of its volume. When performance is degraded due to increasing data volume, adding more data nodes to the infrastructure can improve the system's performance significantly. Load-balancing and resource management functionalities (YARN) are also integrated to its infrastructure to ensure a stable user experience. All DARE modules are easily installed and maintained, following the steps of their detailed documentation. Finally, the engine inputs and exports data in ways that conform to well-established data input formats (Netflow).

	SF01, SF02, SF03, SF04, SF05, SF06, SF07, SF08, SF09, SF10, SF11, SF12	The engine already complies with all mandatory service functional requirements: It offers effective mitigation mechanisms for malware spreading and volumetric Denial of Service attacks. It supports advanced processing services, such as L4 traffic filtering, multi-source log monitoring, correlation and alert generation (Remediation engine). It is capable of detecting attacks with a wide range of techniques (.nfcapd and .pcap protocols), similar to an IDPS. The DARE doesn't support optional service functional requirements such as phishing detection, honeypots and sandboxing techniques
Dashboard	PF03, PF05, PF06, PF07, PF09, PF12, PF13, PF14, PF15, PF16, PF17, PF20, PF21, PF22	The operator is able to control the lifecycle of vNSFs via GUI, supporting the onboarding, instantiation, chaining, configuration, monitoring and termination of vNSFs. Different users can be created for the same SecaaS client by the SecaaS admin. Any access to the SHIELD platform is protected by authentication and authorization mechanisms. A particular user, namely Cyberagent, is created for each SecaaS instance in order to share logs with a third-party identity, namely a Cybersecurity Agency. It is possible to trigger remediation actions in order to mitigate threats and it is also possible to monitor in real-time the detected threats in a particular vNSF.
	NF01, NF02, NF03, NF07, NF08, NF09	The notification reports about a particular action take place in a matter of seconds, however the triggering of actions to third-party modules can take longer, e.g. instantiation of a service. The setup and execution of Dashboard can be performed by a single command.
	SF05, SF11, SF12	Optional service functional requirements are not supported yet, but may be considered for the final release.

5. ENVIRONMENT SETUP GUIDE

5.1. CDH framework installation and configuration

CDH [25] is an Apache-licensed open source framework and is considered as the most popular distribution for Apache Hadoop and related projects. CDH delivers the core elements of Hadoop, scalable storage and distributed computing, along with a Web-based UI and enterprise capabilities (Figure 14). CDH provides multiple benefits in terms of flexibility, integration, scalability, security, high availability and compatibility.

