SIMARGL

D2.5 Final version of the SIMARGL toolkit architecture

Work Package 2: Architecture Specification of the SIMARGL Toolkit to Detect and Counter Malware and Stegomalware

Document Dissemination Level

P	Public	☒
CO	Confidential, only for members of the Consortium (including the Commission Services)	☐

Document Due Date: 31/01/2020
Document Submission Date: 31/01/2020

This work is performed within the SIMARGL Project – Secure Intelligent Methods for Advanced RecoGnition of malware and stegomalware – with the support of the European Commission and the Horizon 2020 Program, under Grant Agreement No 833042

Document Information

Deliverable number:	**D2.5**
Deliverable title:	Final version of the SIMARGL toolkit architecture
Deliverable version:	H
Work Package number:	2
Work Package title:	Architecture Specification of the SIMARGL Toolkit to Detect and Counter Malware and Stegomalware
Due Date of delivery:	31/01/2020
Actual date of delivery:	31/01/2020
Dissemination level:	PU
Editor:	ITTI (deliverable responsible)
Contributor(s):	ITTI(deliverable responsible) FUH ACS CNR RoEduNet Pluribus SIVECO TCS WUT
Reviewer(s):	FUH, OPL
Project name:	Secure Intelligent Methods for Advanced RecoGnition of malware and stegomalware
Project Acronym	SIMARGL
Project starting date:	1/5/2019
Project duration:	36 months
Rights:	SIMARGL Consortium

Version History

Version	**Date**	**Beneficiary**	**Description**
A	15.11.2019	ITTI	Initial version
B	07.01.2020	ITTI	Version for the discussion at Poznań plenary meeting
C	13.01.2020	ITTI	After meeting version
D	24.01.2020	ITTI	Intermediate version
E	27.01.2020	ITTI	Prefinal version
F	29.01.2020	Thales	Added description of Honeypot system
G	30.01.2020	SIVECO	Addition to the sections: System Administration GUI, Logging and health monitoring
H	31.01.2020	ITTI	Final version incorporating reviewers remarks and recommendations

Abbreviations and Acronyms

ACRONYM	**EXPLANATION**
ADC	Application Delivery Controller
ANSSI	Agence nationale de la sécurité des systèmes d'information (Eng: National Cybersecurity Agency of France)
API	Application Programming Interface
C&C	Command and Control
CEP	Complex Event Processing
CMS	Content Management System
COTS	Commercial off-the-shelf
CSV	Comma Separated Values
DB	Data Base
DGA	Domain Generation Algorithm
DNS	Domain Name Service
DPI	Deep Packet Inspection
HDFS	Hadoop Distributed File System
HTML	Hypertext Markup Language
HW	Hardware
IOC	Indicators of Compromise
IP	Internet Protocol
ISP	Internet Service Provider
JBI	Java Business Integration
JMS	Java Message Service
JPA/JTA	Java Persistence API/Java Transaction API
JSR	Java Specification Request
LDAP	Lightweight Directory Access Protocol
LEA	Law Enforcement Agency
MIME	Multipurpose Internet Mail Extensions
NREN	National Research and Education Network
OSGI	Open Services Gateway
PML	Punch Machine Learning
RDD	Resilient distributed data set
REST API	Representational State Transfer Application Programming Interface
SaaS	Software as a Service
SAML2	Security Assertion Markup Language 2.0
SDK	Software Development Kit
SIEM	Security Information and Event Management
SMTP	Simple Mail Transfer Protocol
SoC	Separation of Concerns
SOC	Security Operation Center
SSO	Single Sign-On
TI	Threat Intelligence
UCPP	Units of Constant Purchasing Power
XML	Extensible Markup Language

Table of Contents

Table of Figures

Table of Tables

Table of Listings

1. Introduction

This deliverable contains a formalised SIMARGL architecture with the final specification of particular components, dependencies among them and interfaces needed to integrate all functional blocks into the interoperable SIMARGL toolkit.

1.1 Overview

This deliverable provides a conceptual design of the SIMARGL toolkit composed of different functional components. The consortium has reached a consensus on the specific tools to be adopted for the modelling phase, thereby ensuring that work carried out independently from each partner can be easily integrated in a common documentation. Particularly, while designing the software architecture, the principle of "Separation of Concerns" (SoC) has been adopted, so that the main functions of the overall system have been identified and delegated to specific system components. The interfaces and the interactions among the identified components are defined.

In order to meet all the possible functional requirements (based on the T2.2 output), the SIMARGL architecture of the toolkit has been designed following the lambda style architecture concept taking into account the technical requirements in terms of scalability, processing capacity, velocity and storage management.

This document presents the SIMARGL architecture, its components and their interaction, and is structured in multiple sections:

- Section 1: Introduction,
- Section 2 and 3: Logical and technological perspectives of the SIMARGL toolkit architecture,
- Section 4: Interfaces and interactions of SIMARGL ,
- Section 5: Authorization and Access Control aspects,
- Section 6: Logging and health monitoring of particular toolkit modules.

In addition, all the SIMARGL's components (Punch Platform, Orion Malware, CYBELS Sensor, Attach Prophecy, AlSafe DNS, BDE Platform) are presented in the Annex A.

1.2 Scope

The scope of this document is to present the architecture of the SIMARGL toolkit, its components (Punch Platform, Orion Malware, CYBELS Sensor, Attach Prophecy, AlSafe DNS, BDE Platform, Integration Framework for the SIMARGL system), including the requirements of the Lambda architecture, its interfaces and their interaction. This document is partially based on D2.2: SIMARGL requirements and use cases derivable and D2.3: First version of the SIMARGL toolkit architecture deliverable (completed at M6).

1.3 Relation to other deliverables

Current document scope is related to:

- D2.2 SIMARGL requirements and use cases (M4) where formalized and prioritized requirements and use-cases have been presented.
- D2.3 First version of the SIMRARGL toolkit architecture (M6) that served as the initial architecture specification refined in the current document.

Also, this document will impact all future WP4 deliverables (M12-M26) providing solutions to detect and counter malware and stegomalware that will constitute the SIMARGL toolkit designed here, and in particular, WP5 deliverables in which the tools will be integrated into the toolkit (D5.3 SIMARGL service toolkit is scheduled at M34).

2. Logical architecture

In this section, we describe the logical layered architecture used to implement the SIMARGL toolkit. To design the different functional entities composing the SIMARGL framework, we separated functionalities in coherent groups. Figure 1 depicts the general architectural blueprint considered in this project.

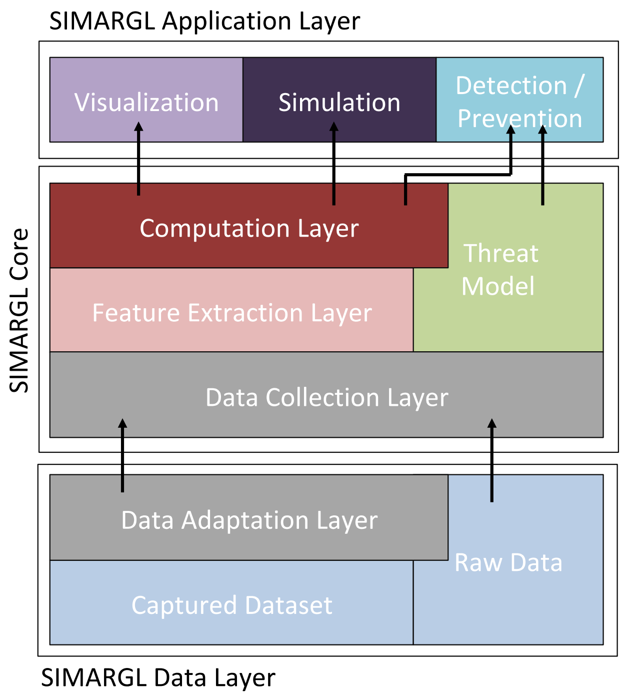


Figure 1: General layered architecture for the SIMARGL toolkit.

As shown, the SIMARGL toolkit is subdivided into three main blocks. Specifically:

- **Application Layer**: it contains all the applications that take advantage of services provided by the SIMARGL platform.
- **SIMARGL Core**: it contains the services responsible for processing data and performing the needed computation.
- **Data Layer**: it is responsible for merging and storing the needed data in a unified format. Optionally, it can also implement some capturing services or perform minimal computation, for instance for filtering purposes.

According to such design, the Application Layer will interact with the SIMARGL Core via a suitable Application Programming Interface (API). For instance, an application could require the toolkit to check for a signature of a malware or to perform some simulations to evaluate the effectiveness of a model. We point out that an application leveraging the services offered by SIMARGL could be also distributed or implemented according to a multi-tier approach.
Therefore, the SIMARGL Core can be described as a sort of "service provider", which can be remotely deployed or collocated within the same node of the application. Possible applications that can be developed by using the SIMARGL toolkit are:

- **Visualization**: used for threat intelligence or for training purposes;
- **Simulation**: used for planning, evaluating how an attack can spread through the network, impact over a specific environment, etc.;
- **Detection / Prevention**: used to feed other detection frameworks, for instance, by providing proper signatures or rules for sanitization/blocking of malicious network traffic.

As regards the SIMARGL Core, it is not only important for implementing applications. The other essential role is its ability to interact with data sources. Roughly, the SIMARGL Core retrieves data, feeds machine learning algorithms and exports the obtained knowledge to the application layer through the API. In essence, the SIMARGL Core is the part of the architecture containing the services responsible for processing data, perform machine-learning-capable computation, apply or infer the threat model according to the provided data or a usage template, condense and extract features from the samples provided by the SIMARGL Data Layer. The latter is responsible for gathering information from the different probes/nodes/sources placed in the network to acquire stimuli and measure the traffic. Therefore, SIMARGL Core is composed of the following layers:

- **Data Collection Layer**: it is responsible of interacting with the SIMARGL Data Layer acting as the source for information of the entire layered architecture. This Data Collection Layer could be a complex platform managing streaming data or a very simple connector, i.e., it depends on the specific application of SIMARGL toolkit and its complexity varies according to the rest of the architecture and the “richness” of the data to be handled.
- **Feature Extraction Layer**: it is devoted to process data for extracting features needed to model or detect the threat. In the case of stegomalware, it can be also used to synthesize high-level indicators or specific signatures able to detect the presence of a hidden flow within the network traffic or the footprint of a hidden-data-exchange between different components of the malware.
- **Threat Model**: it contains information describing the (class of) threats that are addressed by the SIMARGL core. It can interact with data available in the toolkit and also with the feature extraction layer. The threat model is presented here as a standalone entity as to emphasize the “open” nature of the SIMARGL toolkit and it can be updated to consider new threats. However, it can be also a simple block, e.g., a set of constraints to be considered when optimizing a mathematical model or a training of some form of artificial intelligence or statistical method.
- **Computation Layer**: it is in charge of computing data. It takes advantage of the information processed through the architecture to provide outputs that can be used by the application. This layer can be deployed on different architectural flavours, according to the complexity to be addressed or the need of satisfying some real-time constraints. Thus, it can run on a cloud platform, be deployed via containers or in a standalone manner.

The SIMARGL Core can be also implemented in a distributed manner, i.e., it can be deployed on many nodes to increase its computing capability, provide redundancy and scalability, or it can be hosted by different organizations each one responsible to feed the toolkit with their information.

Lastly, the SIMARGL Data Layer is responsible for merging and storing the data acquired throughout different sources. For instance, data can be collected from the network, virtualized execution environment, or imported from some pre-existent databases. Thus, the role of this layer is to present such information to the rest of the SIMARGL architecture in a unique and coherent manner. In general, a flexible data layer should contain the following functional layers:

- **Data Adaption Layer**: it is responsible of routing data to the SIMARGL Core in a unique and coherent manner.
- **Captured Dataset**: it is in charge of storing datasets used by the toolkit. Data can be stored in a database, provided by streaming services (e.g., network probes) or fetched in an asynchronous manner.
- **Raw Data**: it is devoted to store and route data without any processing. This feature can be useful to guarantee compatibility or to have enough flexibility of processing data without any functional overheads or pre-processing.

Similar to the previous layers, also the SIMARGL Data Layer could be distributed. This is the case of having multiple probes placed in different parts of the network (e.g., in core routers, border routers or firewalls) feeding the rest of the toolkit. Besides, the Captured Dataset layer could range from a simple data wrapper to several layers to encapsulate functionalities implemented by network probes.

As a possible example on how the SIMARGL architecture can be used in a real, distributed and complex environment, Figure 2 depicts a possible use case enlightening both the potential and the flexibility of the SIMARGL toolkit to match production-quality deployments.

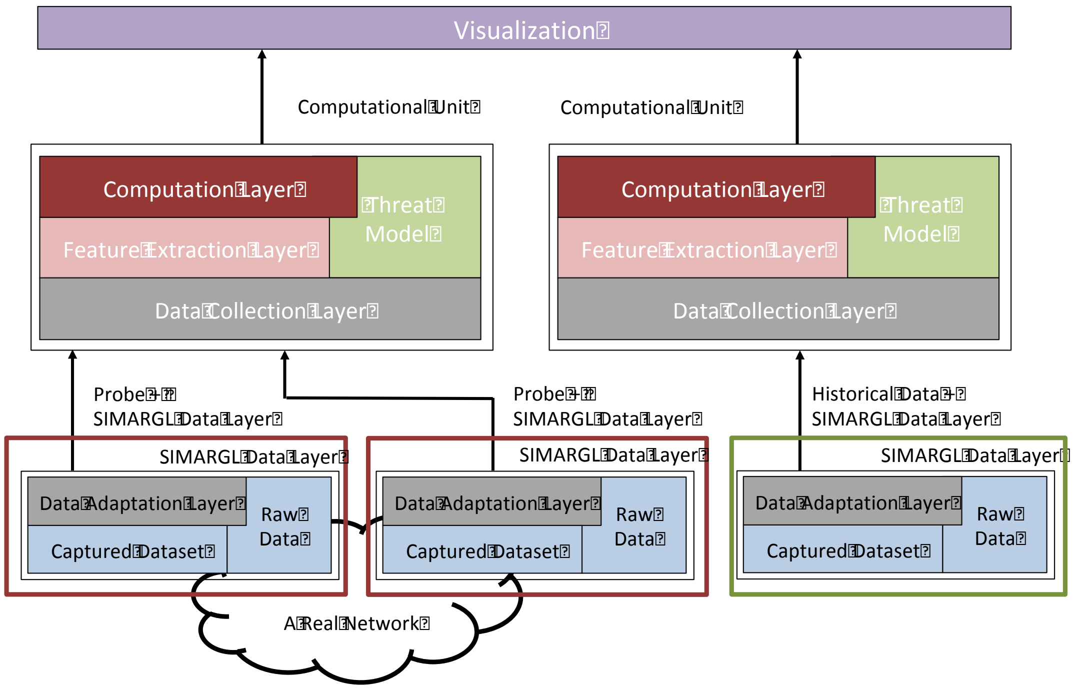


Figure 2: Example of the deployment of the SIMARGL toolkit in a realistic use case.

As shown, the SIMARGL architecture is general enough to match complex scenarios. As regards the SIMARGL Data Layers, different types of implementation are shown. One stack has been adapted to act as a probe and deployed in two portions of the network, whereas the other is used to feed the architecture with historical data. Consequently, the degrees of sophistication of the two variants are very different. In the first case, different layers are needed to encapsulate the functionalities of the probes or to implement a suitable data wrapper and importer. Instead, in the second case, the SIMARGL Data Layer could reduce to a simple data connector or database driver.
Concerning the computational units, that can be deployed in different portion of the managed system/deployment and also run in a multi-tenant flavour.
The application (a visualization service, in this example) can interact with the different Computational Units by using a suitable API.

Lastly, we showcase a simpler use case, which demonstrates the flexibility of the architecture at the basis of the SIMARGL toolkit. Figure 3 depicts an on-line scanning service implemented by using SIMARGL.

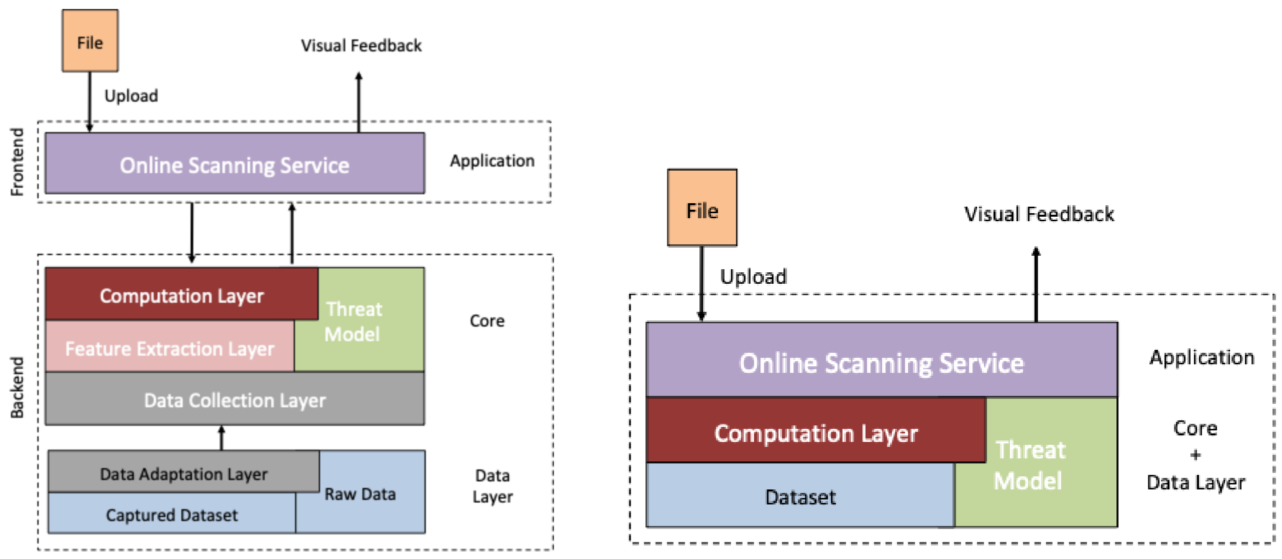


Figure 3: The SIMARGL layered architecture implementing an online scanning service.

Specifically, we consider a service with the following life-cycle: *i*) the user uploads a file to the service to check the presence of malware/stegomalware, *ii*) the service performs the scan; *iii*) a visual feedback is provided back to the user. The architecture depicted in Figure 3.a implements the service by using all the components envisaged for SIMARGL. Thus, the file is handled by the application and analysed in a backend implementing the proper computational aspects and storing the signatures or the rules needed to evaluate the presence of a threat. However, this architecture could be too complex for the service to be delivered. Therefore, Figure 3.b shows a simpler implementation where all the unneeded layers have been removed or merged. This is, for instance, the case of the data layer. In fact, when in the presence of simple static signatures without the need of any real-time data acquisition or processing, it reduces to a simple I/O interface towards a dataset.

3. Technological architecture

In order to efficiently implement the functionalities of all layers composing the logical architecture, we envision the technology stack presented in Figure 4. We have depicted three crucial elements:

- deployment environment (a cluster of physical and virtual machines),
- set of services deployed on top of it,
- and a communication data bus (Apache Kafka).

