ANASTACIA

Advanced Networked Agents for Security and Trust Assessment in CPS/IoT Architectures

D4.4

Final Monitoring Components Services Implementation Report

Distribution level	PU
Contractual date	<31.10.2019> [M34]
Delivery date	<30.10.2019> [M36]
WP / Task	WP4
WP Leader	MONT
Authors	D. Rivera (MONT), E. Cambiaso (CNR), I. Vaccari (CNR), J. Villalobos (ATOS), Miloud Bagaa (AALTO)
EC Project Officer	Carmen Ifrim carmen.ifrim@ec.europa.eu
Project Coordinator	Softeco Sismat SpA Stefano Bianchi Via De Marini 1, 16149 Genova – Italy +39 0106026368 stefano.bianchi@softeco.it
Project website	www.anastacia-h2020.eu

ANASTACIA has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement N° 731558 and from the Swiss State Secretariat for Education, Research and Innovation

Table of contents

PUBLIC SUMMARY

This document represents the final deliverable of the task 4.1 – Monitoring Enablers. It contains description about the changes introduced in the ANASTACIA Monitoring Module during the second phase of the project, both at architectural level and at components level.

First, the Montimage Monitoring Tool (MMT) was further extended to detect SlowDoS attacks. These attacks try to exhaust the connection resources of the server victim and reaches the Denial of Service condition by not allowing any new clients to open new connections to the server. Protocols based on requests and responses (such as HTTP and FTP) might be vulnerable to such attacks, being its detection a non-trivial task, due to the inability to reconstruct whole requests or responses in order to analyse them. The MMT software tackled this problem by implementing an EFSM-based detection rule that measures the partial times taken to transfer the data (request or response) between the client and the server.

Secondly, the Data Analysis component has also be extended to actively monitor the measured data from IoT sensors. This is achieved by learning a constraint-based decision model in three main stages: (1) building a first model; (2) learning the best model, and (3) verifying the model. These stages are supported by constraint-based programming in order to select the best set of features that allow effectively detecting the desired attack.

Finally, a novel component was introduced in the general design of the monitoring module: The *Resource and QoS Monitoring* component. This component has been introduced to provide the Security Orchestrator the ability to directly monitor the amount of resources being used. This is done with the aim of allowing the Security Orchestrator to correctly enforce QoS policies on the monitored platform.

All the aforementioned modifications along with this new member of the monitoring module, allows the ANASTACIA Monitoring Module to extend the support for novel attacks and scenarios during the second phase of the project.

1 INTRODUCTION

1.1 AIMS OF THE DOCUMENT

This document updates the previous D4.1 with the new advancement on the monitoring components that will be implemented for the ANASTACIA platform. Section 2 presents the principal developments organized in the following manner: Section 2.2 presents the general design of the Monitoring Module in the general ANASTACIA architecture, Section 2.2.1 presents the adaptations made on the Montimage Monitoring Tool (MMT) to detect SlowDoS attacks, Section 2.2.2 exposes the advances on the UTRC data analysis module that used machine learning techniques to detect misbehaviours on sensed data values, Section 2.2.3 presents the adaptations introduced in the XL SIEM component to support the new alerts and perform the evaluation of the different mitigation plans, and Section 2.2.4 explains the new Resource and QoS Monitoring component added to the Monitoring Module during the second iteration of the project. Finally, Section 3 presents the conclusions of this document.

1.2 APPLICABLE AND REFERENCE DOCUMENTS

This document refers to the following documents:

- D1.5 – Final Architectural Design
- D4.1 – Initial Monitoring Component Services Report
- D4.3 – Initial Agents Development Report
- D4.5 – Final Reaction Components Services Report

1.3 REVISION HISTORY

Version	Date	Author	Description
0.1	23/07/2019	MONT	Initial ToC
0.2	07/08/2019	CNR	Integrated CNR's contribution to ToC
0.3	06/09/2019	CNR	Added contributions to Section 2.2.1.1.
0.4	18/09/2019	ATOS	Added contributions to Section 2.2.2.
0.5	30/09/2019	MONT UTRC	Added contributions to Section 2.2 and 2.2.1. Added contributions to Section 2.2.3
0.6	10/10/2019	MONT	Added contributions to Sections 1.1, 2, 2.2, 3
0.7	18/10/2019	AALTO MONT	Added contributions to Section 2.2.4 Added Conclusions and Public Summary
0.8	21/10/2019	MONT	First Complete Version
0.9	30/10/19	AALTO	Final revision of the deliverable
1.0	31/10/19	MONT	Addressing revisor's comments and delivery version.