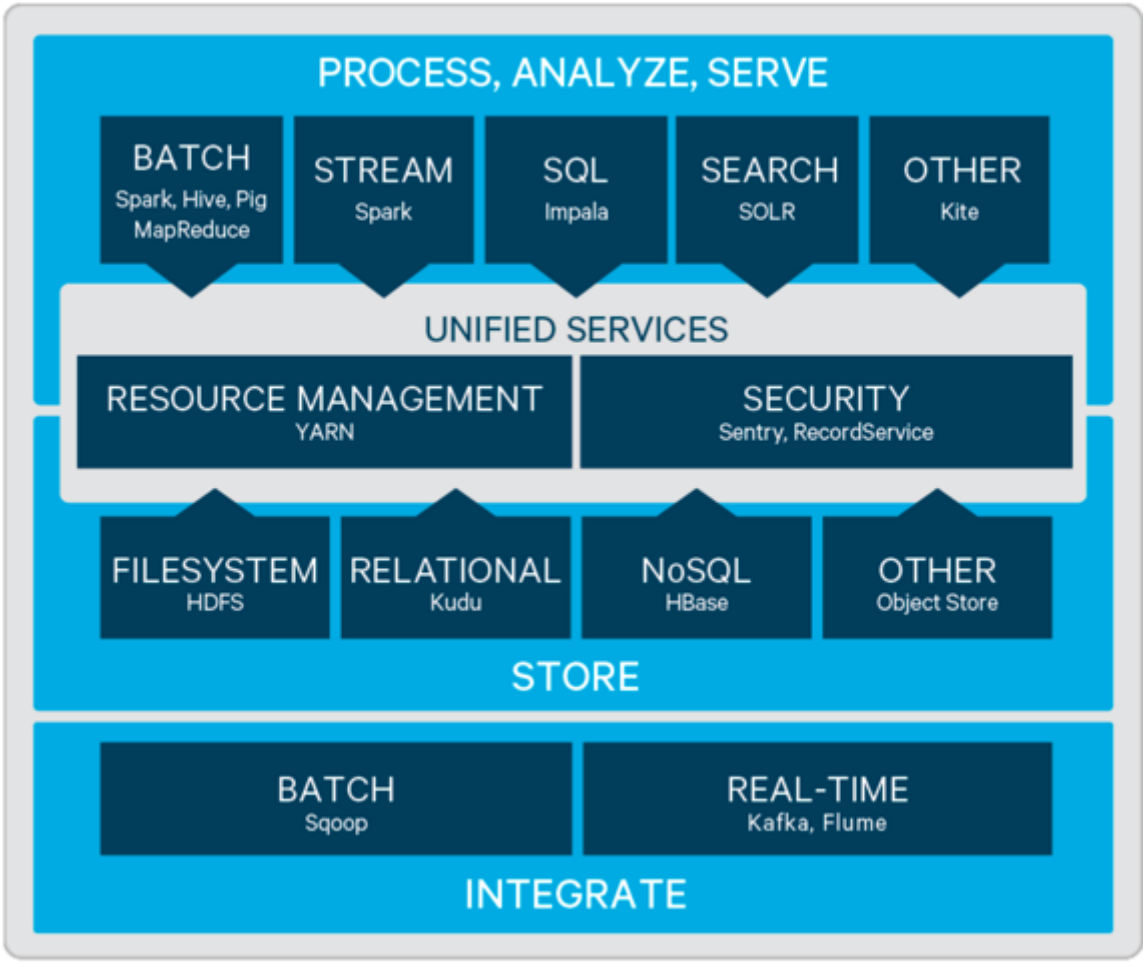

Figure 14: List of CDH supported services

Hardware specifications

Our proposed CDH setup comprises of 3 Ubuntu 16.04 VMs running on ESXi hosts. The first VM is the Cloudera Master host and runs Hadoop master processes such as the HDFS NameNode and YARN Resource Manager. The second VM is the Cloudera Edge Node host and acts as the client access point for launching jobs in the cluster. Finally, the last VM is the Cloudera Worker

host which runs DataNodes and other distributed processes such as Impala. Following are the resource specifications of the hosts as well as the roles assigned to each host.

Table 2: Resource specifications and assigned roles of hosts.

Hosts	# of CPUs	Memory (GB)	Disk (GB)	Roles
Cloudera Manager	8	20	400	· HDFS DataNode · HDFS HttpFS · HDFS NFS Gateway · HDFS SecondaryNameNode · Hive Gateway · Hive MetaStore Server · HiveServer2 · Impala Daemon · Kafka Gateway · Kafka MirroMaker · Activity Monitor · Alert Publisher · Event Server · Host Monitor · Service Monitor · Spark Gateway · YARN JobHistory Server · YARN NodeManager · YARN ResourceManager
Cloudera Edge Node	8	20	400	· HDFS Balancer · HDFS NFS Gateway · HDFS NameNode · Hive Gateway · Impala Catalog Server · Impala StateStore · Kafka Gateway · Kafka Broker

				· Spark Gateway · ZooKeeper Server
Cloudera Worker	8	20	400	· HDFS DataNode · HDFS NFS Gateway · Hive Gateway · Impala Daemon · Kafka Gateway · Kafka Broker · Spark Gateway · Spark History Server · YARN NodeManager