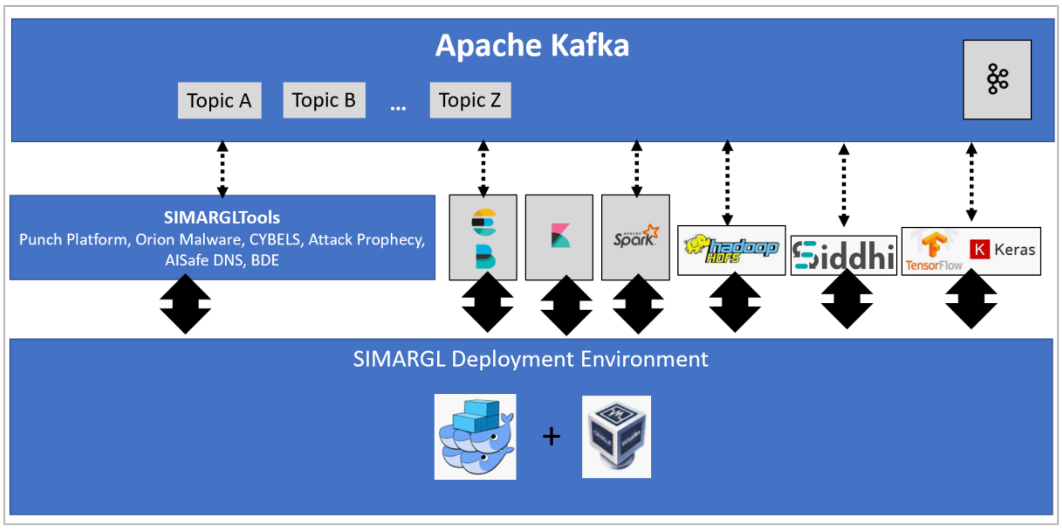
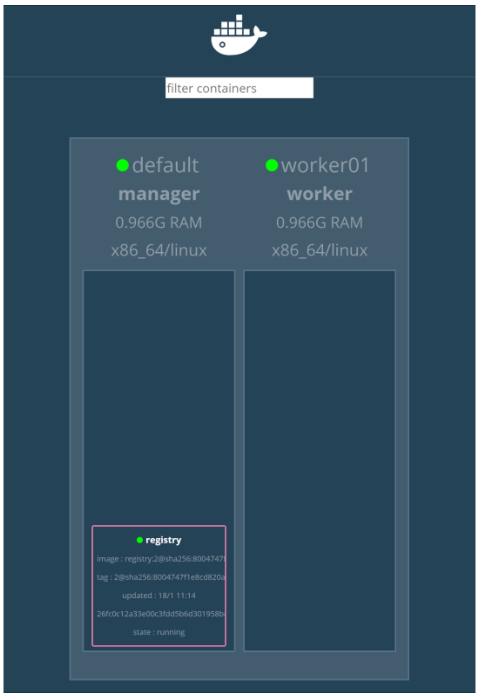

Apache Kafka
Topic A
Topic B
...
Topic Z
SIMARGLTools
Punch Platform, Orion Malware, CYBELS, Attack Prophecy, AISafe DNS, BDE
Spark
hadoop HDFS
Siddhi
TensorFlow
Keras
SIMARGL Deployment Environment
+

Figure 4 Technology stack adapted for SIMARGL platform

3.1 Deployment environment layer

The first and the most bottom layer in the technology stack constitutes the orchestration framework. It is laid down on top of an infrastructure composed of virtual and hardware machines. This layer is intended to implement automated resource management and thus facilitate the entire platform with such capabilities as flexibility, scalability, and fault tolerance. It is the responsibility of the orchestration layer to effectively deploy the services on the available computational nodes (Both physical and virtual). It is achieved thanks to containers that are sort of sandboxes that contain the implemented service together with all the software dependencies (libraries and execution environment). In such a form the services can be easily migrated between the computational nodes and deployed.
In the proposed architecture we decided to use Docker Swarm framework. In fact, we have considered Kubernetes as well. These two are competing solutions and although they allow the user to achieve the same goal, internally they are different. Moreover, there are advantages and disadvantages of using either Kubernetes or Docker Swarm. Although Kubernetes has substantially bigger community of user at the same time it has steeper learning curve. On the other hand, Docker Swarm can be easily and quickly set-up on a wide range of most popular operating systems (Linux, Windows, MacOS). Together with an intuitive collection of tools for cluster management, especially for those who are already familiar with Docker, all the environment happens to be flexible, user-friendly, and easy to maintain. Moreover, Docker Swarm comes with visualisation tools for cluster monitoring (see Docker Swarm Visualizer in Figure 5). This kind of tool allows for quick investigation of the cluster's health, the number of services running, the amount of computational rescues consumed (e.g. RAM memory), etc.



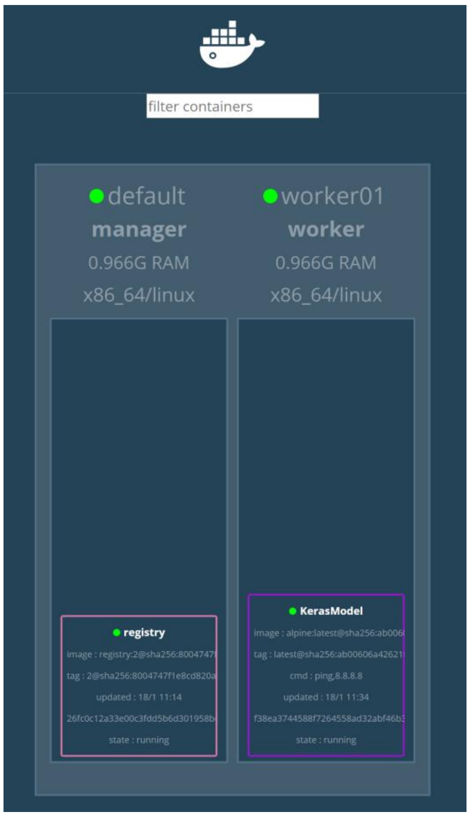
default
manager
0.966G RAM
x86_64/linux

worker01
worker
0.966G RAM
x86_64/linux

registry
image : registry:2@sha256:8004747f
tag : 2@sha256:8004747f1e8cd820a
updated : 18/1 11:14
26fc0c12a33e00c3fdd5b6d301958b
state : running

Figure 5 Two-nodes cluster visualised in Docker Swarm Visualizer tool.

Using single command-line sentence user may easily scale-up and scale-down the system, as it has been shown in Figure 6. In the provided example, user replicates KerasModel service. As the replicas are automatically distributed over two computational nodes. As a result, the entire system can now handle more traffic/data than before.

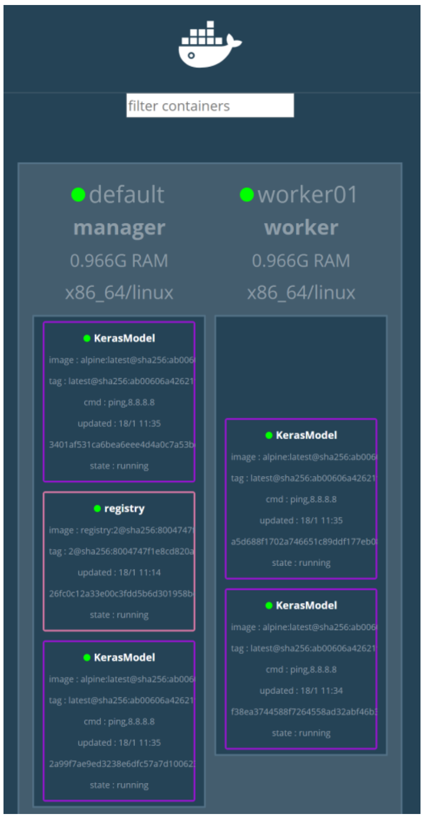


default
manager
0.966G RAM
x86_64/linux

worker01
worker
0.966G RAM
x86_64/linux

registry
image : registry:2@sha256:8004747f
tag : 2@sha256:8004747f1e8cd820a
updated : 18/1 11:14
26fc0c12a33e00c3fdd5b6d301958b
state : running

KerasModel
image : alpine:latest@sha256:ab006
tag : latest@sha256:ab00606a42621
cmd : ping,8.8.8.8
updated : 18/1 11:34
f38ea3744588f7264558ad32abf46b3
state : running



default
manager
0.966G RAM
x86_64/linux

worker01
worker
0.966G RAM
x86_64/linux

KerasModel
image : alpine:latest@sha256:ab006
tag : latest@sha256:ab00606a42621
cmd : ping,8.8.8.8
updated : 18/1 11:35
3401af531ca6bea6eee4d4a0c7a53b
state : running

registry
image : registry:2@sha256:8004747f
tag : 2@sha256:8004747f1e8cd820a
updated : 18/1 11:14
26fc0c12a33e00c3fdd5b6d301958b
state : running

KerasModel
image : alpine:latest@sha256:ab006
tag : latest@sha256:ab00606a42621
cmd : ping,8.8.8.8
updated : 18/1 11:35
2a99f7ae9ed3238e6dfc57a7d10062
state : running

KerasModel
image : alpine:latest@sha256:ab006
tag : latest@sha256:ab00606a42621
cmd : ping,8.8.8.8
updated : 18/1 11:35
a5d688f1702a746651c89ddf177eb0
state : running

KerasModel
image : alpine:latest@sha256:ab006
tag : latest@sha256:ab00606a42621
cmd : ping,8.8.8.8
updated : 18/1 11:34
f38ea3744588f7264558ad32abf46b3
state : running

Figure 6 Example of replicating KerasModel service.

3.2 Apache Kafka communication data bus

We plan to facilitate high-throughput, low latency, and fault tolerance of the data ingestion capabilities using Apache Kafka. It is a distributed streaming platform, and enables both real-time event processing and event-driven communication between various components.

From the architectural point of view, Apache Kafka constitutes a flexible and efficient way to integrate all components of SIMARGL toolkit, both existing tools provided by consortium, and new ones developed during the project lifetime.

Moreover, thanks to the additional components, such as Kafka Connect, it is easy to efficiently transport the data from raw format to various data sinks (e.g. HDFS, Elasticsearch, MongoDB, etc.). It is of high relevance, when it comes to efficient data processing. For instance, to store large flat files of various types, HDFS (Distributed Hadoop File System) can be employed. Moreover, the same raw data can be stored in several databases at the same time, so that it is possible to leverage the strengths of various data searching engines. For example, structured data (e.g. NetFlow) can be stored in Elasticsearch to enable full-text search capabilities (and visualisation via Kibana) and to HDFS to run various analyses using batch processing.

3.3 Computational services layer

The key building blocks of this layer constitute a microservice. It is an independently deployable component, which (as it was mentioned before) typically is packed as a Docker container, but in fact this is one of the options for deployment. As it is shown in Figure 7, the key elements building the microservice components are:

- API that allows other clients to interact with the service in synchronous (e.g. REST) and asynchronous ways (e.g. events).
- Client API for interacting with other components/services.
- Event subscriber (listener) that allows the service to retrieve notification from other services.
- Internal and private storage that maintains all relevant data required for serving the purpose of the microservice.

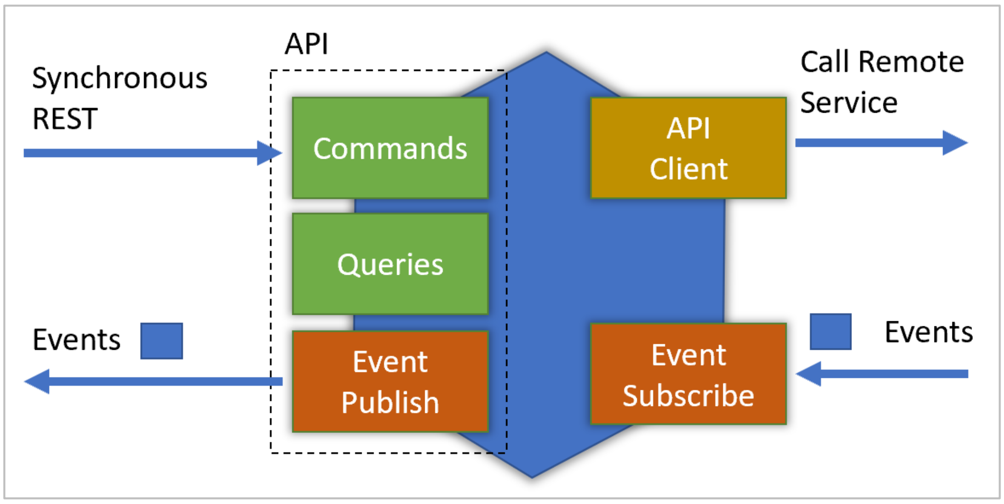


Figure 7 General architectural outline of a microservice.

The synchronous API calls can be essentially divided into commands and queries types. The command is a type of a remote invocation that internally mutate the data of the service. For instance, service may expose methods which add, update or delete some data (e.g. method addToTheBlockingList(IP)). Another type of API calls constitutes queries, that can be essentially used to find specific data using various search criteria. Such calls do not mutate internal data.

On the diagram presenting the SIMARGL framework architecture there are several pre-defined services (and microservices) that either will support or provide dedicated system functionality. These are:

- Elasticsearch database
- Kibana visualisation
- Applications using Tensorflow or Keras ML libraries
- Siddhi complex event processing
- Apache Spark

From all of the elements on the list we envision Apache Spark as one of the elements that will enable efficient data processing. It provides an engine that processes big data workloads. Apache Spark uses the data abstraction called resilient distributed dataset (RDD) to manage the data in a distributed way in the cluster and to implement fault-tolerance. Moreover, it has a dedicated SparkML library that implements various machine learning algorithms in a distributed way, that can work both with data batches and streams. The intermediate (or final) results of the analysis can be sent back from Apache Spark to Apache Kafka in order to make the new data (or enriched/augmented one) available to other applications for further analysis.

4. Interfaces and interactions of SIMARGL platform

The SIMARGL framework will provide a common look-and-feel through a portal-like user interface to guide the user to the underlying functionality of the internal components. A portal-like approach is common practice for integrating distinct functional components and providing an integrated presentation layer.

4.1 API Gateway

When the application is broken down into a set of separate services, eventually it happens that these need to communicate in order to provide complex business capabilities. That capability usually needs to assemble the results obtained from multiple services. The aggregation could be performed in different ways and one of the most popular and widely used is the API Gateway pattern (see Figure 8).
In general, the gateway can be seen as a reverse proxy, which is used by services that reside in the backend (are hidden behind the reverse proxy). It takes requests from the client and forwards these requests to one of the backend services. There are several advantages of using the API Gateway pattern. Firstly, it constitutes a single entry point for any call. This, for instance, allows for implementing the authorisation functionalities at the gateway. Secondly, the gateway can translate the request protocol to something else such as AMQP (Advanced Message Queuing Protocol) or Apache Kafka messages. Thirdly, the gateway can proxy request from the client to multiple services and aggregate results.

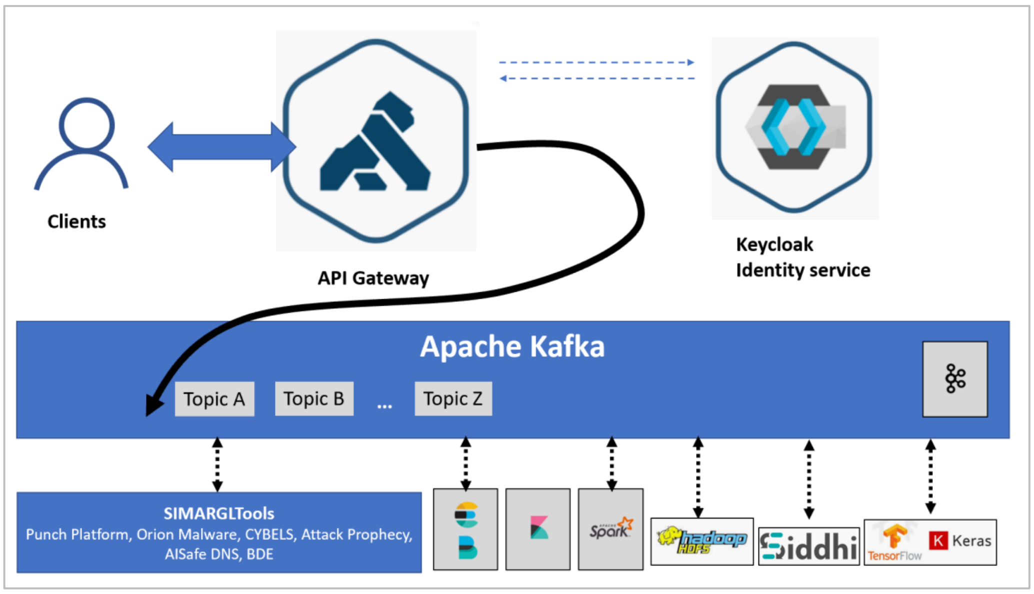


Figure 8 API Gateway approach unifying separate interfaces of all components.

4.2 Threat intelligence Incidents Management GUI

SIMARGL will provide a specialized dashboard-based user interface for operational management of threat intelligence, displaying incident managed by the current user. For each incident the current task will be highlighted and possibly follow up actions will be indicated, allowing the user to make decisions. Predefined charts will also be available for providing an overview and organizing information about incidents.
The dashboard will be fully configurable for each user, becoming a one-stop workplace for analysts to see their workload information and be able to take actions. Having scalability in mind, we will leverage computational capabilities of the SIMARGL platform. For example, one of the usage scenarios can be the adoption of Apache Kafka, and Elastic (ELK) Stack for NetFlow data processing. Using Kafka Connect to harvest the NetFlow data it can be easily moved (through the Apache Kafka) to Elasticsearch database, on top of each Kibana dashboard can be quickly deployed (see Figure 9).

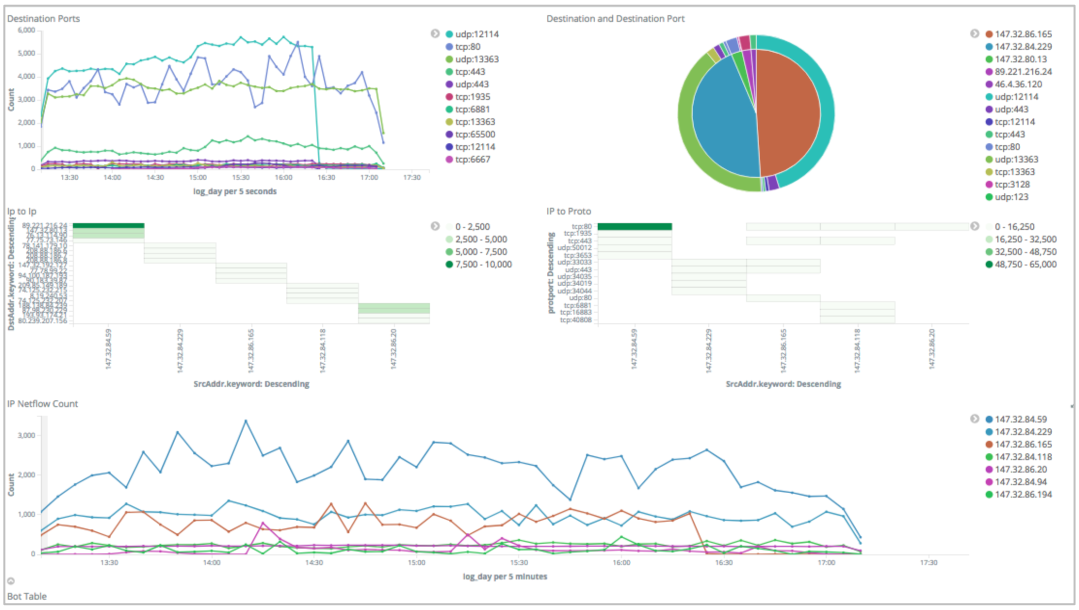
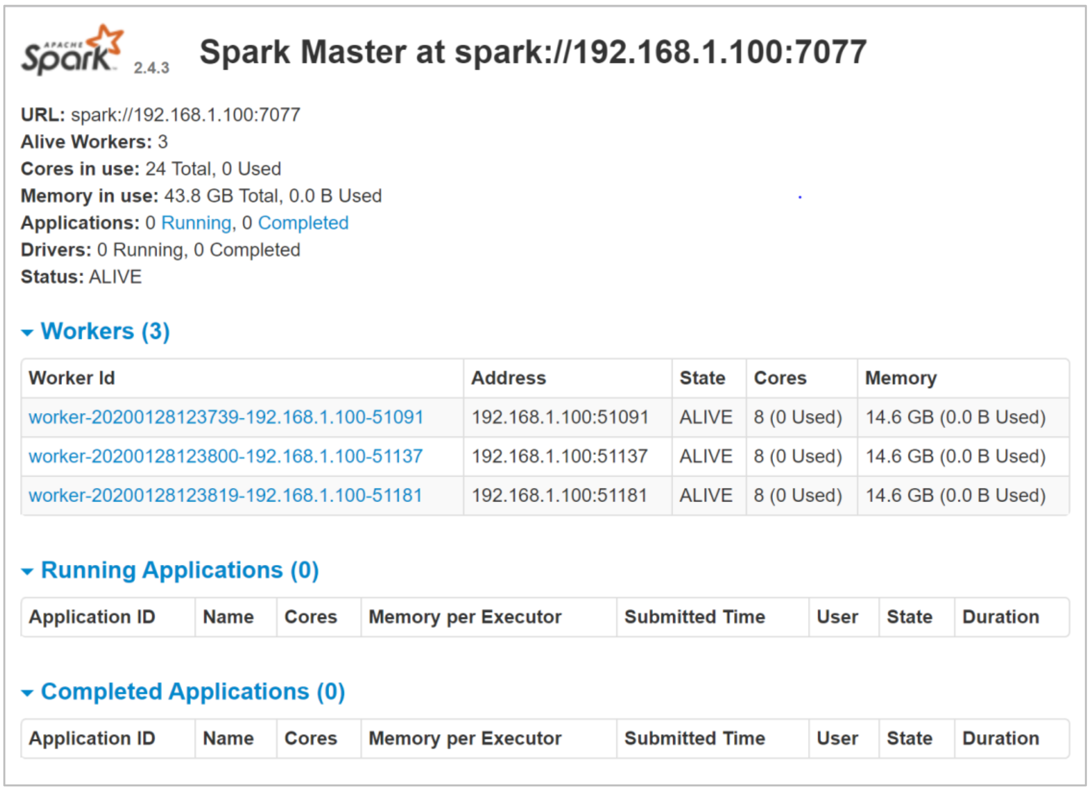


Figure 9 Example of NetFlow data indexed in Elasticsearch database and visualised in Kibana.

4.3 System Administration GUI

SIMARGL will provide a specialized administrator interface for managing platform, as well as monitoring system activities and conducting logs audits. We plan to compose it from existing management consoles that are available for all the tools we are going to adapt in the platform. For instance, the administrator will be able to manage the Apache Spark cluster (see Figure 10).
We will also consider providing advanced searching and filtering capabilities, such as viewing and searching custom log properties, automatic color-coding to draw attention to error and warnings, detailed traces on end-user operations, tracking where log messages originated in application code, as well as real-time log tailing with watch filters for monitoring events.



Spark Master at spark://192.168.1.100:7077

URL: spark://192.168.1.100:7077
Alive Workers: 3
Cores in use: 24 Total, 0 Used
Memory in use: 43.8 GB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory
worker-20200128123739-192.168.1.100-51091	192.168.1.100:51091	ALIVE	8 (0 Used)	14.6 GB (0.0 B Used)
worker-20200128123800-192.168.1.100-51137	192.168.1.100:51137	ALIVE	8 (0 Used)	14.6 GB (0.0 B Used)
worker-20200128123819-192.168.1.100-51181	192.168.1.100:51181	ALIVE	8 (0 Used)	14.6 GB (0.0 B Used)

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration

Figure 10 Monitoring and management console provided by Apache Spark framework.

5. Authorization and Access Control

Commonly, the **authentication** is carried out starting from the user that logs in. Typically, the approach is described as follows:

- Security module verifies user credentials and creates a session containing information such as its name, roles and permissions/constraints.
- The session is then returned to the client and recorded at the user side (e.g. as a cookie when web interface is considered).

As for the **authorization**, in various scenarios, it is carried out by the security module. It happens each time the backend service is interacted and commonly it includes the following aspects:

- Requests of particular actions are sent to a security module alongside security information,

- Security information (carried out in the request) is verified,
- In the next steps, a user's rights are checked to confirm whether the request can be executed or not.

For the SIMARGL architecture we have proposed adapting Keycloak identity service. Keycloak is an open-source Identity and Access Management system. It gives the user all kinds of mechanism to secure a broad spectrum of backend services and applications. The Keycloak enables capabilities such as:

- Single Sign-On
- Identity Brokering and Social Login
- User Federation
- Client Adapters
- Standard Protocols
- Admin Console
- User Account Management Console

There are several scenarios of adapting Keycloak server:

- when an unauthenticated human client (which wants to access a particular backend application) will be forced to first access the Keycloak login page. After a successful login the client is brought back to the original application with a security token. In that scenario the backend application just needs to confirm the token is valid.
- when an internal service (without interaction with human operator) will call API request on another service that requires the caller to be authenticated. In such a situation, service will securely store credentials that will be associated with that service (not the human user) and perform authentication via HTTP(S) channel.
- when internal service requires user credentials to call API request on another services. In such scenario authorisation is delegated to Keycloak and (when successful) a security token is returned. The service uses this token to call other services on behalf of user.

In order to avoid manual work required to create users accounts, we consider adapting Keycloak as an integration platform used in one of the following scenarios:

- connect Keycloak into existing LDAP and Active Directory servers,
- use Kerberos bridge to automatically authenticate users that are logged-in to a Kerberos server,
- use ”Social Login” feature to Enable login with Google, GitHub, Facebook, Twitter, and other social networks.

However, one can also point to cases where none of these options will be possible to deploy for some reasons (e.g. security, legal or organisational). Therefore, the backup solution will be to manually create users accounts using Keycloak web console.

For user authorisation we plan to adapt RBAC (Role-Based Access Control) model. In that regard, each user will be assigned to a role that will directly map to specific privileges that user will have (e.g. guest, resource owner, admin, etc.). More details on user-role mapping have been shown in Figure 12. In general, the model has a graph-like structure with the key node called a Realm. It maintains the set of users, credentials, roles, and clients. In the RBAC model, Roles play an important part in access authorisation because applications often apply permissions and access rights based on roles rather than an individual user's name. Roles can be assigned at a level of Real or Client, which is an entity that could be related to a specific access channel (e.g. mobile device, web-based application, desktop application, etc.).

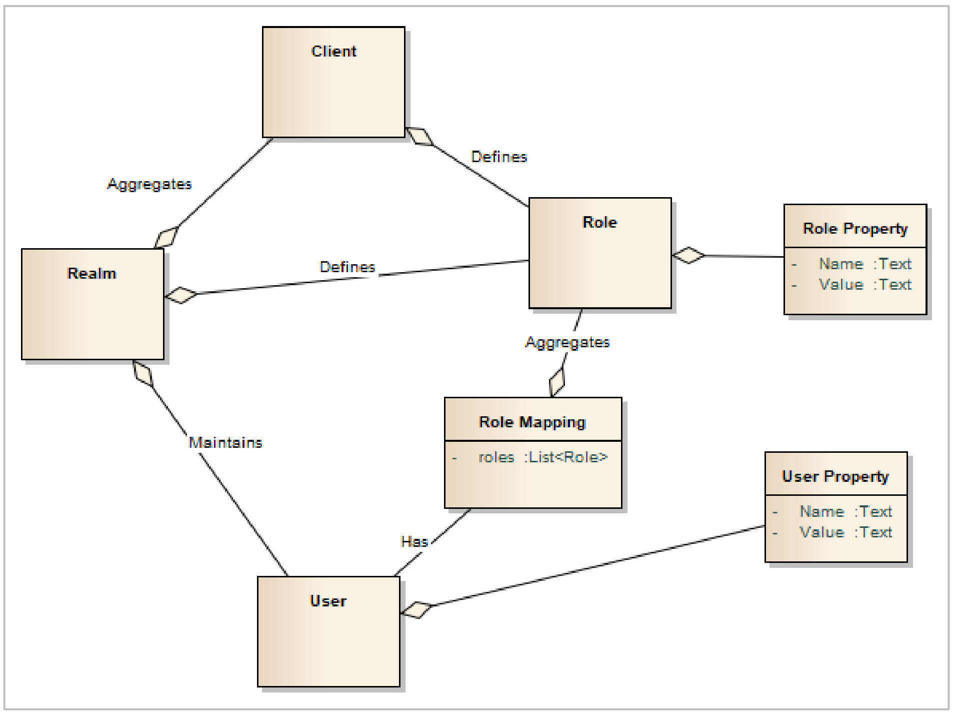


Figure 12 The data model describing users, their roles, and properties.

6. Logging and health monitoring

There relevant information allowing to track the health of separate modules will be most likely kept in form of logs locally at key elements of the SIMARGL platform. In order to provide the administrator with the big picture the logfiles scattered across the network elements, these need to be collected in one place for the analysis. At this stage of development of SIMARGL platform, the most suitable solution enabling logging and health capabilities is the Elastic stack[1]. It consists of four elements:

- Beats – a sensor retrieving and capturing the operational data (the logfiles)
- Logstash – an entity ingesting the data into the Elasticsearch engine
- Elasticsearch – a data storage and indexation engine
- Kibana – a visualisation frontend for advanced analysis of the ingested data

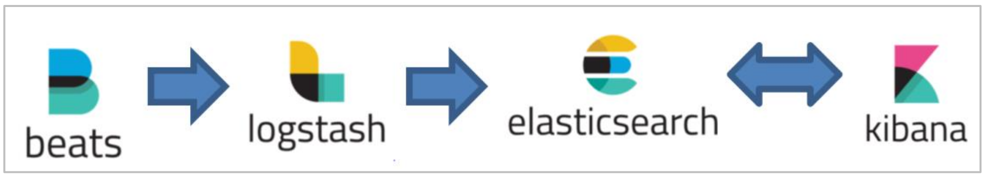


Figure 13 Elastic stack.

With Logstash we enrich and transport data sources. It is an open source data collection engine with real-time pipelining capabilities. Logstash will dynamically unify data from disparate sources and normalize the data into destinations. Moreover, Logstash welcomes data of all shapes and sizes:

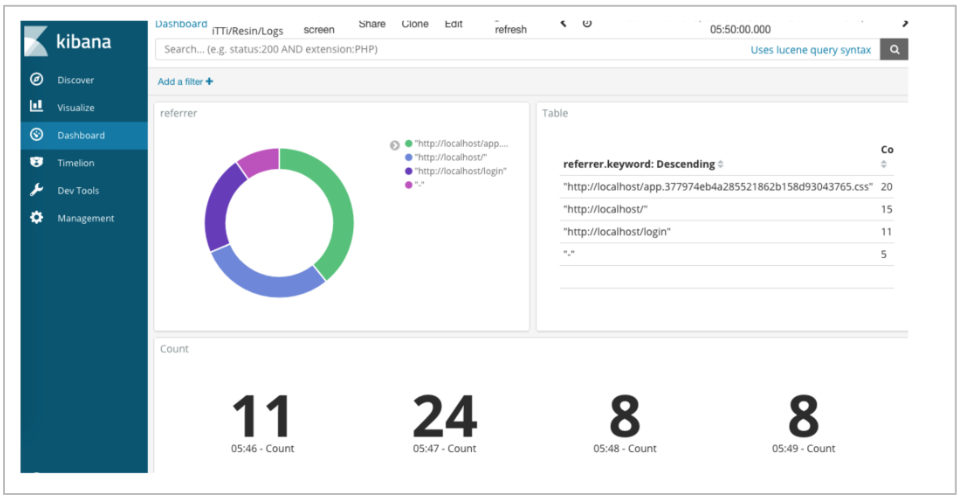
- Logs and metrics (logging data, application logs, many other log formats like networking and firewall logs and other)
- The Web:
    - Transform HTTP requests into events
    - Create events by polling HTTP endpoints on demand
- Data Stores and Streams

[1] https://www.elastic.co/elk-stack

- Sensors

Elasticsearch allows for full-text searches and enables variety of data analysis capabilities (aggregation, filtering, projection, etc.). Kibana is used to visualize and explore the data.

An example of Kibana dashboard produced for logfile data analysis has been shown in Figure 15. It can be easily customised at runtime, but in the given example it allows for visualising the name of the resources that have been accessed by the user (the pie chart on the left) and the number of request within the selected timespan (the bottom row).



referrer.keyword: Descending	Co
"http://localhost/app.377974eb4a285521862b158d93043765.css"	20
"http://localhost/"	15
"http://localhost/login"	11
"-"	5

11
05:46 - Count

24
05:47 - Count

8
05:48 - Count

8
05:49 - Count

Figure 15 Example of Kibana dashboard – logs visualisation.

Kibana dashboard will allow us to summarize the events from all components where data input was sent through Core Management system (Figure 11). When the alert on your summary alert dashboard indicates a problem, you want to be able to drill down to the underlying metrics dashboard, even having the possibility to analyse deeply and link to the original component dashboard that generated it.

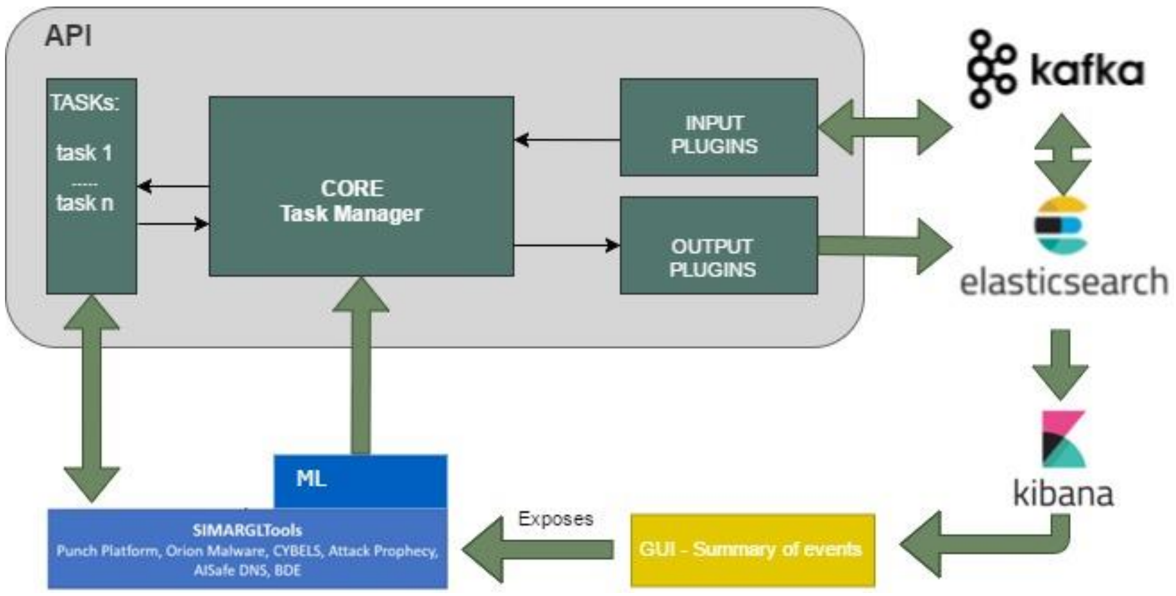


Figure 11 Data flow; links to components.

7. Conclusions

In this deliverable, we have specified formal SIMARGL architecture with the final specification of particular components, dependencies among them and interfaces needed to integrate all functional blocks into the interoperable SIMARGL toolkit. In particular, we have described logical architecture and technological views. In that regard, we have explained the deployment environment and highlighted the key technological choices. We decided to use Docker Swarm for the containers orchestration, Apache Kafka as event/message data bus, and various additional frameworks that will support computational capabilities (e.g. Tensorflow, Apache Spark, etc.). Moreover, this document also elaborates on matters related to authorisation, authentication, logging and health monitoring. In that regard, we envisioned the adaptation of Keycloak identity service in order to ease the management of RBAC access control model and proposed ELK (Elasticsearch + Logstash + Kibana) stack for log analysis.

Annex A: Functional components of the SIMARGL toolkit

The SIMARGL toolkit will contain multiple independent components. This section presents the SIMARGL's components and the Lambda architecture requirements.

Annex A1: Punch Platform (Thales)

Thales's Punch Platform is a component implementing a lambda-based architecture for moving and processing a vast amount of data in different environments. It is scalable by design, as it is based on processes than can be dispatched on as many computers as needed to provide the capacity to manage its input workload.
The Punch Platform can be seen as an enabler: it provides capacities and their orchestration through pipelines. It is an infrastructure that hosts the processes in charge of performing the computations.

Annex A1.1: Punch Platform capacities

By providing the capacities shown in Figure 12, the Punch Platform helps teams to build effective solutions in different contexts:

- Log analytics.
- Full-text search.
- Security intelligence.
- Business analytics.
- Operational intelligence.

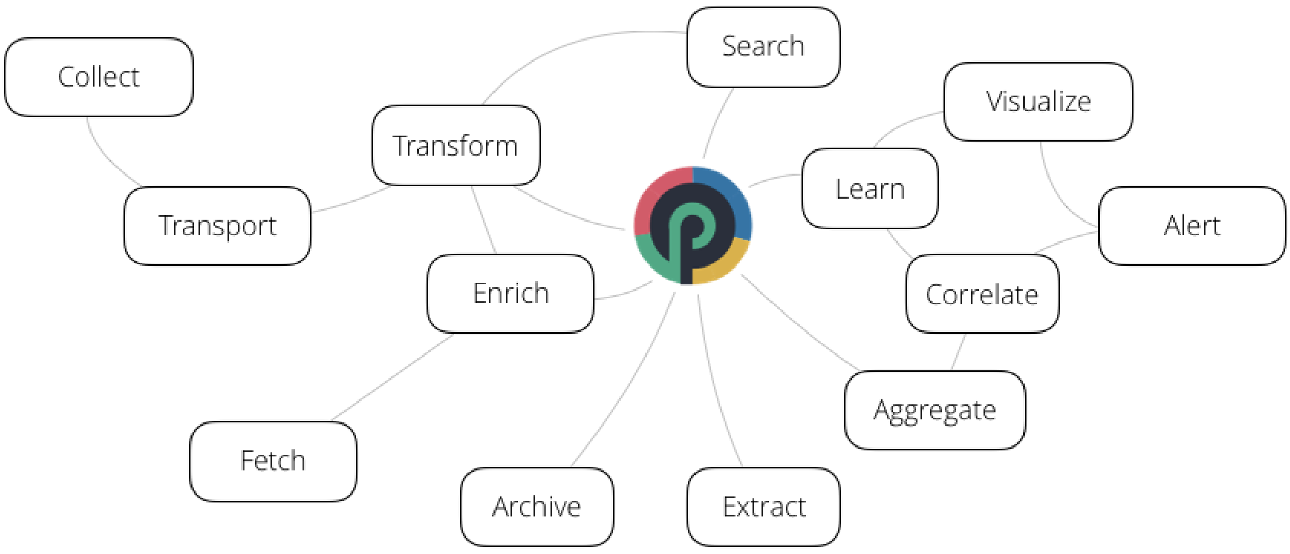


Figure 12 Punch Platform capacities

In the SIMARGL context, the Punch Platform can be used to gather information from different sources (network probes, logs, and cyber threat intelligence sources). It provides the required information to processes integrated into pipelines, which can further format, transform and enrich data. Moreover, it makes correlations and is able to train and execute the model against the data.
As the Punch Platform is highly configurable, it can also be used to access existing data stored in different databases, either to use it to enrich data it receives, or to perform treatments on those data. For example, machine learning batch processes can use information to look for threats or attacks that were not known when the data was stored or use it to train a certain model.

One key feature of the Punch Platform is its scalability. It can be used on environments of different capacities:

- Edge: refers to small platforms with a number of nodes/servers ranging from one to ten. The edge runs on limited hardware resources yet provides advanced stream processing features, potentially including machine learning capabilities.
- Data Center: refers to large-scale on-premise platforms composed of up to several hundred of machines. Besides the size and resources, such servers are managed by human operators on a daily basis. Although running critical services, they must be easily upgraded, adapted, and enriched with new services.
- SOCs: refers to Security Operation Center. See a SOC as a DataCenter-like platform focused on providing cybersecurity detection and forensic capabilities. Some SOCs, however, require running on limited hardware to monitor small systems.
- Clouds: refers to cloud and cloud-native environments where applications and resources need to be monitored, e.g., for security concerns or for tracking resources and application usages. Most often, clouds environments are required to track resources, applications and security simultaneously.

If the Punch Platform will be used in the experimentation phase in an Edge configuration, nodes could be added later to exploit the results on site at full scale.

When building SOCs, most efforts are usually devoted to design data processing applications to parse, filter, route, and enrich data. It often requires aggregating, detecting, and reprocessing data. The Punch Platform environment comes with a dataflow programming tool to design such data processing applications, using a model-driven approach. Each application is represented as a set of interconnected pipelines, used to design processes workflow.

Each pipeline can use several processing nodes. The simplest pipeline allows designing input-processing-output patterns, such as the one needed for a log management solution. Much more sophisticated models can be defined using the same tool to fit different needs. The Punch Platform provides ready to use nodes, and supports user defined ones.

The Punch Platform provides the following processors:

- Storm: runs pipelines referred to as topologies.
- Spark: runs batch pipelines, in particular the ones running machine learning modules.
- Spark Streaming: runs continuous pipelines, in particular the ones running machine learning modules.
- Punch: provides a lightweight pipeline engine. It offers a minimal yet performant single-process engine to run pipelines on constrained environments such as edge platforms. It is also well-suited to deploy pipelines on dockerized platforms.

Moreover, Python 3 integration allows to exploit Python-defined processes. It also includes access to the Keras toolbox to use Deep Learning algorithms.

The Punch Platform also provides a Complex Event Processing (CEP) capability. It can be used to create on-the-fly analytics' information or correlation rules. This capability is of particular interest in a machine learning context, because it allows the creation of data based not only of data's content, but also on flow's behaviour. For example, by using CEP rules, it is easy to count the event rate for a particular kind of equipment: maybe the number of events per second is an indication of interest.

Annex A1.2: Production-grade features

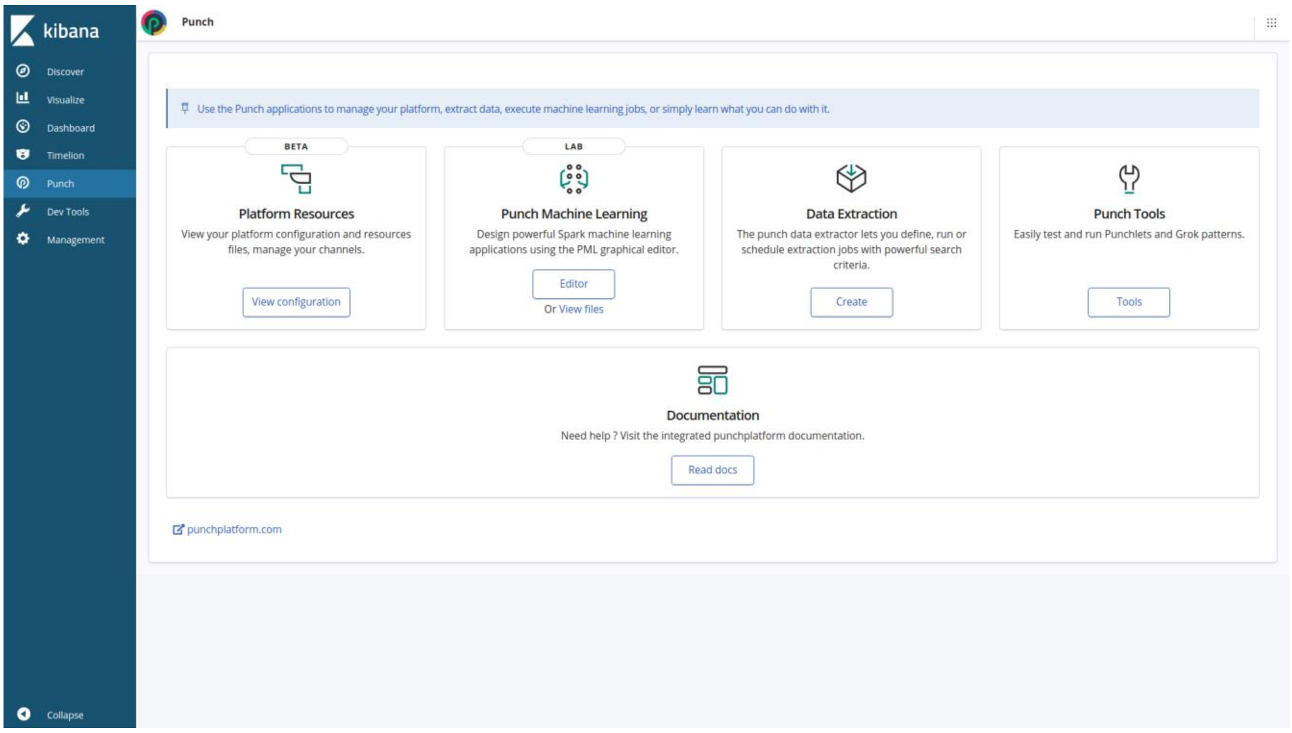
As the Punch Platform is a carrier-grade product, it takes into account real-life aspects of a production environment. The Punch Platform monitors its components because if problems cannot be seen, they cannot be managed. Since the Punch Platform is mainly used for critical tasks, its health must be controlled and managed. To this aim, it automatically monitors its inner components, and keeps an updated global health status, which is exposed using a REST API. Information on the status of the Punch Platform can be integrated into an existing supervision dashboard to periodically report any problems that may occur.