CDH environment setup and configuration

The installation procedure is sufficiently documented in Cloudera's installation guide and requires 7 main steps as follows:

Step 1: Configure a Repository:
https://www.cloudera.com/documentation/enterprise/6/6.0/topics/configure_cm_repo.html#cm_repo

Step 2: Install JDK:
https://www.cloudera.com/documentation/enterprise/6/6.0/topics/cdh_ig_jdk_installation.html#topic_29

Step 3: Install Cloudera Manager Server:
https://www.cloudera.com/documentation/enterprise/6/6.0/topics/install_cm_cdh.html#cmig_topic_6_6

Step 4: Install Databases:
https://www.cloudera.com/documentation/enterprise/6/6.0/topics/cm_ig_installing_configuring_dbs.html#cmig_topic_5

Step 5: Set up the Cloudera Manager Database:
https://www.cloudera.com/documentation/enterprise/6/6.0/topics/prepare_cm_database.html#cmig_topic_5_2

Step 6: Install CDH and Other Software:
https://www.cloudera.com/documentation/enterprise/6/6.0/topics/install_software_cm_wizard.html#cm_installation_wizard

Step 7: Set Up a Cluster:
https://www.cloudera.com/documentation/enterprise/6/6.0/topics/cluster_setup_wizard.html#concept_b4d_wkh_ycb

6. INSTALLATION GUIDE

6.1. Open Source status

The WP4 components are available at the SHIELD organization's public repository in Github: https://github.com/shield-h2020. All of them with the exception of the Security DA module are shipped under the Apache 2.0 license. The latter is derived from the security module of Talaia's commercial product and is integrated into the SHIELD platform as a closed source, stand-alone component.

The data acquisition components (namely the Distributed Collector and the Streaming Worker) are located at the repositories named "vnsfs-collectors" and "dare-workers" respectively, under the SHIELD organization, with the link provided above. The analytics components (Apache Spot and the anomaly detection and threat classification components of the Cognitive DA module) are located at the "dare" repository. Finally, the Remediation Engine component is available at the "dare-sec-topo" repository and the Security Dashboard is located at the "dashboard" repository. The contents of all these repositories provide the source code of their component, the installation and deployment scripts and the documentation regarding setup, configuration and deployment. All the developments that were presented in the demos corresponding to the year two review have been marked as a release tag v0.2 in the repositories.

6.2. Apache Spot

The DARE Cognitive DA leverages Apache Spot [6] as a built-in, community-driven cybersecurity solution, to bring advanced analytics to all IT Telemetry data on an open, scalable platform. It is an open-source software for leveraging insights from flow and packet analysis which expedites threat detection, investigation, and remediation via machine learning and consolidates all enterprise security data into a comprehensive IT telemetry hub based on open data models. Spot's scalability and machine learning capabilities support an extendable ecosystem of ML-based applications that can run simultaneously on a single, shared, enriched dataset to provide organizations with maximum analytic flexibility.

Apache Spot version 1.0 was released in August 7, 2017 and is the latest stable release used by the DARE. It can be cloned from the following link (as tarball):

https://www-us.apache.org/dist/incubator/spot/1.0-incubating/apache-spot-1.0-incubating.tar.gz

Alternatively, you can clone our forked repository[1] which contains all relevant DARE modules, including Apache Spot.

You can verify this release using 1.0 signatures and checksums[PGP, SHA-512, MD5] with project release KEYS.

In order to validate the build follow the instructions:

[1] https://github.com/shield-h2020/dare

- Download the tarball from the above link.
- Decompress the tarball:
  `tar -zxvf apache-spot-1.0-incubating.tar.gz`
- Change directory:
  `cd apache-spot-1.0-incubator`

Since Apache Spot is composed of more than one module or sub-projects, some of them need compilation, while others (Python or Javascript based) don't.