Another critical point of any production environment concerns the upgrade management. Updating a platform encompasses both the platform itself and the business components. Punch Platform update management relies on various features: centralized configuration management, deployment tooling, dynamic monitoring and functional component deployments, etc.

The Punch Platform can be configured to keep data online and indexed as long as needed, but it as a cost in terms of computing power and memory. Data can be archived on a secondary storage unit in order to comply with needs in a cost-effective manner.

Annex A1.3: Graphical interface

As seen in Figure 13, the Punch Platform comes with a graphic interface to help users to prepare and test their configurations. The console is a plugin added to the Kibana Web interface.



Use the Punch applications to manage your platform, extract data, execute machine learning jobs, or simply learn what you can do with it.

View your platform configuration and resources files, manage your channels.

Design powerful Spark machine learning applications using the PML graphical editor.

The punch data extractor lets you define, run or schedule extraction jobs with powerful search criteria.

Easily test and run Punchlets and Grok patterns.

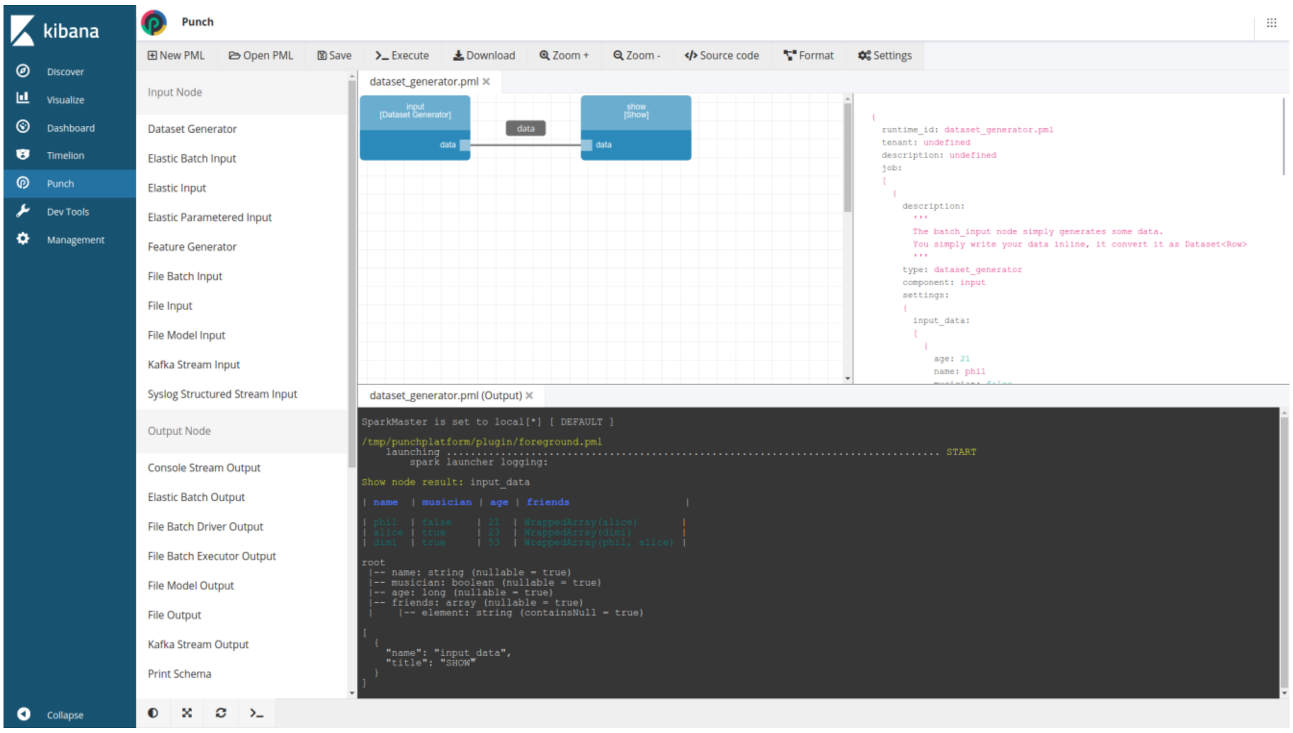
Need help ? Visit the integrated punchplatform documentation.

Figure 13 Punch Platform graphical interface

The menus shown in Figure 13 provide access to the Punch Platform toolset:

- Platform Resources: Configuration of channels punchlets.
- Data extractions: Access to a data extraction interface to export data either in CSV format or to consult them in Kibana.
- Punch Machine Learning: Access to the Punch Machine Learning (PML) graphical editor for pipeline creation using Spark components.
- Punch & Grok testers: Easily test and run punchlets or grok patterns.
- Documentation: Access to the Punch Platform documentation.

The PML editor has been created to minimize the workload for the definition of a pipeline. By using a drag and drop interface, the components can be added and interconnected easily. Of course, some configuration still has to be done, but the workload is reduced to speed up data scientists' tests.



```
{
  runtime_id: dataset_generator.pml
  tenant: undefined
  description: undefined
  job:
  [
    {
      description:
        '''
        The batch_input node simply generates some data.
        You simply write your data inline, it convert it as Dataset<Row>
        '''
      type: dataset_generator
      component: input
      settings:
      {
        input_data:
        [
          {
            age: 21
            name: phil
```

```
SparkMaster is set to local[*] [ DEFAULT ]

/tmp/punchplatform/plugin/foreground.pml
    launching ............................................................................... START
        spark launcher logging:

Show node result: input_data

| name  | musician | age | friends                    |

| phil  | false    | 21  | WrappedArray(alice)        |
| alice | true     | 23  | WrappedArray(dimi)         |
| dimi  | true     | 53  | WrappedArray(phil, alice)  |

root
 |-- name: string (nullable = true)
 |-- musician: boolean (nullable = true)
 |-- age: long (nullable = true)
 |-- friends: array (nullable = true)
 |    |-- element: string (containsNull = true)

[
  {
    "name": "input_data",
    "title": "SHOW"
  }
]
```

Figure 14 Punch Platform – Output example

An example of output for the Punch Platform is presented in Figure 14.

Annex A1.4: SIMARGL proposed use

The Punch is a ready-to-go-production data analytics solution. It is built on a foundation of key open source technologies such as Elasticsearch, spark and Kafka, all packaged together with a number of business modules in a single well-documented and supported distribution.

The Punch is unique since it lets users design data processing pipelines via configuration. These pipelines cover a wide range of functional use cases: data collection and transport from edge to central platforms, data parsing, filtering and enrichment, real time alerting, or machine learning. Combined with cutting-edge data visualization capabilities, multi-tenancy and security, the Punch Platform can be deployed on cloud or on-premise infrastructures and provide a production-ready service platform.

Based on this principle, each SIMARGL consortium member could develop and test their work on their side and deploy it in the end-user environment without modifications. Once the Punch Platform components themselves are deployed, each SIMARGL component would be independent.

Each SIMARGL consortium member would be associated with a Punch Platform tenant. This basically means that each member:

- is restricted to access only its own data.
- can design processing pipelines on its own, with no risk to affect other members.
- can leverage all the existing punch modules, and/or deliver its own extensions.
- can decide to share or not its extensions with other members.

This feature could secure the behaviour of the components by making impossible side effects from one component to another. This kind of problems can occur when integrating components from different providers, and can be hard to identify and to correct. And, of course, it is possible to add tenants, and to stop / start their components without any impact on other tenants, with the exception of those relying upon shared hardware resources.

All these capabilities are already in use on Thales cybersecurity platforms, deployed as SaaS-managed services and processing data from several customers, each associated with a tenant. In addition, the Punch Platform security module provides finer grain role-based access control, which should be required to further restrict or limit the access to some data from several users from the same consortium.

It is worth understanding the processing capabilities of the Punch Platform and how these can be used by each consortium member to design and deploy its own components. This can be done as follows:

Pipelines are defined by the user via simple configuration files. A graphical editor (shown in Figure 14) is also provided to make it straightforward to design pipelines, even machine learning use cases. The Punch Platform then translates these pipelines into executable units and leverage apache Storm and Spark runtimes to benefit from high-performance and scalable processing.

The Punch Platform comes with a number of ready-to-use modules covering a wide array of data access and manipulation, leveraging a SQL language to deal with real-time correlation (SQL rule engine) or with large datasets (leveraging SparkQL). Standard log parsers are also shipped in.

In addition to these modules, consortium members can also develop their own business modules, should it be required to enrich the Punch Platform with domain-specific functions. Additional modules can be coded in Python, Java as well as by using the Punch Platform compact script language, which is the programming language of the platform itself. Choosing the right development environment depends on the use case. Whatever language is selected, the new modules can be delivered to the platform and, in turn, deployed as part of pipelines.

This clean and simple model has several key advantages:

- Additional modules can be coded with few or no adherence with the underlying technologies APIs.
- Once deployed in Punch Platform, pipelines benefit from the integrated monitoring and security.
- Upgrading a Punch Platform with the latest COTS hardware/software technologies is simple and with little or no impact to the existing modules.

This overall strategy, modularity, and design have been consolidated by more than five years upgrading large-scale production platforms in Thales applications deployed worldwide.

Another important key capability of the Punch Platform is its extensibility and modularity. In most cases, a single platform is used by several tenants, so as to mutualize the platform components. This results in a simple to monitor and cost-effective solution. It may be required to deploy some resources specifically for a given tenant or to augment the resources (disk/cpu/ram/servers) of some mutualized services. Both are simple and easy operations. The Punch Platform has been specifically designed to start small**,** and then be scaled up in proportion to the tenants' and/or data traffic needs.
Lastly, the Punch Platform aims at leveraging open source technologies. There is no vendor locking risks, all modules can be easily run on standard Java, Python or open-source frameworks. It means that consortium members can develop their components outside the Punch Platform and then integrate them in a later phase. Using components with the Punch Platform within SIMARGL and in other contexts does not require creating different versions of the components.

The Punch Platform documentation is available online: https://punchplatform.com/.

Annex A2: Orion Malware

Orion Malware is a file analysis solution provided by Airbus CyberSecurity.

Orion combines several analysers to perform investigation and assessments on files. The following analysers can be activated independently:

- **Scanner:** Static analyser with parsing capabilities of different file format (Office, EXE, PDF, HTML, etc.) and able to implement a variety of threat heuristics;
- **Magic:** Automated reverse engineering of malware executable with C&C and malware configuration extraction capabilities;
- **Antivirus:** Set of commercial antiviruses run in parallel;
- **Sandbox:** Airbus CyberSecurity own sandbox (QSPY) for dynamic file analysis (Files are detonated in virtual machines).

Annex A2.1: Performance

Table 1 gives an estimation of files analysis performance for each appliance models.

Orion appliance models	240	440	540	640
	Files/hour	**Files/hour**	**Files/hour**	**Files/hour**
Scanner + magic	**900**	**1000**	**2050**	**4200**
5 Antivirus	**700**	**800**	**1100**	**2300**
5 Antivirus + scanner + magic	**700**	**800**	**1100**	**2300**
5 Antivirus + scanner + magic + sandbox	**100**	**250**	**650**	**1250**

Table 1: Orion Malware - Performance

Annex A2.2: Graphical interface

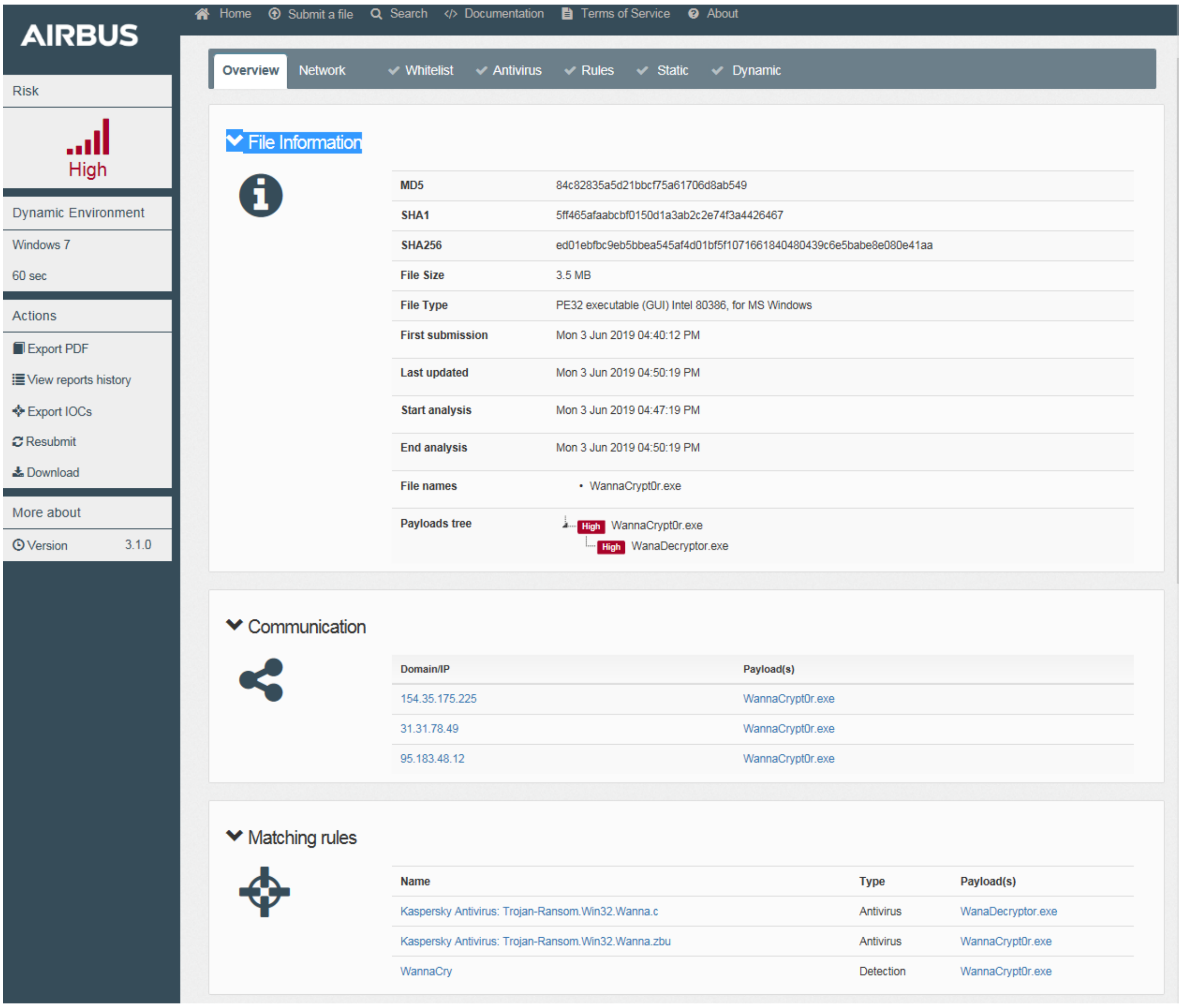
This subsection resumes some information included in the analysis report provided through the Graphical Interface of Orion Malware.
The analysis report can also be exported as:

- PDF document
- JSON file
- MISP event

Orion Malware graphical interface is made up of seven tabs presenting the analysis results through different aspects.

The first one, name “Overview” and illustrated in Figure 15 and Figure 16 gives a résumé of the analysis including the following elements:

- File information
- Communication
- Matching rules
- Risk of Compromising



File Information

MD5	84c82835a5d21bbcf75a61706d8ab549
SHA1	5ff465afaabcbf0150d1a3ab2c2e74f3a4426467
SHA256	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
File Size	3.5 MB
File Type	PE32 executable (GUI) Intel 80386, for MS Windows
First submission	Mon 3 Jun 2019 04:40:12 PM
Last updated	Mon 3 Jun 2019 04:50:19 PM
Start analysis	Mon 3 Jun 2019 04:47:19 PM
End analysis	Mon 3 Jun 2019 04:50:19 PM
File names	• WannaCrypt0r.exe
Payloads tree	High WannaCrypt0r.exe High WanaDecryptor.exe

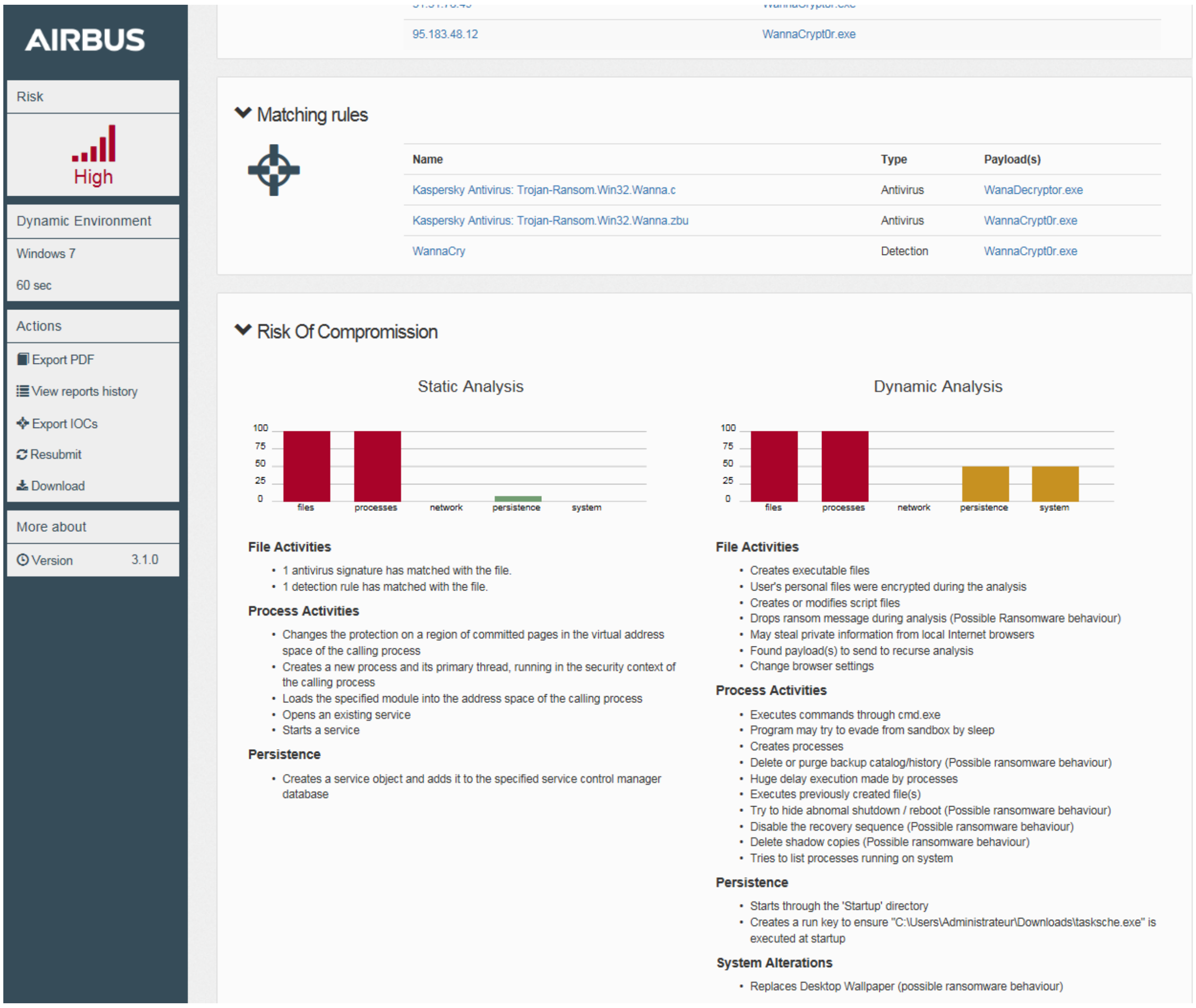
Communication

Domain/IP	Payload(s)
154.35.175.225	WannaCrypt0r.exe
31.31.78.49	WannaCrypt0r.exe
95.183.48.12	WannaCrypt0r.exe

Matching rules

Name	Type	Payload(s)
Kaspersky Antivirus: Trojan-Ransom.Win32.Wanna.c	Antivirus	WanaDecryptor.exe
Kaspersky Antivirus: Trojan-Ransom.Win32.Wanna.zbu	Antivirus	WannaCrypt0r.exe
WannaCry	Detection	WannaCrypt0r.exe

Figure 15 Orion Malware - Overview Tab – Part 1



95.183.48.12	WannaCrypt0r.exe

Matching rules

Name	Type	Payload(s)
Kaspersky Antivirus: Trojan-Ransom.Win32.Wanna.c	Antivirus	WanaDecryptor.exe
Kaspersky Antivirus: Trojan-Ransom.Win32.Wanna.zbu	Antivirus	WannaCrypt0r.exe
WannaCry	Detection	WannaCrypt0r.exe

Risk Of Compromission

Static Analysis

File Activities

- 1 antivirus signature has matched with the file.
- 1 detection rule has matched with the file.