To install each module please follow the below instructions:

- Decompressed tarball content should be the same with the content located in:
  https://github.com/apache/incubator-spot/tree/v1.0-incubating
- To install each component please follow the official documentation guide:
  http://spot.apache.org/doc/#installation

- Spot-ingest (centralized), Spot-setup, Spot UI and Spot-OA have specific requirements to install manually.
  http://spot.apache.org/doc/#configuration
  http://spot.apache.org/doc/#ingest
  http://spot.apache.org/doc/#oa
  http://spot.apache.org/doc/#ui
- Spot-ML is the only component to build the binary files using sbt assembly commands. To install it follow these instructions:
  http://spot.apache.org/doc/#ml

We have realised that due to the work that has been done during the first year 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 impose 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, 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 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 must 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 second iteration of the project design and we enter the second iteration of the development phase, having drafted a clear technical roadmap till the end of the project.

6.3. Distributed Collector and Streaming Worker

Distributed collector is a daemon that runs in each vNSF (one collector per vNSF). When a new file is created, collector detects it and then the file is decoded and translated into comma-separated output with specific structure. Then, the Apache Avro serialization framework [26] is used to convert the output to an Avro-encoded format and send it afterwards to the Apache Kafka streaming platform. In the Kafka cluster, published messages are consumed by streaming workers daemons in the analytics engine. The role of the Streaming Worker is to listen to a particular topic of the Kafka cluster and consume the incoming messages. Streaming data is divided into batches (according to a time interval). These batches are deserialized by the Worker, according to the supported Avro schema, parsed and registered in the corresponding table of Hive. Streaming Worker can be deployed in local, client or cluster mode.

Installations thus, includes dependencies in Python regarding:

- Avro serialization framework
- Kafka-python for the Apache Kafka distributed system
- Watchdog – a Python API and shell utilities that monitor file system events

Moreover, in Linux OS there is a dependency with pip the Python package manager and in order to process specific pipelines, installation on appropriate tools is needed:

- Spot-nfdump – a version for processing netflow
- Tshark – a part of wireshark distribution for processing pcap files.

Required installation and dependency files along with detailed configuration can be found at SHIELD public repositories[2][3].

[2] https://github.com/shield-h2020/vnsfs-collectors
[3] https://github.com/shield-h2020/dare-workers

6.4. Cognitive DA module

6.4.1. Anomaly Detection DL

The Autoencoder is developed using Python and using the PySpark library to run it in SPOT architecture based on Spark. Moreover, as the Autoencoder uses neural networks, it is necessary to install a deep learning framework. In this case we have used BigDL [27], a framework supported by Intel that is focused on Deep Learning in CPUs (in contrast to most frameworks that are designed only for GPUs).

Installation includes the following steps:

1. Install BigDL
2. Install Python
3. Install PySpark and Numpy
4. Download the code from SHIELD public repository [4]
5. In order to execute the detection use the following command:

```
./ml_security_zoo.sh PHASE TYPE YYYYMMDD
./ml_security_zoo_csv.sh PHASE TYPE YYYYMMDD
```

where:

`./ml_security_zoo.sh` is for parquet data sources.

`./ml_security_zoo_csv.sh` is for csv data sources.

PHASE can be *train* or *test*, for training or detection stages.

TYPE should be *flow*. (a.k.a. netflow)

YYYYMMDD specific date to use for training or current date for detecting anomalies.

6.4.2. Anomaly Detection ML

Anomaly detection ML installation expands the algorithms available in the Apache Spot ML module from DARE. Installation includes the following steps:

1. Install python library dependencies such as pandas, numpy or scikit-learn
2. Clone or download the spot-anomalies module developed from SHIELD public repository.
3. Edit the Apache Spot configuration file (spot.conf) to customize relevant parameters training and testing dataset paths, specific to algorithms parameters (OCSVM, iForest, LOF)

In order to manage this module, a common command is provided:

```
./ml_security PHASE TYPE YYYYMMDD
```

where:

PHASE can be train or test, for training or detection stages.