Process Activities

- Changes the protection on a region of committed pages in the virtual address space of the calling process
- Creates a new process and its primary thread, running in the security context of the calling process
- Loads the specified module into the address space of the calling process
- Opens an existing service
- Starts a service

Persistence

- Creates a service object and adds it to the specified service control manager database

Dynamic Analysis

File Activities

- Creates executable files
- User's personal files were encrypted during the analysis
- Creates or modifies script files
- Drops ransom message during analysis (Possible Ransomware behaviour)
- May steal private information from local Internet browsers
- Found payload(s) to send to recurse analysis
- Change browser settings

Process Activities

- Executes commands through cmd.exe
- Program may try to evade from sandbox by sleep
- Creates processes
- Delete or purge backup catalog/history (Possible ransomware behaviour)
- Huge delay execution made by processes
- Executes previously created file(s)
- Try to hide abnormal shutdown / reboot (Possible ransomware behaviour)
- Disable the recovery sequence (Possible ransomware behaviour)
- Delete shadow copies (Possible ransomware behaviour)
- Tries to list processes running on system

Persistence

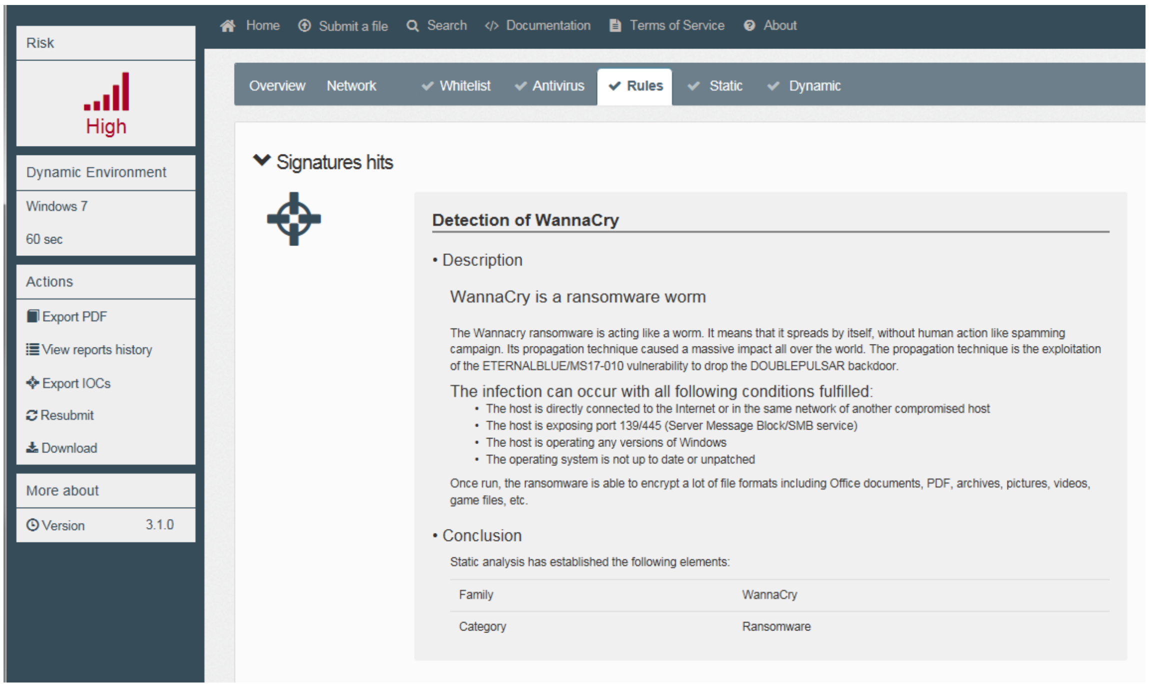
- Starts through the 'Startup' directory
- Creates a run key to ensure "C:\Users\Administrateur\Downloads\tasksche.exe" is executed at startup

System Alterations

- Replaces Desktop Wallpaper (possible ransomware behaviour)

Figure 16 Orion Malware - Overview Tab – Part 2

The "Rules" tab (Figure 17) gives detailed explanations of each signatures match (if any).



Detection of WannaCry

• Description

WannaCry is a ransomware worm

The Wannacry ransomware is acting like a worm. It means that it spreads by itself, without human action like spamming campaign. Its propagation technique caused a massive impact all over the world. The propagation technique is the exploitation of the ETERNALBLUE/MS17-010 vulnerability to drop the DOUBLEPULSAR backdoor.

The infection can occur with all following conditions fulfilled:

- The host is directly connected to the Internet or in the same network of another compromised host
- The host is exposing port 139/445 (Server Message Block/SMB service)
- The host is operating any versions of Windows
- The operating system is not up to date or unpatched

Once run, the ransomware is able to encrypt a lot of file formats including Office documents, PDF, archives, pictures, videos, game files, etc.

• Conclusion

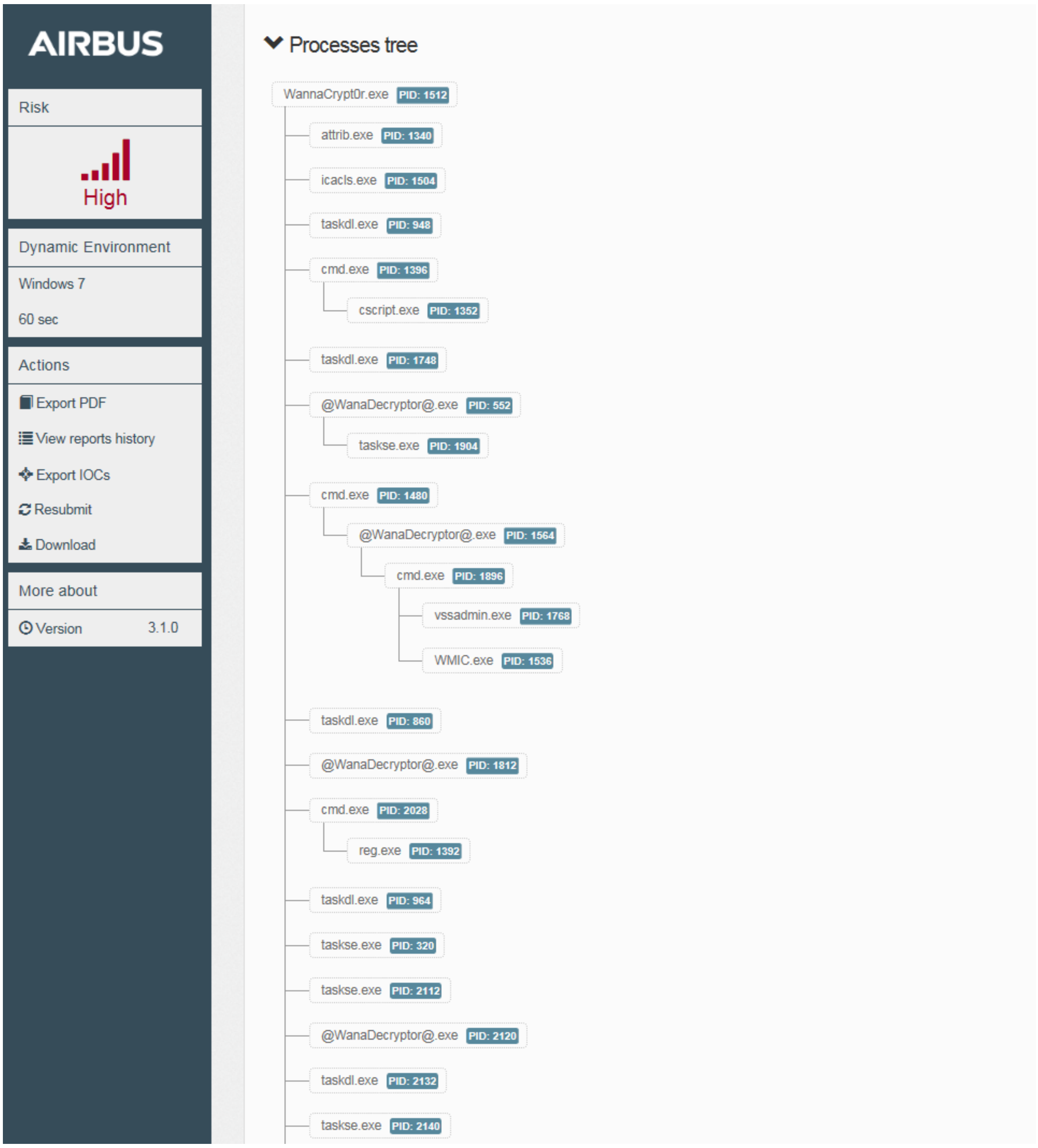
Static analysis has established the following elements:

Family	WannaCry
Category	Ransomware

Figure 17 Orion Malware – Rules Tab

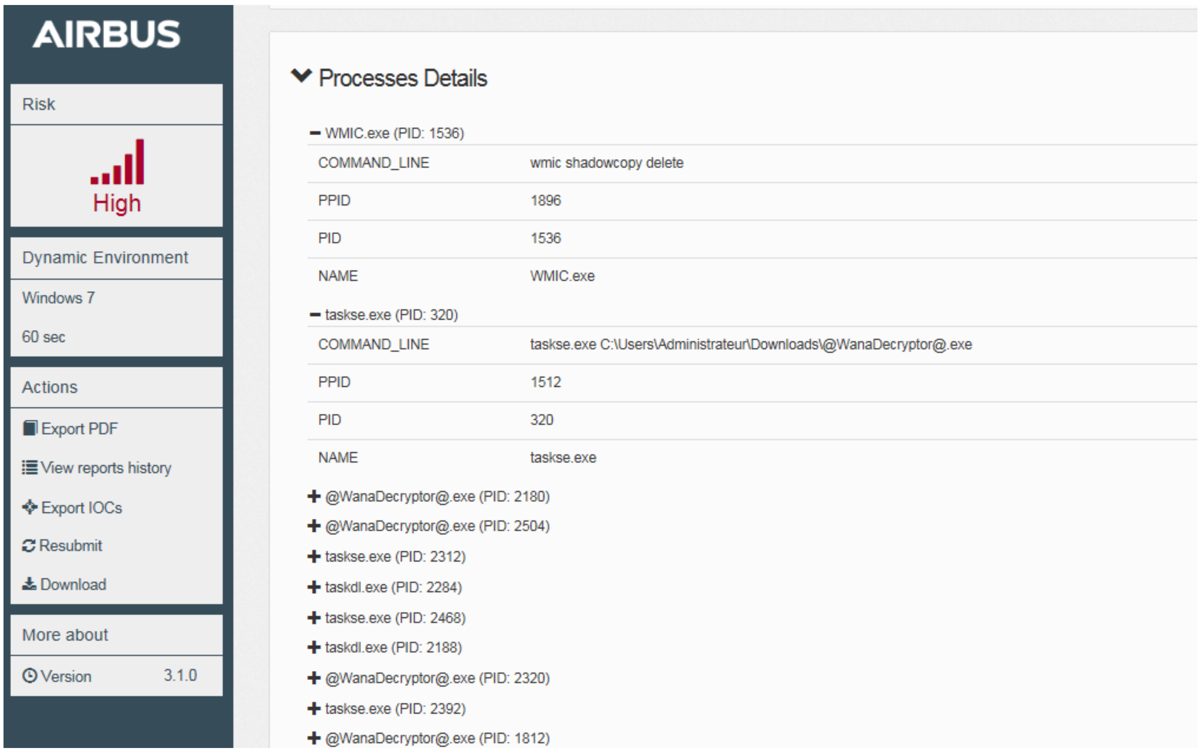
The "Dynamic" tab (Figures 18-20) presents the sandbox report. It contains full technical details of the actions performed by the payload, like:

- Processes creation
- Files creation/alteration/deletion
- Network communication (including DNS queries)
- Registry modifications
- Evasion techniques



```
WannaCrypt0r.exe PID: 1512
├── attrib.exe PID: 1340
├── icacls.exe PID: 1504
├── taskdl.exe PID: 948
├── cmd.exe PID: 1396
│   └── cscript.exe PID: 1352
├── taskdl.exe PID: 1748
├── @WanaDecryptor@.exe PID: 552
│   └── taskse.exe PID: 1904
├── cmd.exe PID: 1480
│   └── @WanaDecryptor@.exe PID: 1564
│       └── cmd.exe PID: 1896
│           ├── vssadmin.exe PID: 1768
│           └── WMIC.exe PID: 1536
├── taskdl.exe PID: 860
├── @WanaDecryptor@.exe PID: 1812
├── cmd.exe PID: 2028
│   └── reg.exe PID: 1392
├── taskdl.exe PID: 964
├── taskse.exe PID: 320
├── taskse.exe PID: 2112
├── @WanaDecryptor@.exe PID: 2120
├── taskdl.exe PID: 2132
└── taskse.exe PID: 2140
```

Figure 18 Orion Malware – Dynamic Tab – Processes Tree



Processes Details

− WMIC.exe (PID: 1536)

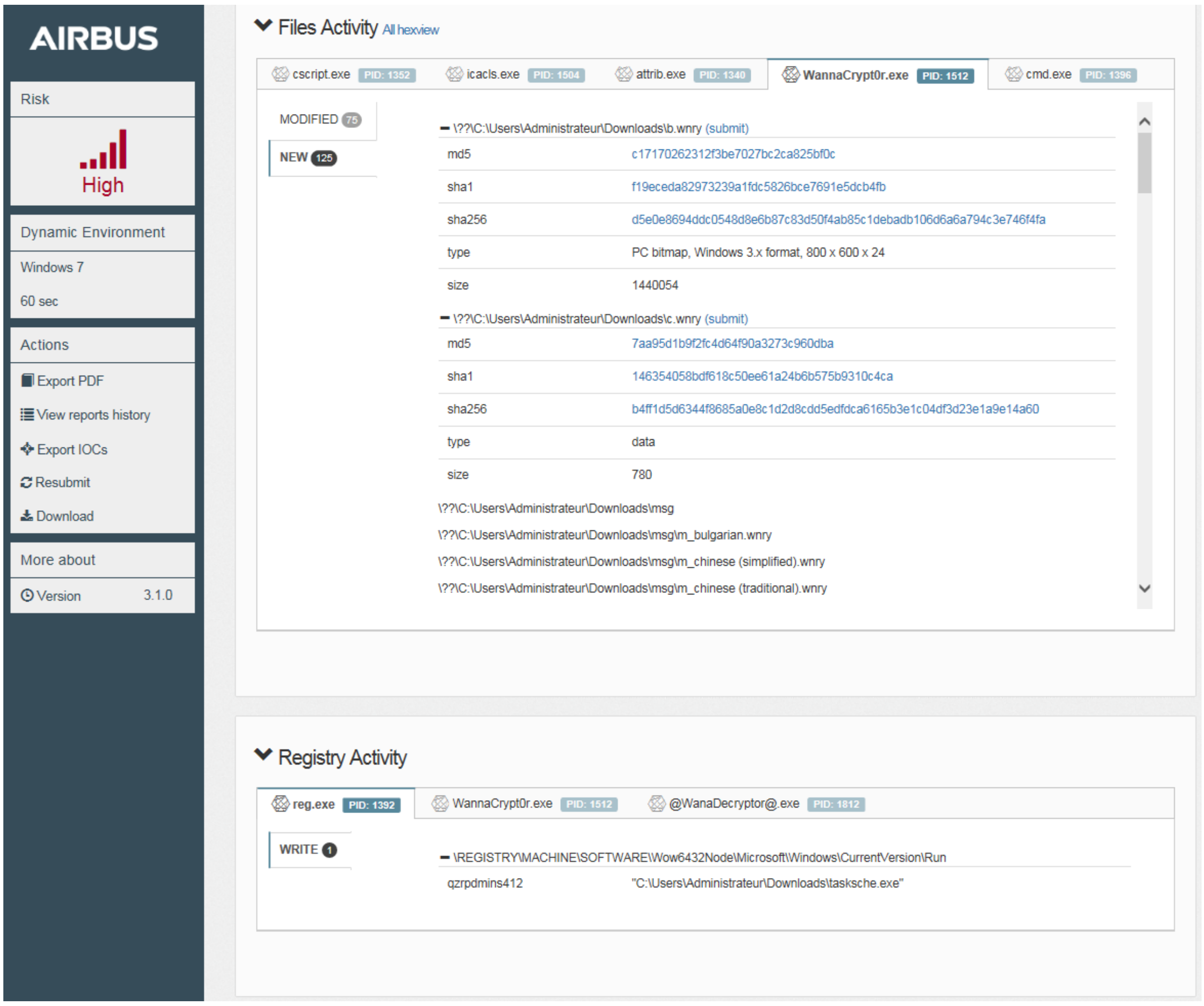
COMMAND_LINE	wmic shadowcopy delete
PPID	1896
PID	1536
NAME	WMIC.exe

− taskse.exe (PID: 320)

COMMAND_LINE	taskse.exe C:\Users\Administrateur\Downloads\@WanaDecryptor@.exe
PPID	1512
PID	320
NAME	taskse.exe

+ @WanaDecryptor@.exe (PID: 2180)
+ @WanaDecryptor@.exe (PID: 2504)
+ taskse.exe (PID: 2312)
+ taskdl.exe (PID: 2284)
+ taskse.exe (PID: 2468)
+ taskdl.exe (PID: 2188)
+ @WanaDecryptor@.exe (PID: 2320)
+ taskse.exe (PID: 2392)
+ @WanaDecryptor@.exe (PID: 1812)

Figure 19 Orion Malware – Dynamic Tab – Processes Details



\??\C:\Users\Administrateur\Downloads\b.wnry (submit)

md5	c17170262312f3be7027bc2ca825bf0c
sha1	f19eceda82973239a1fdc5826bce7691e5dcb4fb
sha256	d5e0e8694ddc0548d8e6b87c83d50f4ab85c1debadb106d6a6a794c3e746f4fa
type	PC bitmap, Windows 3.x format, 800 x 600 x 24
size	1440054

\??\C:\Users\Administrateur\Downloads\c.wnry (submit)

md5	7aa95d1b9f2fc4d64f90a3273c960dba
sha1	146354058bdf618c50ee61a24b6b575b9310c4ca
sha256	b4ff1d5d6344f8685a0e8c1d2d8cdd5edfdca6165b3e1c04df3d23e1a9e14a60
type	data
size	780

\??\C:\Users\Administrateur\Downloads\msg

\??\C:\Users\Administrateur\Downloads\msg\m_bulgarian.wnry

\??\C:\Users\Administrateur\Downloads\msg\m_chinese (simplified).wnry

\??\C:\Users\Administrateur\Downloads\msg\m_chinese (traditional).wnry

\REGISTRY\MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run

qzrpdmins412	"C:\Users\Administrateur\Downloads\tasksche.exe"

Figure 20 Orion Malware – Dynamic Tab – Files Activity

Annex A2.3: Orion extendibility

Orion Malware can use a set of detection and unpacking rules for both static and dynamic analyses. The aim of this subsection is to present how rules are managed.

Rules can be manually or automatically fetched from a remote repository (UCPP) or imported from a file. Available formats are:

- Yara
- openIOC
- Magic plugins: Magic is a library developed by Airbus CyberSecurity to allow automatic reverse engineering of malware belonging to the same family. Its main goal is to be able to unpack malware configuration files and extract information like C&C IP/DNS, set of commands, just to mention some. Magic plugins are developed in Python and the magic framework can also be used to add new detection capabilities to Orion like new file types support, machine learning for specific files etc.
- Heuristic plugins: Heuristic plugins are Python modules which allow the user to add new detection heuristic based on static and dynamic reports. This allows, for example, the user to add new detection "rules" tailored for his environment. Imported rules can be downloaded, enabled, disabled or deleted. Rules are regrouped by ruleset. A ruleset can be enabled, disabled, downloaded or deleted.

Annex A2.3.1: Rules restrictions

- Yara - The Yara engine used for analysis and parsing rules is at version 3.6.3
- openIOC - The elements that can be used in IOC XML file are listed below:
    - Context document:
        - FileItem
        - RegistryItem
        - ProcessItem
        - Network
        - PortItem
    - Context search:
        - FileItem/FileName
        - FileItem/FullPath
        - FileItem/FilePath
        - FileItem/FileExtension
        - FileItem/Md5sum
        - FileItem/Sha1sum
        - FileItem/Sha256sum
        - FileItem/SizeInBytes
        - RegistryItem/ValueName
        - RegistryItem/Path
        - RegistryItem/Value
        - RegistryItem/KeyPath
        - FileItem/StringList/string
        - RegistryItem/Type
        - ProcessItem/name
        - ProcessItem/path
        - ProcessItem/arguments
        - ProcessItem/SectionList/MemorySection/Injected
        - Network/DNS
        - Network/URI
        - Network/UserAgent
        - Network/HTTP_Referr
        - PortItem/remoteIP
        - PortItem/remotePort
        - PortItem/protocol
    - PortItem/process
- Magic - All magic modules should be written in Python 2.7. Only modules with description will match files during analysis

Annex A2.3.2: Plugins – Static analysis plugins

The aim of this section is to provide basic knowledge necessary to create static analysis plugins to customize Orion Malware detection.

Magic plugins - Magic plugins are Python classes integrated into the Magic component of Orion Malware. This component is in charge of the static analysis based on rules and is used to:

- Detect malicious samples
- Extract of features (C&C addresses, configuration information)
- Unpack specific samples

When Magic successfully unpacks a sample, the unpacked file is analysed. This enables the detection of packed samples.

Structure

The plugin must define a class for the module and import magic components. It may import necessary Python modules (e.g., sys, pefile).

Below we report the functions that Magic will try to launch in order to detect and extract information from a sample:

- **__init__:** should initialize the Magic module and set information about the sample.
- **Identify:** will be called once by Magic to see if the sample being analysed matches.
- **Run:** will be called once the sample has been identified; its role is to extract information about the sample and unpack payloads (if possible).
- **get_payload:** will be called by Magic until it returns "None"; each returned value must be an unpacked payload.

Method __init__: Initialization

The **__init__** method defines variables that describe the sample. The *RELEASE* variable defines the maturity of the rule. By default, only beta rules and above are taken into account by Magic. Possible values are:

- "stable"
- "beta"
- "alpha"

Listing 1 is an example of the Magic module basic structure, containing the headers and the **__init__** method:

```
from sa.magic.core.module import Module
from sa.magic.core.constante import *

    class sample(Module):
        RELEASE = "stable"

        def __init__(self):
            Module.__init__(self)
            self.name = "Sample name"
            self.malware = "Malware name"
            self.description = "Sample description"
            self.description_threatintelligence = "TI description"
            self.category = CATEGORY_RAT
```

Listing 1: Orion Malware - Basic structure of a Magic module

Method identify: Detection

The first usage of Magic plugins is to implement sample detection. The **identify** function (see Listing 2) must be implemented and is given to the file being analysed as a string in the *data* variable.

```
def identify(self, data):
    if (...):
        return MALWARE_IDENTIFIED

    return MALWARE_UNKNOWN
```

Listing 2: Orion Malware - identify function

For the detection module, the return value of this function must be *MALWARE_IDENTIFIED* or *MALWARE_UNKNOWN*.

For the unpacker module, the return value must be PACKER_IDENTIFIED or PACKER_UNKNOWN.

Method run: Feature extraction

If the **identify** method returned *MALWARE_IDENTIFIED* then the **run** method (Listing 3) is called in order to extract information on the sample being analysed. The aim of this method is to extract and parse configuration information and packed payloads.

The extracted information must be returned in a report having a tree layout.
Each reported item is a tuple (field, value).

```
def run(self,data):
    self.init_report()
    e_root = Module.add_section(self, 'module_%s' % (self.__module__.split('.')[-1]))
    self.add_item(e_root, TYPE_CC_ADDRESS, "(address)")
    self.add_item(e_root,TYPE_CC_PORT,"(port)"
```

Listing 3: Orion Malware - **run** method in case of MALWARE_IDENTIFIED

Method get_payload: Unpacking

If the **identify** method returned PACKER_IDENTIFIED then Magic will try to get the payloads by calling the **get_payload** method. This method should return, each time it is called, a different payload amongst the extracted ones or “None” when they all have been returned.

You should first extract the payloads in the *run* method (Listing 4). We suppose we have an **extract_payloads** method that returns a set containing the extracted payloads. It should be called within the **run** method.

```
def run(self,data):
    self.init_report()
    self.payload = extract_payloads(data)
```

Listing 4: Orion Malware - **run** method in case of PACKER_IDENTIFIED

Hence, given that we extracted a set of payloads in the *self.payload* variable, a sensible way to implement the unpacking method is presented in Listing 5.

```
def get_payload(self):
    if len(self.payload) == 0:
        return None

    return self.payload.pop()
```

Listing 5: Orion Malware - **get_payload** method

Annex A2.3.3: Plugins – Dynamic heuristic plugins

The aim of this section is to provide basic knowledge necessary to create dynamic analysis plugins to customize Orion Malware detection.
Dynamic heuristic plugins are Python classes that are integrated into the QSPY component of Orion Malware in charge of the dynamic/behavioural analysis and will be used for:

- Extracts suspicious/malicious indicators from dynamic activities as file activities, registry activities, network activities, etc.
- Identifies dropped files or detected payload to send back in complete analysis (recursive analyse).
- Evaluates the global risk level.

The plugins allow to implement heuristics from the following dynamic activities found by the sandbox:

- File activities
- Process activities

- Registry activities
- Network activities
- Mutex Activities

Structure

The plugin must define a class that inherits **qspy.heuristics.plugins.heuristic.Heuristic** abstract class as presented in Listing 6.

```python
from qspy.heuristics.plugins.heuristic import Heuristic
from qspy.heuristics.api.heuristic_type import HeuristicType
from qspy.heuristics.plugins.heuristic import OsType

class MyHeuristic(Heuristic):
    def init(self):
        self.supported_os_types = [OsType.WINDOWS, OsType.LINUX]
        self.heuristic_type = HeuristicType.PROCESSES
        self.risk_descriptions.add('Your description')
        self.risk_score = 20

    def run(self):
        results = self.search_file_activities(pattern_path=r".*//myfile"),
        actions=[ProcessAction.NEW])
        return len(results) > 0
```

Listing 6: Orion Malware - Definition of class that inherits **qspy.heuristics.plugins.heuristic.Heuristic**

Method init: Initialization

The method **init** must initialize the following variables:

- **supported_os_types**: List of supported operating system.
- **heuristic_type**: Heuristic type corresponding of the domain of compromise related to a defined heuristic.
- **risk_descriptions**: List of description to display in report if heuristic matches.
- **risk_score**: score between 0 to 100 to add to category defined by heuristic_type.

For each analysis report, heuristic plugins are instantiated and **init** method is called.

Method run: Detection

The method **run** is called only if the supported operating systems declared by the plugin match with the current analysis.

The method **run** must return **True** if heuristic matches otherwise **False**.

If method **run** returns **True**, risk score and descriptions will be added in risk section of the report and will be taken into account in the global risk evaluation.

Annex A2.4: SIMARGL proposed use

Orion Malware is a solution providing both static and dynamic file analysis.

Its main usage within the SIMARGL project will be to provide detailed files analysis coming from different assets, like for example:

- Network probes (like CYBELS Sensor from Thales)
- WEB proxy
- Mail gateway

Orion extendibility will also allow implementing new detection techniques like those aiming at revealing the presence of information-hiding-capable malware, steganography-based communication and covert channels, and scripts obfuscation, just to mention some.

Annex A2.5: Data flow

Orion Malware is file based and does not provide network capture features. Hence it relies on other assets for its inputs (as described in section 0).

Default supported file formats - In version 3.3 Orion Malware supports the following file formats:

- PE32: executable, binary
- PE32+: all type of executable
- PDF (1.5, 1.6, 1.7): the acrobat reader format
- Microsoft Office (97 to 2003): doc, docx, docm, dotx, dotm, xls, xlsm, xltm, ppt, pptx, ppsm, psx,ptm, potm, potx
- Data: all types are accepted. The type of the file is the one found by the Unix command “file”.
- ZIP: always available, encrypted with password “infected” (basic form of encryption), with basic compression mode as store and “deflate” (the bzip format is not supported). Beware that the AES-256 encryption is not supported.
- RAR
- 7z: 7zip archive
- GZIP: tar archive
- DOS: DOS executable
- RTF: Rich Text Format
- SWF: Macromedia Flash files
- APK: Android application files
- JS: Javascript files
- EMAIL : SMTP mail, RFC 822 mail, MIME entity, HTML document
- Microsoft WINHelp: WinHelp file ".chm"
- JAR: Java archive file
- ELF: Linux/GNU/SOlaris/BSD binary files
- CAB and MSI files: auto-extractible format
- OpenOffice

Export data formats - Orion Malware provides several outputs:

- HTML report
- PDF report
- JSON report (through REST APIs)
- MISP (Malware Information Sharing Platform) event (with a configured MISP server)

Input data flow - Orion Malware can accept requests through:

- Manual WEB submission
- MAIL
- ICAP (Internet Content Adaptation Protocol)
- REST APIs

Data flow overview – Figure 21 details some possible data flows with Orion in the SIMARGL project.

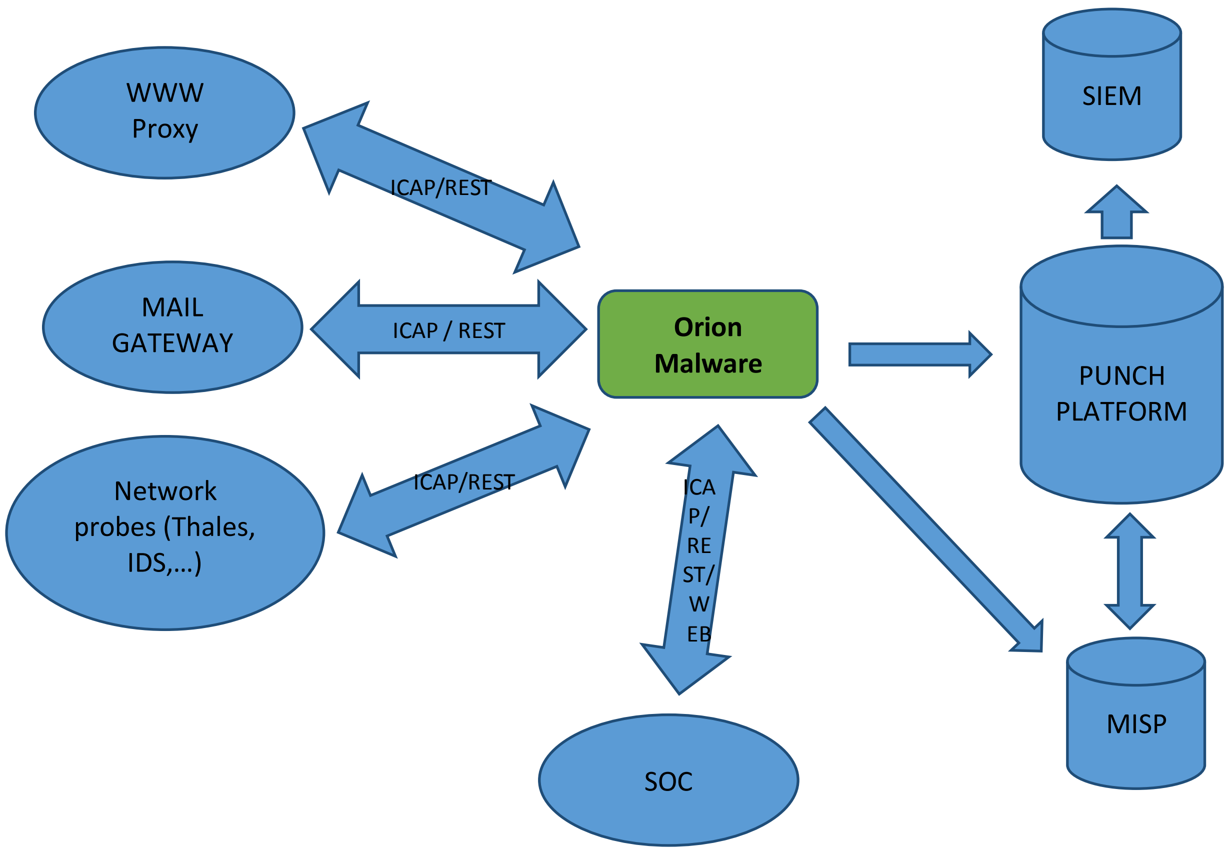


Figure 21 Orion Malware - Data flow overview

Annex A2.6: REST API

Orion Malware is accessed using REST APIs. The REST API conforms to standard architectural principles defined by Representational State Transfer (REST). It supports these key architectural concepts common to RESTful services.

Uses stateless interactions: The REST service does not use login sessions or store other state information on the server. Instead, the client maintains this information about each resource, which makes the REST service easier to use across load-balanced servers. For general introduction, see Representational state transfer at https://en.wikipedia.org/wiki/Representational_state_transfer.

Complete API documentation is available in the annex document ***SIMARGL-D2.3-Annex A.docx***

Annex A3: CYBELS Sensor

CYBELS Sensor may not be in the systems of end users running the SIMARGL toolkit. Nevertheless, the network probe deployed in SIMARGL will be based on the same core component in order to be able to reuse all of the work done within SIMARGL. In the following, we will describe the main capacities of CYBELS Sensor.

In essence, CYBELS Sensor is a cyberattack detection probe designed specifically to monitor critical infrastructures. It has been certified by the ANSSI (French agency for cybersecurity) as a secured probed to be used by French companies. It has been identified as being of vital importance by the authorities, to fulfil their legal obligations. CYBELS Sensor combines signature-based detection on network traffic and files exchanged over the network with metadata generation to help operators to identify malicious traffic and have a better understanding of the overall traffic semantic.

Annex A3.1: Components description

CYBELS Sensor is composed of three parts:

- The probe, which is connected to the network to analyse. It searches for the signatures within the traffic, and produces events whenever it finds one. It also produces metadata about the analysed traffic.
- The management centre is an appliance that allows probe management. It is from this console that the detection rules are deployed on the probe. A management centre can be used to manage several probes.
- The operation centre is an appliance used to visualize the events generated by the probes. These events can either be about the traffic or about the security state of the sensor.

The operation centre can forward events using syslog protocol to a correlator, a SOC or any system compatible with the syslog protocol. The link between the probes and the management and operation centre is secured using IPSec.

Depending on the model, CYBELS Sensor is able to analyse a throughput of:

- 1 Gbps.
- 2 Gbps.
- 4 Gbps.
- 10 Gbps.

Annex A3.2: Detection capabilities

CYBELS Sensor can detect threats on networks using several detection capabilities:

- Search for complex attack signatures (i.e., DPI).
- File analysis.
- Netflow information generation.
- Metadata generation.

If Netflow and metadata are not used as detection elements themselves, they can be exploited either to have a better understanding of the traffic, or to feed machine learning algorithms to recognize threats.

Annex A4: Attack Prophecy (Pluribus One)

Attack Prophecy® is a modern solution for the active monitoring and protection of the web services. Thanks to its AI-powered engine, it has the capability to reconstruct from the traffic incoming to Web Services a model of their expected and legitimate behaviour. Once built, such model is then matched against the incoming traffic, enabling the detection of vulnerabilities, attacks exploiting them, as well as to identify misconfigurations and other issues which may somehow impair the regular behaviour of the services.
The human operator has the chance to provide feedbacks to the systems, validating requests which Attack Prophecy marks as "anomalous" as legitimate (in the case they are just a false alarm) or malicious. In this latter case, an anomaly becomes an "alert", which is promptly notified to the Security Operation Center by the means of a connection with the SIEM. Whilst Attack Prophecy has the capability to place under the "alerts" category requests considered malicious with a high confidence, it benefits from the operator feedback, which allows the learnt model to become progressively more precise and thus more effective in spotting threats. This "Learn, Detect, and Protect" approach is represented in Figure 22.

A distinguishing feature of Attack Prophecy is the availability of a guided procedure for the creation of protection rules to be deployed on board the Web Application Firewall standing upfront the services, which allows the operator to create rules which are specific for the monitored services. Such rules, which are learnt from the traffic and which then allow to protect Web Services even in presence of custom applications (e.g., that is when Web Services are not operated leveraging standard CMSs) are easily defined by the operator through a graphical interface and without the need for the operator itself to hand manually complicated and error-prone regular expressions.

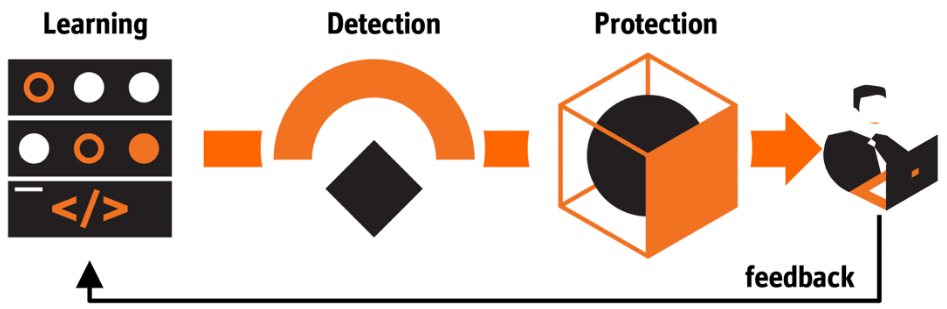


Figure 22 The "Learn", "Detect", "Protect" approach implemented by Attack Prophecy®.

Annex A4.1: Sizing and Processing Capabilities

The internal architecture of Attack Prophecy and the highly efficient approach to the traffic analysis it implements internally, make the solution not greedy in terms of HW resources required. Thus, it is possible to monitor even services belonging to entities receiving large amounts of traffic (e.g., Public Administrations, Large Companies, Universities) with limited amount of resources.
Nevertheless, whilst the sizing is not critical, we do provide below here a qualitative indication of the minimum amount of resources required per installation type, separating them in three different sizes (Small, Medium, Large) and providing also an indication of the type of user which may correspond to each size.
It is in addition worth to remark that being Attack Prophecy a "passive" component (it observes the traffic in real time but it is not in-line as the reverse proxy, the load balancer, or the ADC) no issues on the availability of the services may arise from an insufficient amount of resources assigned to the Attack Prophecy virtual appliance.

The following list presents various types of installation and their recommendations and requirements:

- Type of Installation: **Small**

- **Recommended CPUs:** 4
- **RAM Memory:** 4GBs
- **Disk space required:** 40GB
- **HTTP queries per day (on average):** hundreds of thousands.
- **Type of user:** University Research Laboratory, Small company

- Type of Installation: **Medium**
    - **Recommended CPUs:** 6
    - **RAM Memory:** 8GBs
    - **Disk space required:** 100GB
    - **HTTP queries per day (on average):** up to 1-2 Million
    - **Type of user:** Medium size company

- Type of Installation: **Large**
    - **Recommended CPUs:** 8
    - **RAM Memory:** 16 GBs
    - **Disk space required:** 300GB
    - **HTTP queries per day (on average):** tens of millions
    - **Type of user:** Large Public Administration, Large University

Annex A4.2: Deployment and Installation procedure

The following sections present information about the Attack Prophecy's deployment options and installation.

Annex A4.2.1: Deployment Options

Attack Prophecy natively supports the possibility of being deployed according to different schemes, which makes possible to meet a wide range of different needs.

Generally speaking, two are the components Attack Prophecy needs to interact with:

- **Data Source (mandatory):** we refer with the word *data source* to any component from which Attack Prophecy can read the traffic directed toward the web services. The data source might be then represented by the web server itself, or by the reverse proxy(-ies), the load balancer(s), or the application delivery controller(s) deployed in front of the web server themselves.

- **Interceptor (optional):** we refer with the word *interceptor* to any component able to use the protection rules created via Attack Prophecy to inspect the traffic and to apply some specific action on the malicious requests. For instance, an interceptor might be the Web Application Firewall, the Application Delivery Controller, the Reverse Proxy, or eventually even the firewall if it has the capability to inspect the traffic at the application layer. The presence of the *interceptor* is required only to enable the possibility to protect web services via the protection rules: thus, in the case an interceptor is not available or is not configured, Attack Prophecy is still able to monitor the traffic and to spot malicious requests.

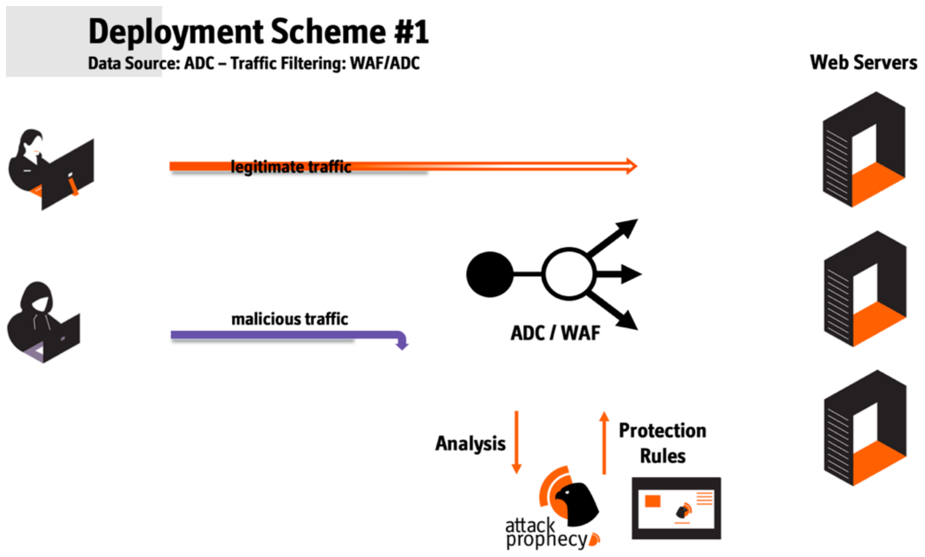


Figure 23 Deployment Scheme #1 of Attack Prophecy (1).

Figure 23 presents how the traffic is read from the Application Delivery Controller, which is also capable to operate as *interceptor* matching the protection rules generated by Attack Prophecy against the incoming network traffic.