TYPE should be flow. (a.k.a. netflow)

[4] https://github.com/shield-h2020/dare/

YYYYMMDD specific date to use for training or current date for detecting anomalies.

Anomaly detection module installation, detailed configuration and management is depicted in the public SHIELD GitHub repository[5].

6.4.3. Threat Classification ML

The Apache Spark distributed-computing framework is leveraged to implement a scalable Random Forest classifier using the MLLib library. Our implementation utilizes the PySpark API to train a Random Forest model using labeled data, which can then be used to label the outliers detected by the Anomaly Detection modules. Since our installation involves an existing CDH cluster, our implementation can be executed in a distributed manner across all nodes, with the help of the YARN cluster manager. When run on YARN, Spark application processes are managed by the YARN ResourceManager and NodeManager roles.

Please note that this module requires Spark version 2.2.1 or higher and Python 3.4 or higher. Follow the instructions below to install the Random Forest classifier to any node of the CDH cluster:

1. Download or clone the Random Forest module developed from SHIELD public repository[6].
2. Install the necessary Python3 libraries, following the module's documentation.
3. You can either use the pretrained Random Forest model, or you can train your own using a labelled dataset with the following command:
   `./classifier3.sh train <HDFS_PATH_OF_LABELED_DATA> <#_OF_TREES(opt.)>`
4. You can make predictions for unlabeled netflow data that is ingested and stored in the Hive DB, with the following command:
   `./b_sim_anom_class_pub.sh <YYYYMMDD>`

The output of the classification procedure is being saved at the path where the trained model exists. You can edit these paths by modifying the threat_classifier0.4.py script. The classification report is sent to the correct topics of the Remediation Engine and to the Dashboard via the RabbitMQ queue.

6.4.4. Threat Classification DL

The MultiLayer Perceptron classifier serves as an alternative to the Random Forest classifier described above, and further expands our data analytics solutions. It is developed using the desktop version of the Deep Learning Studio, which is compatible with a number of open-source programming frameworks, popularly used in artificial neural networks, including MXNet and Google's TensorFlow.

Our netflow classification model was developed using the Keras environment, a high-level neural networks API, written in Python and capable of running on top of TensorFlow, developed with a focus on enabling fast experimentation. The trained model can be imported and executed on a node of the CDH cluster, following the steps below:

1. Download or clone the MLP-classifier module developed from SHIELD public repository[7].

[5] https://github.com/shield-h2020/dare
[6] https://github.com/shield-h2020/dare/tree/master/classifier_ml
[7] https://github.com/shield-h2020/dare/tree/master/classifier_dl

2. Install necessary Python3 and libLAS packages (if not already installed).
3. Install Tensorflow, Keras and sklearn dependencies
4. Perform the classification procedure on netflow data using the trained model, by running the following command:
   `python3 test.py <NETFLOW_FILE.CSV> tensorflow`

The results will be output to `test_result.csv` on the same directory with the model.

6.5. Security DA module

The installation of the Security DA Module depends on the installation of an instance of Talaia's engine[8] that is running the latest anomaly detection models. In the same environment it is required to copy the two python-based scripts that are responsible for performing the traffic ingestion and the anomaly notification tasks, `watch_nfcapd.py` and `anomaly_reporter.py` respectively.

The `watch_nfcapd.py` requires the configuration of the IP address of the HDFS namenode of the Apache Spot cluster and the directory in which the distributed collector saves the collected traffic.

The `anomaly_reporter.py` requires the configuration of the RabbitMQ settings so that the logs from the anomaly reports can be sent to the correct topics and subsequently read by the Dashboard and the Recommendation and Remediation engine.

As soon as the configuration is complete, these two scripts can be ran from the command line without any further arguments.

6.6. Recommendation and Remediation Engine

The Remediation Engine is a collection of Python 3 scripts, available at the SHIELD public repository[9] and they require a quick and minimal setup in order to be configured and running. You can find the

Installation steps:

1. First, you need a working Python 3 environment. Installing the pip3 package is also suggested. In a Debian/Ubuntu Linux environment this can be accomplished by typing:
   `# apt install python3 python3-pip`
2. Once the remediation engine source code has been downloaded, the additional Python packages can be installed by issuing the command:
   `$ pip3 install -r requirements.txt`
3. Now, the actual remediation engine component can be installed in the current Linux box:
   `$ python3 setup.py install`
4. It is also suggested to launch the integrated tests by issuing the command:
   `$ python3 setup.py test`
   All the tests *must* succeed.
5. The module can be used as a standalone daemon by using the following command:
   `$ python3 daemon.py -c /path/to/cybertop.cfg -l /path/to/logging.ini`

[8] https://www.talaia.io/Big-Data-Engine
[9] https://github.com/shield-h2020/dare-sec-topo

The two files `cybertop.cfg` and `logging.ini` are used to respectively configure the daemon itself and the verbosity of its logs. Some examples are already provided in the `tests` folder.
Alternatively, the module can also be installed as a system service so that it is automatically started when the system boots. This can be achieved by executing:
`# daemon/cybertop_systemd_install.sh`

6. To setup the service the configuration file `/etc/default/cybertop` must be manually edited. In particular it must be modified in order to find the `cybertop.cfg` and `logging.ini` configuration files for the actual daemon configuration.

6.7. Dashboard and APIs

The Dashboard component is deployed as a multi-container application using the Docker Compose tool. The only requirements for the host system are Docker (17.06.0 or later) and Docker Compose (3.0 or later).

You can download or clone the Dashboard module from SHIELD public repository[10]. The setup and execution of the Dashboard can be performed using the provided "run.sh" script, for instance:

```
./run.sh --environment .env.production --verbose
```

This command will raise the application containers and use the configuration environment for production. Different environment configurations can be added or modified by creating or changing ".env" files.

In the case of Dashboard being setup for the first time the following additional command must be executed in order to initialize the database:

```
docker exec -it docker_dashboard-persistence_1 bash -c
"/usr/share/dev/dashboard/docker/setup-datastore.sh --environment
.env.production"
```

To be noted that the "setup-datastore.sh" script also uses an environment configuration (.env.production in this example) which should be the same of the one used in the "run.sh" script.

From this point on, the ISP operator can access the Dashboard frontend in http://DASHBOARD_GUI_HOST_WEB , where the "DASHBOARD_GUI_HOST_WEB" is the IP/hostname defined in the environment configuration file. The default ISP operator credentials are:

```
username: admin
password: adminpass
client:   default
```

[10] https://github.com/shield-h2020/dashboard

7. CONCLUSIONS

7.1. Present status

This document presents the final version and the technical details of the information-driven Data Analysis and Remediation Engine as well as of the Dashboard. The document starts with describing the final layout of the DARE architecture, along with all major updates and additions to the WP4 components, since the last deliverable (D4.2). This facilitates the reader to keep track of all the development work that has been done to extend the engine's capabilities in the last phase of the project.

After this, we provide the final mapping of requirements and how does each component fulfil them. That covers platform functional and non-functional requirements, as well as service functional and ethical compliance requirements (the latter being described at the following Annex). The environment and virtual infrastructure setup, as well as the installation and configuration guides are provided to give a high-level view on how to deploy the SHIELD platform from scratch, by following the provided steps, related to the environment and 3rd party tools required for the platform and to the deployment and configuration for the WP4 components described above.

7.2. Future work

The work of all WP4 tasks (i.e., mostly on the development side) concludes with the delivery of this report. The integration and assessment procedures of the WP4-based components with the rest of components in the SHIELD platform will continue until the end of WP5 and the termination of the project itself. The results of the WP4 activities will be provided in the final, upcoming demonstrations; and may be advertised and exposed through dissemination efforts like events (Winter School) and papers.