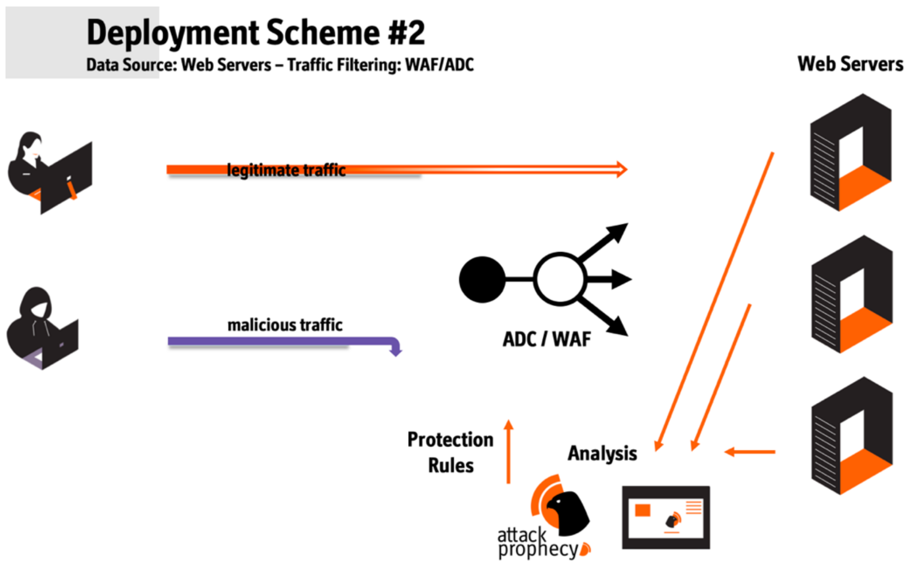


Figure 24 Deployment Scheme #1 of Attack Prophecy (2).

In Figure 24, the traffic is read from the web servers, which operate as *data sources*. The WAF operates as *interceptor* matching the protection rules generated by Attack Prophecy against the incoming network traffic.

Currently, a number of different *Data* Sources and *Interceptors* is already supported, which are listed below here.

List of Data Sources Currently supported:

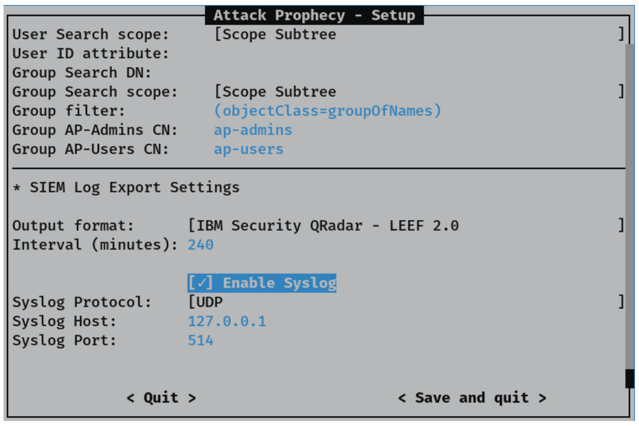
- NGINX - https://www.nginx.com
- Oplon LBL® Application Delivery Controller - https://www.oplon.net/it/product/lbl-adc/
- Apache HTTP Server (forthcoming) - https://httpd.apache.org

List of Interceptors Currently supported:

- ModSecurity - https://modsecurity.org
  - ModSecurity is an Open Source Web Application Firewall which works well with both NGINX and the Apache HTTP Server.
- Oplon LBL®WAF - https://www.oplon.net/it/product/lbl-waf/

Annex A4.2.2: Installation

The Attack Prophecy installation is performed through a guided procedure, which leverages the packet manager available on standard Linux distributions. Pluribus One currently distributes an **.rpm** package suitable to install Attack Prophecy on Red Hat-based distributions. The system has been widely tested on CentOS 7 distribution, compatibility with which is fully granted by Pluribus One. Support to other Linux distributions, including Debian-based ones will be granted soon as well.
Packages are distributed using Pluribus One owns repositories, which can be then configured[2] among those used by the operating system for software download and update.
An example screenshot from the Attack Prophecy setup procedure is shown in Figure 25.



```
                       Attack Prophecy - Setup
User Search scope:     [Scope Subtree                                ]
User ID attribute:
Group Search DN:
Group Search scope:    [Scope Subtree                                ]
Group filter:          (objectClass=groupOfNames)
Group AP-Admins CN:    ap-admins
Group AP-Users CN:     ap-users

* SIEM Log Export Settings

Output format:      [IBM Security QRadar - LEEF 2.0                  ]
Interval (minutes): 240

                    [✓] Enable Syslog
Syslog Protocol:    [UDP                                             ]
Syslog Host:        127.0.0.1
Syslog Port:        514

              < Quit >                          < Save and quit >
```

Figure 25 One of the screens of the Attack Prophecy setup interface.

Annex A4.3: Detection Capabilities

A list of the detection modules available with Attack Prophecy follows:

1. Probing: This macro-category includes all the automatic attacks that are characterized by a high number of failed requests against web services. For example, information gathering attacks to enumerate vulnerable resources and configuration errors on the monitored services. A typical case study for these attacks is represented by scans against web applications (e.g., Open-Source Content Management System) whose vulnerability is known, or scans to detect methods and protocols supported by the server, thus attempting to use it improperly (e.g., as proxy).

2. SQL Injection: This macro-category includes all the cyber intrusions that resorts to malicious inputs (anomalous) against servers or web applications. For example, this category includes popular injection attacks such as SQL/XPath/OS/LDAP injection, Path Traversal, Remote File Inclusion, Cross Site Scripting. In general, the modules that belong to this category are able to detect any attack

[2] Here https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/sec-configuring_yum_and_yum_repositories is an example of how to configure repositories for YUM, the package manager used by RedHat based distributions.

that is based on “unexpected” input (methods, protocols, domain names, URI, URI attributes) towards servers and web application, in order to exploit their vulnerabilities.

3. Path Traversal

4. Brute Force: This macro-category includes all the automatic attacks that are characterized by a high and repetitive number of requests against web services. For example, this category includes attacks to obtain access credentials for a login page, to automatically dump contents, or to automatically query services.

5. Linked Sites: This macro-category includes all the cyber-attacks that resort to contents (related to the monitored services) that are stored on linked websites/domain names. Such sites are typically crafted by violating legitimate, vulnerable websites, in order to host a phishing page that uses contents of the target. This is done in order to maximize the similarity between the original page and the phishing page.

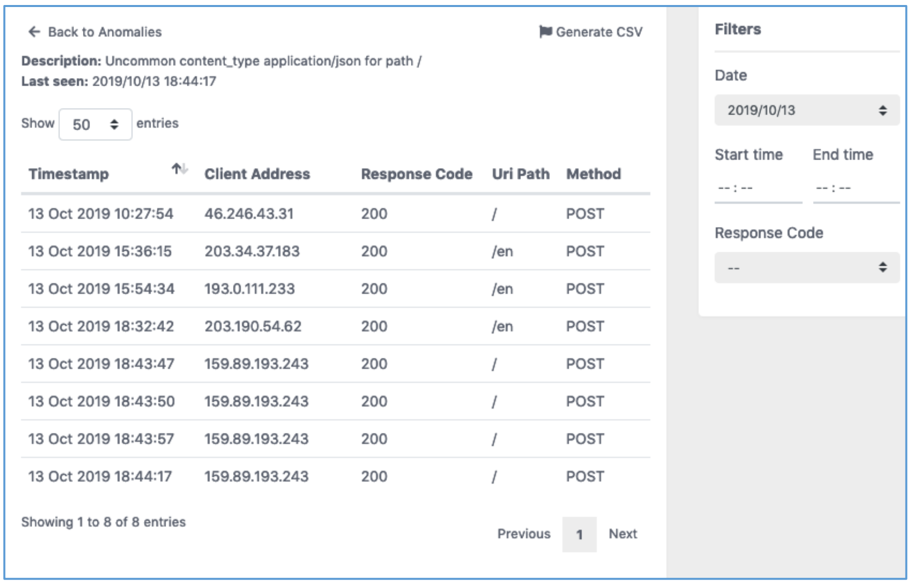
Annex A4.4: Support to Real Time Requirements

All the Attack Prophecy detection modules operate in real-time against the HTTP traffic. Thus, attacks can be reported externally as soon as they are spot by the system.

Annex A4.5: Support to Storage Requirements

Data persistence in Attack Prophecy is granted by a COTS relational database. The database supports full text indexing of the recorded queries, as well as queries on time range.

The operator can easily inspect the data from the Attack Prophecy dashboard, and eventually export it in a CSV file format. An example of how this is possible in the Attack Prophecy dashboard is provided in Figure 26 and Figure 27.



Description: Uncommon content_type application/json for path /
Last seen: 2019/10/13 18:44:17

Show 50 entries

Timestamp	Client Address	Response Code	Uri Path	Method
13 Oct 2019 10:27:54	46.246.43.31	200	/	POST
13 Oct 2019 15:36:15	203.34.37.183	200	/en	POST
13 Oct 2019 15:54:34	193.0.111.233	200	/en	POST
13 Oct 2019 18:32:42	203.190.54.62	200	/en	POST
13 Oct 2019 18:43:47	159.89.193.243	200	/	POST
13 Oct 2019 18:43:50	159.89.193.243	200	/	POST
13 Oct 2019 18:43:57	159.89.193.243	200	/	POST
13 Oct 2019 18:44:17	159.89.193.243	200	/	POST

Showing 1 to 8 of 8 entries

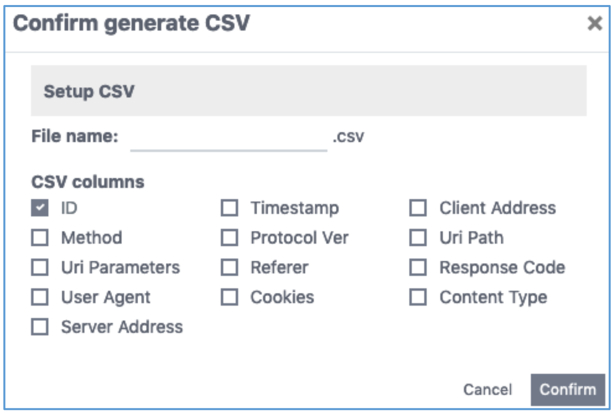
Filters

Date: 2019/10/13

Start time: --:--
End time: --:--

Response Code: --

Figure 26 The Attack Prophecy data inspection dashboard (1)


Confirm generate CSV ×

Setup CSV

File name: .csv

CSV columns

☑ ID	☐ Timestamp	☐ Client Address
☐ Method	☐ Protocol Ver	☐ Uri Path
☐ Uri Parameters	☐ Referer	☐ Response Code
☐ User Agent	☐ Cookies	☐ Content Type
☐ Server Address		

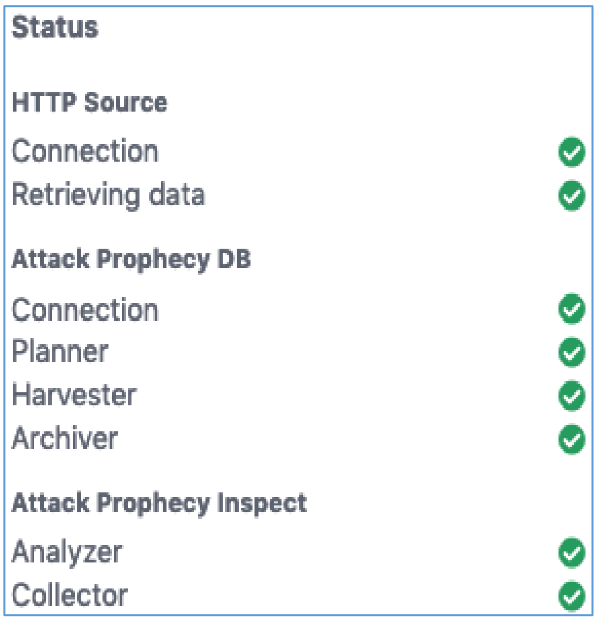
Figure 27 The Attack Prophecy data inspection dashboard (2)

The underlying routine making this possible on the Attack Prophecy dashboard can be leveraged to transfer selected data to SIMARGL, offering thus support to the Storage Requirements.

Depending on the storage availability, a retention period going from a few days up to several months can be configured.

Annex A4.6: Support to Supervision Requirements

Information regarding the Attack Prophecy status is constantly available through the Dashboard, making possible to spot problems as soon as they arise as well as to provide a prompt diagnosis of their cause. Two widgets are available in the dashboard which provide such information: the *Status* widget and the *Server Monitor* widget. The *Status* widget (Figure 28) allows to verify that all the internal services of Attack Prophecy are working properly. Namely, such services are responsible to retrieve the traffic from the source (*HTTP Source* services, which read from the reverse-proxy, the ADC, or from the web-server), to store it in the internal database (*Attack Prophecy DB* services), and to inspect it through the detection modules (*Attack Prophecy Inspect* services). The *Server Monitor* widget (Figure 29) offers instead a perspective on the hardware resources (Disk, CPU, Memory) used by Attack Prophecy.
Such information can be automatically sent out to an external collector of information, for instance by calling an API made available by the collector.


Status

HTTP Source

Connection	✓
Retrieving data	✓

Attack Prophecy DB

Connection	✓
Planner	✓
Harvester	✓
Archiver	✓

Attack Prophecy Inspect

Analyzer	✓
Collector	✓

Figure 28 The Attack Prophecy Status widget

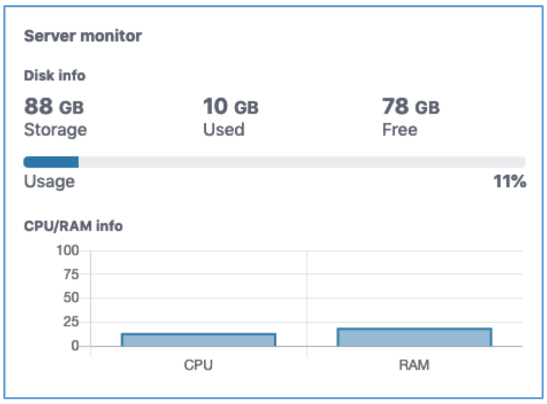


Figure 29 The Attack Prophecy Server Monitor widget.

Annex A4.7: Integration of Attack Prophecy with the SIMARGL platform

The most straightforward way to integrate Attack Prophecy with the SIMARGL platform is to leverage its built-in capability to communicate and share data with other platforms (such as Security Information and Event Management systems), by means of syslogs. The system already provides support for the following formats:

- Log Event Extended Formats (LEEF) – versions 1.0[3], 2.0[4]
- Common Event Format[5] (CEF) – versions 0.1, 1.0

Listing 7 presents an example of a syslog file in the LEEF format.

```
Oct 14 15:01:29 hawaii-attackprophecy-test LEEF: 2.0|Pluribus One|Attack
Prophecy|3.2.8|WordPress probing|9050|cat=Detected count=6
description=Targeted probing for path: /wp-login.php    devTime=Oct 14 2019
18:30:04.000 devTimeFormat=MMM dd yyyy HH:mm:ss.SSS    proto=TCP httpURI=/wp-
login.php    httpDomainName=pralab.diee.unica.it sev=6    src=117.193.150.8
183.83.68.238 213.149.61.17 217.165.183.172 85.204.246.240 89.35.39.60
```

Listing 7: Attack Prophecy - Syslog file in the LEEF format

The portfolio of supported syslog formats will be further extended since the next releases of the solution. Thus, the integration of Attack Prophecy with SIMARGL will potentially benefit from this improvement. Additionally, specific syslogs formats might be defined and implemented to support the particular needs of the SIMARGL platform.

[3] https://developer.ibm.com/qradar/wp-content/uploads/sites/89/2017/02/QRadar_LEEF_Format_Guide_V1.0.pdf
[4]https://www.ibm.com/support/knowledgecenter/SS42VS_DSM/com.ibm.dsm.doc/b_Leef_format_guide.pdf?origURL=SS42VS_DSM/b_Leef_format_guide.pdf
[5] https://community.microfocus.com/t5/ArcSight-Connectors/ArcSight-Common-Event-Format-CEF-Implementation-Standard/ta-p/1645557

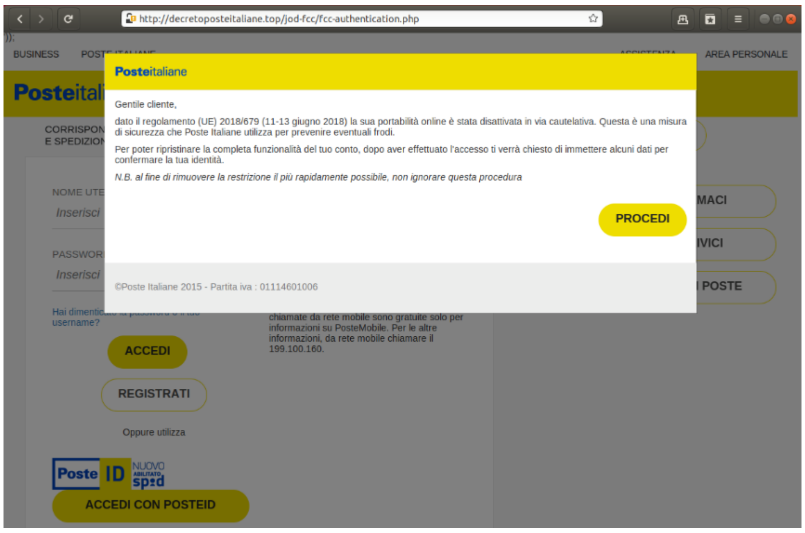
Annex A5: AISafe DNS (Pluribus One)

AISafe DNS is an AI powered solution for the detection of attacks against the endpoints. It belongs to the category of Security Web Gateways[6], as it preserves users and devices from connecting to malicious contents on the Web.

The AI-engine which empowers AISafe DNS has three distinguishing features:

- It is secure against attacks aimed to mislead the detection algorithm, which finally ensures an increased protection and resiliency of the solution;
- It is explainable, as it not only allows to spot an attack pattern but also to properly categorise the threat. The property offers support to the people in the Security Operation Center team, and also enables providing tailored training and awareness programmes toward the victims of the attack.
- It is very-highly-scalable, since it allows to monitor small network as well as large ones (Internet Service Provider level).

It offers protection against a broad range of attacks, including malware, phishing campaigns, and, more in general, all these scams for which the DNS is abused to increase the effectiveness of the attack.
In many of these attacks, the DNS infrastructure plays a pivotal role. In fact, it might be either used to avoid blacklisting (as in the case of DGA malware or Fast-Flux service networks – see an example in Figure 30) or to increase the effectiveness of the attack, especially when domain names under which the pages which deliver the attack are carefully chosen to mislead the users.


http://decretoposteitaliane.top/jod-fcc/fcc-authentication.php

Posteitaliane

Gentile cliente,

dato il regolamento (UE) 2018/679 (11-13 giugno 2018) la sua portabilità online è stata disattivata in via cautelativa. Questa è una misura di sicurezza che Poste Italiane utilizza per prevenire eventuali frodi.

Per poter ripristinare la completa funzionalità del tuo conto, dopo aver effettuato l'accesso ti verrà chiesto di immettere alcuni dati per confermare la tua identità.

N.B. al fine di rimuovere la restrizione il più rapidamente possibile, non ignorare questa procedura

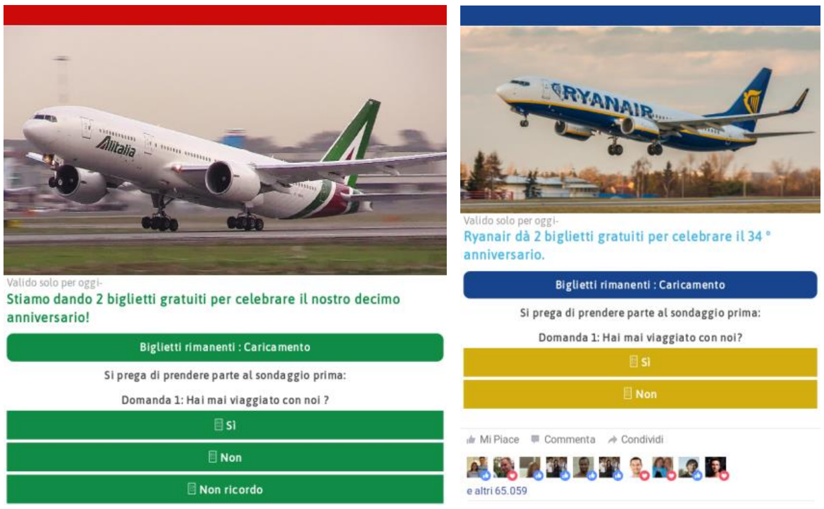
©Poste Italiane 2015 - Partita iva : 01114601006

chiamate da rete mobile sono gratuite solo per informazioni su PosteMobile. Per le altre informazioni, da rete mobile chiamare il 199.100.160.

Figure 30 A phishing page distributed through a Fast-Flux service network.

In Figure 31, it is presented an example of scam fraud targeting the customers of airline operators. The campaign was delivered through typo-squatting attacks which abused Internationalized Domain Names.

[6] https://www.gartner.com/reviews/market/secure-web-gateways



Valido solo per oggi-

Stiamo dando 2 biglietti gratuiti per celebrare il nostro decimo anniversario!

Biglietti rimanenti : Caricamento

Si prega di prendere parte al sondaggio prima:

Domanda 1: Hai mai viaggiato con noi ?

Sì

Non

Non ricordo

Valido solo per oggi-

Ryanair dà 2 biglietti gratuiti per celebrare il 34 ° anniversario.