8. REFERENCES

[1] SHIELD consortium, "D2.2 Updated Requirements, KPIs, design and architecture," 2017.

[2] SHIELD consortium, "D4.1 Specifications, design, and architecture for the usable information driven engine," 2017.

[3] SHIELD consortium, "D4.2 Updated specifications, design and architecture for the usable information-driven engine," 2018.

[4] "HDFS Architecture Guide," [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. [Accessed March 2018].

[5] "Apache Kafka," [Online]. Available: https://kafka.apache.org/. [Accessed March 2018].

[6] "Apache Spot," [Online]. Available: http://spot.incubator.apache.org/. [Accessed March 2018].

[7] B. e. al, "Latent Dirichlet Allocation," *Journal of Machine Learning Research,* vol. 3, pp. 993-1022 , 2003.

[8] P. Baldi, "Autoencoders, Unsupervised Learning and Deep Architectures," in *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, Washington, 2011.

[9] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor and J. Platt, "Support Vector Method for Novelty Detection," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, Cambridge, 1999.

[10] F. T. Liu, K. M. Ting and Z.-H. Zhou, "Isolation Forest," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, Washington, 2008.

[11] U. Carrasquilla, *BENCHMARKING ALGORITHMS FOR DETECTING ANOMALIES IN LARGE DATASETS.*

[12] R. Domingues, M. Filippone, P. Michiardi and J. Zouaoui, "A Comparative Evaluation of Outlier Detection Algorithms," *Pattern Recogn.,* vol. 74, pp. 406-421, 2 2018.

[13] M. M. Breunig, H.-P. Kriegel, R. T. Ng and J. Sander, "LOF: Identifying Density-based Local Outliers," *SIGMOD Rec.,* vol. 29, pp. 93-104, 5 2000.

[14] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," in *In Proceedings of SIAM Conference on Data Mining*, 2003.

[15] T. K. Ho, "Random Decision Forests," in *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, Washington, 1995.

[16] L. Breiman, "Bagging Predictors," *Mach. Learn.,* vol. 24, pp. 123-140, 8 1996.

[17] M. Fernández-Delgado, E. Cernadas, S. Barro and D. Amorim, "Do We Need Hundreds of Classifiers to Solve Real World Classification Problems?," *J. Mach. Learn. Res.,* vol. 15, pp. 3133-3181, 1 2014.

[18] G. Biau, "Analysis of a Random Forests Model," *J. Mach. Learn. Res.,* vol. 13, pp. 1063-1095, 4 2012.

[19] Apache Spark MLlib , The Apache Foundation, "https://spark.apache.org/mllib/," [Online].

[20] C. Siaterlis and V. Maglaris, "Detecting Incoming and Outgoing DDoS Attacks at the Edge Using a Single Set of Network Characteristics," in *Proceedings of the 10th IEEE Symposium on Computers and Communications*, Washington, 2005.

[21] S. Andropov, A. Guirik, M. Budko and M. Budko, "Network Anomaly Detection Using Artificial Neural Networks," in *Proceedings of the 20th Conference of Open Innovations Association FRUCT*, Helsinki, 2017.

[22] Deep Learning Studio, deepcognition.ai,, "https://deepcognition.ai/products/desktop/," [Online].

[23] Keras: The Python Deep Learning library, "https://keras.io/," [Online].

[24] RabbitMQ, open source message broker., "https://www.rabbitmq.com/," [Online].

[25] CDH - Cloudera Distribution for Hadoop, "https://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html," [Online].

[26] Apache Avro data serialisation framework, "https://avro.apache.org/," [Online].

[27] BigDL, a distributed deep learning library for Apache Spark, "https://bigdl-project.github.io/0.7.0/," [Online].

LIST OF FIGURES

Figure 1: The SHIELD architecture 7
Figure 2: Data flow diagram of the DARE 8
Figure 3: Architecture of the DARE 9
Figure 4: Distributed data acquisition and storage architecture 10
Figure 5: An autoencoder. The phase between the input and the code is called encoder and the phase between the code and the output is called decoder. 12
Figure 6: One-class SVM. The points within the decision boundaries are considered normal cases. 12
Figure 7: Isolation Forest. Outliers (red) are less frequent than regular observations and require less splits (closer to the root of the tree). 13
Figure 8: Local Outlier Factor. Points that have a substantially lower density than their neighbours can be considered to be outliers. 14
Figure 9: Random Forest. It constructs a multitude of decision trees at training and outputs the mode of the classes of the individual trees. 15
Figure 10: MultiLayer Perceptron. Each neuron unit calculates the linear combination of its real-valued inputs and passes it through a threshold activation function. 16
Figure 11: Periodic re-train lifecycle management of the DARE modules 16
Figure 12: Overview of the Recommendation and Remediation Engine 18
Figure 13: Security Incidents summary of the Dashboard 19
Figure 14: List of CDH supported services 22

LIST OF TABLES

Table 1: Compliance of WP4 components to PF, NF and SF requirements. 20
Table 2: Resource specifications and assigned roles of hosts. 23
Table 3: Compliance of WP4 components to PF, NF and SF requirements. 39

LIST OF ACRONYMS

Acronym	Meaning
API	Application Programming Interface
DARE	Data Analysis and Remediation Engine
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
ERC	Ethical and Regulatory Compliance (requirements)
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
IDPS	Intrusion Detection and Prevention System
IP	Internet Protocol
IF	Isolation Forest
ISP	Internet Service Provider
LDA	Latent Dirichlet Allocation
LOF	Local Outlier Factor
MLP	MultiLayer Perceptron
NF	Non-Functional (requirement)
NFV	Network Function Virtualisation
NS	Network Service
PF	Platform Functional (requirement)
PoP	Point of Presence
RF	Random Forest
SecaaS	Security as a Service
SF	Service Functional (requirement)

SP	Service Provider
SQL	Structured Query Language
SVM	Support Vector Machines
TM	Trust Monitor
UC	Use Case
URI	Uniform Resource Identifier
UUID	Universally unique identifier
vNSF	virtual Network Security Function
vNSFO	vNSF Orchestrator

Annex A. REGULATORY COMPLIANCE

D4.2 provides an overview of the EU regulatory ecosystem that affects SDN/NFV adoption such as GDPR, ePrivacy, net neutrality etc. A set of regulatory compliance specifications was created, for each WP4 component that parses personal data in any form. D2.3 also provides compliance requirements. This Annex provides the mapping of Requirements to the individual components, as of Year 2.

Table 3: Compliance of WP4 components to PF, NF and SF requirements.

Components	Requirements	Justification
DARE	ERC01, ERC02, ERC04, ERC05, ERC06, ERC09	The DARE storage and processing components expose methods for accessing and deleting personal identifiable information, as all relevant data (stored in the HDFS and in the Hive DB) is easily retrieved by IP address which can be associated to persons. In terms of analytics, since the DARE offers multi-user support, each user can gain access only to threat results that are relevant to his organisation. This data can be erased upon request or after a user-defined period of time, without affecting the efficiency of the ML-based modules. Moreover, all anomaly detection and threat classification modules are based on open-source technologies and are thus transparent in terms of data processing. The Security DA module is a commercial product and as such some specific details related to the exact algorithmic implementations may not be disclosed.
Dashboard	ERC03, ERC04, ERC05, ERC06, ERC09	The Dashboard provides transparent procedures with regard to all its available services: It presents simple information when a NS is selected, including (contact details of the data controllers, data protection and regulatory compliance info, as well as a log of any action to throttle or block traffic). In case of personal data breach, it conveys a notification to the contact associated with the tenant.