Biglietti rimanenti : Caricamento

Si prega di prendere parte al sondaggio prima:

Domanda 1: Hai mai viaggiato con noi?

Sì

Non

Mi Piace Commenta Condividi

e altri 65.059

Figure 31 Example of a scam fraud targeting the customers of airline operators.

Annex A5.1: Sizing and Processing Capabilities

The internal architecture of AISafe DNS and the highly efficient approach to the traffic analysis it implements internally, make the solution not greedy in terms of HW resources required. Thus, it is possible to monitor even services belonging to entities receiving large amounts of traffic (e.g., ISPs, Public Administrations, Large Companies, Universities) with limited amount of resources.
Nevertheless, whilst the sizing is not critical, we do provide below here a qualitative indication of the minimum amount of resources required per installation type, separating them in three different sizes (Small, Medium, Large) and providing also an indication of the type of user which may correspond to each size.
It is in addition worth to remark that being AISafe DNS a "passive" component (it observes the traffic in real time, but it is not in-line with DNS servers) no issues on the availability of the services may arise from an insufficient amount of resources assigned to the Attack Prophecy virtual appliance.

- Type of Installation: **Small-Medium**
  - **Recommended CPUs:** 8
  - **RAM Memory:** 8GBs
  - **Disk space required:** 200GB (depends on the retention configured)
  - **DNS queries per day (on average):** tens of millions.
  - **Type of user:** University Campus, Large Company

- Type of Installation: **Large**
  - **Recommended CPUs:** 8
  - **RAM Memory:** 16 GBs
  - **Disk space required:** 500GB
  - **HTTP queries per day (on average):** hundreds of millions
  - **Type of user:** ISPs

Annex A5.2: Deployment and Installation procedure

In this section will be presented the AlSafe DNS's deployment options and installation procedure.

Annex A5.2.1: Deployment Options

The deployment of the AISafe DNS solution within a given infrastructure is based on two components, named respectively AISafe Sensor and AISafe Perception. The AISafe Sensor, presented in Figure 32, is basically a probe, which while installed within the infrastructure hosting the recursive DNS services, is able to collect the traffic from the rDNSs servers. Traffic might be collected alternatively:

- feeding the sensor with .pcap traces; a tcpdump-like utility can be used for this purpose as well as a span-port configured directly on the switch.
- Feeding the sensor with the BIND[7] log files.
  The AISafe Sensor features a number of detection modules,
  which are run locally on the virtual appliance on which the sensor is installed. Alerts may be sent by the sensor directly to the Security Operation Center team, for instance by the means of a SIEM.

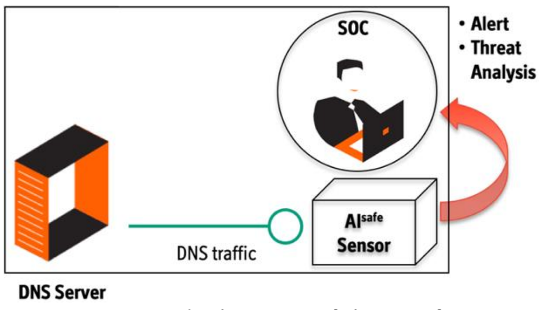


Figure 32 Basic deployment of the AISafe Sensor

A second deployment scenario is the one where the Sensor is also connected with the AISafe DNS Perception platform, as seen in Figure 33. The AISafe DNS Perception platform is a cloud platform entirely developed and managed by Pluribus One, where information on the attacks detected by the several installations of the AISafe Sensor is aggregated and then further elaborated. For certain categories of attacks (e.g. for the detection of Fast-Flux networks) this allows to deliver superior detection capabilities. It is worth to remark here that even in the presence of a connection with the Perception traffic, the raw DNS traffic is never sent out of the perimeter of the network where the Sensor is installed. The Sensor shares in fact with the Perception only statistical information (that is features useful for detection) by the means of a secure and authenticated API.

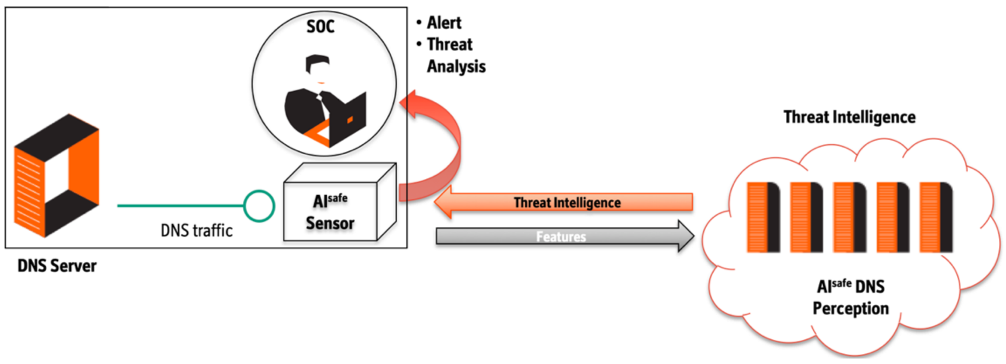


Figure 33 Deployment of the AISafe AISafe DNS Sensor coupled with the AISafe DNS Perception platform

[7] https://www.isc.org/bind/. Is the most widely used Domain Name System software on the Internet. On Unix-like operating systems it is the de facto standard.

Annex A5.2.2: Installation

The AISafe DNS installation is performed through a guided procedure, which leverages the packet manager available on standard Linux distributions. Pluribus One currently distributes an **.rpm** package suitable to install Attack Prophecy on Red Hat based distributions. The system has been widely tested on the CentOS 7 distribution, compatibility with which is fully granted by Pluribus One. Support to other Linux distributions, including Debian-based ones will be soon granted as well.
Packages are distributed using Pluribus One owns repositories, which can be then configured[8] among those used by the operating system for software download and update.
The setup procedure is performed through a graphical interface similar to that of Attack Prophecy and shown in Figure 25.

Annex A5.3: Detection Capabilities

The detection modules of AISafe DNS are the following: Domain Fluxing (DGA Malware), IP Fluxing (Fast-Flux Networks) and *Squatting Attacks.

Annex A5.3.1: Domain Fluxing (DGA Malware)

Domain Fluxing refers to attacks where domain names are algorithmically generated (possibly in a high number: this is the reason why we talk about domains flux). Modern malware achieves this capability thanks to a DGA (Domain Generation Algorithm), which dynamically generates domains names: generally speaking, the DGA generates a large number of domain names, and the malware then makes use of a small subset of them for actual C&C communication (as shown in Figure 34). This approach usually allows *botmasters* to harden the infrastructure of their botnets.

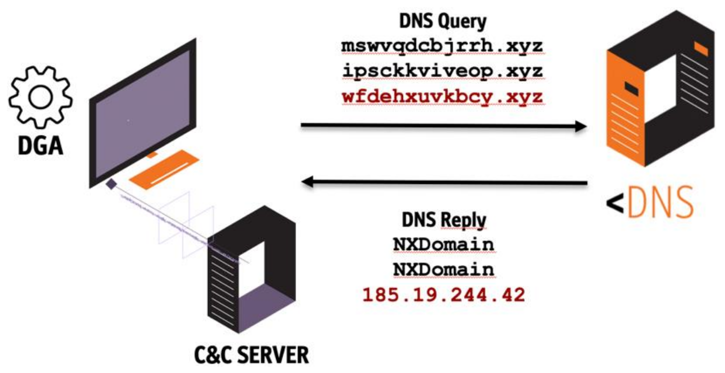


Figure 34 Behaviour of a DGA malware, which dynamically generates domain names to let the malware communicate with the C&C server of the botnet

Domains generated by DGA malware may exhibit a different layout. Typically, four different layouts are adopted:

- **Dictionary based domain names** (E.g. jeannettegeorgeson.net)
- **Alphabetic layout** (E.g. isrkvfkoxnwsatkdb.ki)
- **Numeric layout** (E.g. 9af4a478.net)
- **Alphanumeric layout** (E.g. txjcbx3oejncdvg4.com).

[8] Here https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/sec-configuring_yum_and_yum_repositories is an example of how to configure repositories for YUM, the package manager used by RedHat based distributions.

AISafe DNS features a module allowing the detection of DGA malware, identifying:

- the malicious domain name;
- the IP address(es) on which the domain has been resolved;
- the clients (victims) who have contacted it;
- the nameserver who served the resolution.

Annex A5.3.2: IP Fluxing (Fast-Flux Networks)

Fast-Flux Service Networks exploit an architecture which is conceptually similar to that of Content Delivery Networks. The key idea behind Fast-Flux networks is to construct a distributed proxy network on top of compromised machines that redirects traffic through these proxies to a central site, which hosts the actual content. In such a way several advantages are obtained by the attacker:

- taking down any of the proxies does not affects the availability of the central site (Mothership);
- the attacker always returns a different set of IP addresses for a DNS query and thus distributes the traffic over the whole proxy network;
- an increased resilience of the botnet is obtained, since taking down such schemes usually needs cooperation with a domain name registrar;
- the Mothership itself becomes really hard to track.

Such scheme may be leveraged to deliver a broad range of attacks, from malware distribution to phishing campaigns.

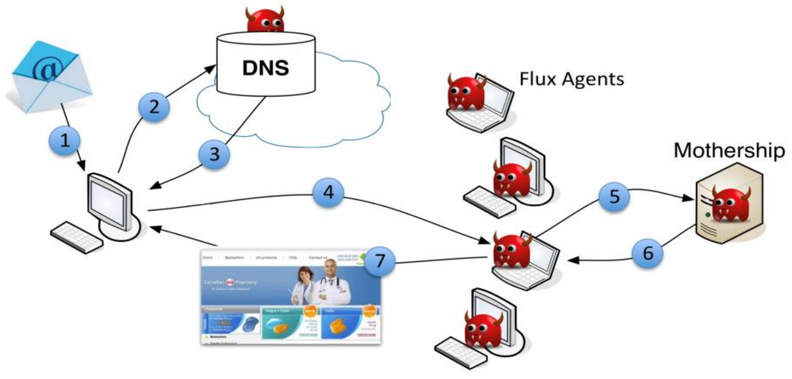


Figure 35 An example representation of how attacks are delivered through a Fast-Flux service network. Credit: Prof. Roberto Perdisci, UGA.

AISafe DNS features a module allows the detection of Fast-Flux networks by identifying:

- the malicious domain name;
- the IP address(es) on which the domain has been resolved (which are themselves victims in the case of Fast-Flux networks);
- the clients (victims) who have contacted it;
- the nameserver who served the resolution.

*Annex A5.3.3: *Squatting Attacks*

Cybersquatting[9] refers to the practice adopted by cyber-criminals of registering and using domains names that are confusingly similar to legitimate and well-known ones. Different techniques are used by the attackers to obtain malicious domains starting from legitimate ones, based on different kinds of alterations that may be introduced in the original name. We will generally refer to such techniques which all are listed in Table 2, as to *Squatting techniques. The general reason for using *squatted domains in cyber-attacks is that the attack becomes generally more credible (e.g., in the case of a scam or phishing campaign), they are generally less noticeable, and thus the overall effectiveness of the attack campaign increases.

Domain Name	***SQUATTING Type**
youtube.com	Original Domain
yewtube.com	Homophone-Based Squatting
youtubg.com	Bitsquatting
Y0UTUBE.COM	Homograph-Based Squatting
youtubee.com	Typosquatting
youtube-login.com	Combosquatting
youṭube.com	Homogliph-Based Squatting

Table 2: AISafe DNS - Domain Name and *Squatting Type

AISafe DNS features a module which allows the detection of *Squatting attacks, identifying:

- the malicious domain name;
- the IP address(es) on which the domain has been resolved;
- the clients (victims) who have contacted it;
- the nameserver who served the resolution.

Annex A5.3.4: Other detection capabilities

In addition to the detection capabilities listed above, AISafe DNS features also the capability to:

- Identify, based on their overall behaviour, the presence of infected clients within the network. In this specific case, only the client's IP address is reported as malicious;
- Identify command injection attempts. In command injections attacks infected clients generates unresolvable queries toward the DNS which contain for instance shell commands, like those for instance to download via scp or ftp malicious scripts on infected machines.

Annex A5.4: Support to Real Time Requirements

All the AISafe DNS detection modules operate in real-time against the DNS traffic. Thus, attacks can be reported externally as soon as they are spot by the system.

[9] A definition of cybersquatting is provided in the Anticybersquatting Consumer Protection Act – U.S. 1999:
"A person shall be liable in a civil action by the owner of a mark if […] '(ii) registers, traffics in, or uses a domain name that— '(I) in the case of a mark that is distinctive at the time of registration of the domain name, is identical or confusingly similar to that mark; […] '(III) is a trademark, word, or name protected by reason of section 706 of title 18, United States Code, or section 220506 of title 36, United States Code.

Annex A5.5: Support to Storage Requirements

Data persistence in AISafe DNS is granted by a COTS relational database. The database supports full text indexing of the recorded queries, as well as queries on time range.

The operator can easily inspect the data from the AISafe DNS dashboard, and eventually export it in a CSV file format. The underlying routine making this possible on the Attack Prophecy dashboard can be leveraged to transfer selected data to SIMARGL, offering thus support to the Storage Requirements.

Annex A5.6: Support to Supervision Requirements

Information regarding the AISafeDNS status is constantly available through the Dashboard, making possible to spot problems as soon as they arise as well as to provide a prompt diagnosis of their cause. Two widgets are available in the dashboard providing such information: the *Status* widget and the *Server Monitor* widget. The *Status* widget allows to verify that all the internal services of AISafe DNS are working properly. Namely, such services are responsible to retrieve the traffic from the source (*DNS Source* services), to store it in the internal database (*AISafe DNS DB* services), and to inspect it through the detection modules (*AISafe DNS Inspect* services). The *Server Monitor* widget (Figure 36) offers instead a perspective on the hardware resources (Disk, CPU, Memory) used by Attack Prophecy.
Such information can be automatically sent out to an external collector of information, for instance by calling an API made available by the collector.

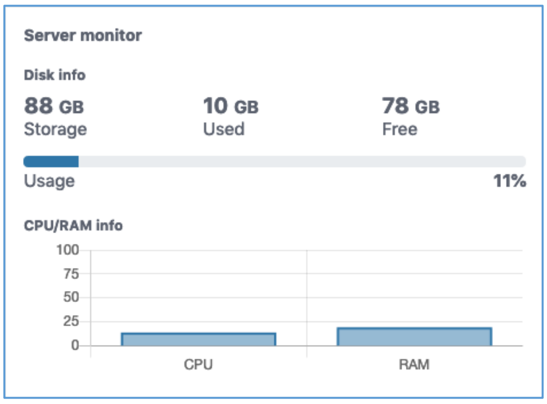


Figure 36 The AISafe DNS Server Monitor widget.

Annex A5.7: Integration of AISafe DNS with the SIMARGL platform

The most straightforward way to integrate the AISafe DNS solution with the SIMARGL platform is to leverage its built-in capability to communicate and share data with other platforms (such as Security Information and Event Management systems), by means of syslogs. The system already provides support for the following formats:

- Log Event Extended Formats (LEEF) – versions 1.0[10], 2.0[11]
- Common Event Format[12] (CEF) – versions 0.1, 1.0

[10] https://developer.ibm.com/qradar/wp-content/uploads/sites/89/2017/02/QRadar_LEEF_Format_Guide_V1.0.pdf
[11]https://www.ibm.com/support/knowledgecenter/SS42VS_DSM/com.ibm.dsm.doc/b_Leef_format_guide.pdf?origURL=SS42VS_DSM/b_Leef_format_guide.pdf
[12] https://community.microfocus.com/t5/ArcSight-Connectors/ArcSight-Common-Event-Format-CEF-Implementation-Standard/ta-p/1645557

Listing 8 presents an example of a syslog file in the LEEF format.

```
Oct 14 11:03:29 hawaii-aisafeDNS-test LEEF: 2.0|Pluribus One|Aisafe
DNS|1.0.1|DGA|cat=Detected count=6    description=Targeted probing for path:
/wp-login.php    devTime=Oct 14 2019 14:32:04.000 devTimeFormat=MMM dd yyyy
HH:mm:ss.SSS    proto=UDP DomainName=sdas434213opa.com sev=6
src=117.193.150.8 183.83.68.238 213.149.61.17 217.165.183.172 85.204.246.240
89.35.39.60
```

Listing 8: AISafeDNS - Syslog file in the LEEF format

The portfolio of supported syslog formats will be further extended since the next releases of the solution. Thus, the integration of AISafe with SIMARGL will potentially benefit from this improvement. Additionally, specific syslogs formats might be defined and implemented to support the particular needs of the SIMARGL platform.

Annex A6: BDE Platform

Adopting a conventional SIEM in a large organisation is challenging due to the growing volume of collected data and an increasing number of heterogeneous sources producing logs/alarms at various data rates. This leads to a situation where a human operator becomes overwhelmed by a huge amount of information, typically scattered across the entire system.
To address these challenges, ITTI has developed a highly scalable anomaly and cyberattack detection distributed system, which foundations constitute efficient and battle-proven technologies. The general architecture is depicted in Figure 37.

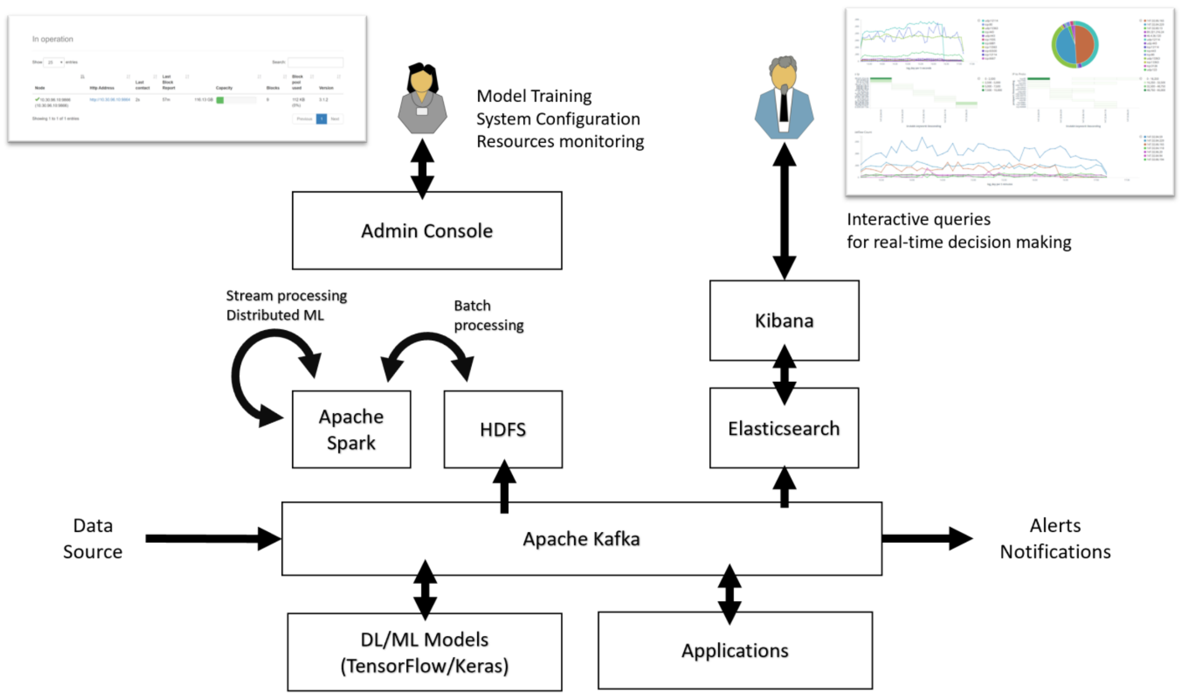


Figure 37 BDE Platform - General Architecture

The various modules are connected using apache Kafka - a distributed streaming platform. It serves two purposes: reactive, event-driven communication (instead of request-response approach) and real-time event processing.
Although the architecture enables various detection models to be connected to the platform (e.g., TensorFlow/Keras Models), currently, the majority of algorithms are implemented on Apache Spark system. This framework provides an engine that processes big data workloads. Apache Spark uses a data abstraction called resilient distributed dataset (RDD) to manage the data in a distributed way in the cluster and to implement fault-tolerance. Currently, we have implemented several modules allowing detection of malware and botnet presence based on the network traffic analysis.
Thanks to our dedicated Scala language library, which implements various software archetypes and adapters, it is possible to set up complex machine learning and pattern extraction pipelines using just a few lines of code.
Using various configuration schemas of the platform, the data can be curated in several ways using HDFS (Hadoop Distributed File System) or NoSQL databases (e.g., Elasticsearch). Depending on the nature of the data, it can be further queried in order to extract relevant patterns and perform visual analytics.

Annex A7: Honeynet (Thales)

The aim of the Honeynet is to provide a subnet with active nodes to lure attackers and make them use their attack tools against targets that have no real value.

To minimize the resource usage, the nodes will start only when they are needed. It means that when no traffic is sent to the Honeynet, it will consume fewer resources. A maximum limit will be created for the number of active nodes that can be created.

In order to provide such a service, the necessary resources are, for a small configuration:

- **Recommended vCPUs:** 16
- **RAM Memory:** 16 GBs
- **Disk space required:** 600GB

Depending of the target network architecture to create, some more resources may be mandatory. A trade off will be made between the depth of the network to create and the resources that can be allocated.