1.4 Acronyms and Definitions

Acronym	Meaning
CP	Constraint Programming
CSP	Constraint Satisfaction Problem
DDoS	Distributed Denial of Service
DoS	Denial of Service
EFSM	Extended Finite State Machine
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
IDS	Intrusion Detection System
LCM	Life Cycle Management
MiTM	Man-in-The-Middle
MMT	Montimage Monitoring Tool
SDA	Slow DoS Attack
SMTP	Simple Mail Transfer Protocol
SSH	Secure SHell
TCP	Transmission Control Protocol
VNF	Virtual Network Function
BMS	Building Management Systems

2 ANASTACIA MONITORING MODULE

The ANASTACIA Monitoring Module has been conceived to provide the platform with a set of monitoring components that allow the system operator to detect security breached and privacy issues.

The Monitoring Module has been further extended to provide support for novel threats on the ANASTACIA platform. The following sections describe the principal innovations introduced in the Monitoring Module at the design level of the module, and particularly in each on the components that form the ANASTACIA monitoring module.

2.1 ARCHITECTURAL DESIGN

During the second phase of the ANASTACIA development, the General Architecture of the platform has seen some changes. These modifications included changes in the Monitoring Module, which now allow the Security Orchestrator to have knowledge about the amount of resources being used and enforce QoS in the monitored network. Figure 1 shows the new design of the Monitoring Module as reported in D1.5.

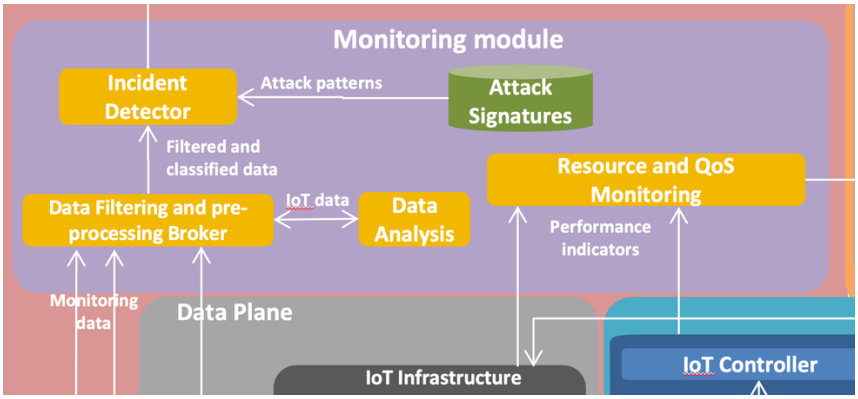

Figure 1 General Design of the ANASTACIA Monitoring Module

In this final design, the main four components of the Monitoring Module can still be recognized. Their main functions have not been changed and remain as explained in D4.1:

- Data Filtering and Pre-processing Broker: A common channel to transfer data between the monitoring agents and the monitoring module.
- Data Analysis: A behavioural-based analysis module used to detect anomalies in IoT sensors.
- Attack Signatures: Repository of attack signatures that the ANASTACIA platform will use to detect security threats.
- Incident Detector: A collector of all the extracted data that raises alerts once a security threat has been detected.

Along with these four components, a new one has been added:

- Resource and QoS Monitoring: A component that actively monitors the network and determines the level of resources used to enforce QoS policies.

The five components here presented have been implemented in the monitoring module. The details about the underlying technologies that bring these components to life are given in the following Sections.

2.2 IMPLEMENTATION DETAILS

Given the design presented in the previous Section, the ANASTACIA platform implements this design by means of introducing four monitoring components:

- Montimage Monitoring Tool: An DPI-based event-correlation security tool that analyses the network traffic on the monitored network.
- UTRC Data Analysis module: A machine learning-based module that continuously monitors and analyses the data extracted from sensors in order to detect anomalies in the measurements.
- ATOS XL SIEM: An event-correlation engine that processes the information received by the aforementioned detectors, perform a correlation of these events at a higher level, and evaluate the impact and risks of the different mitigation strategies.
- Resource and QoS Monitoring: An extension of the Security Orchestrator module that takes actively monitors the used resources in order to support the decisions taken in the Security Orchestrator.

The following subsection gives details about the implementation of the aforementioned components.

2.2.1 Montimage Monitoring Tool

The Montimage Monitoring Tool (MMT) is one of the principal security assets that allow the ANASTACIA platform to monitor the network flows and perform signature-based security analysis.

As described in D4.1, the MMT software uses a library of rules that define: (1) security properties that need to be respected at all times, or (2) attack signatures, defining the conditions and trigger network events that need to be produced to detect an attack. During the second iteration of ANASTACIA, the MMT software was extended to support two new detection rules aiming to support the final case of study:

- A rule for detecting potential data leaks from an IoT network.
- A SlowDoS attack detection rule.

The first rule aims to detect and prevent potential data leaks from the IoT network. It makes use of a whitelist that contains the IP addresses of the trusted BMS servers of the system. In this sense the IoT controllers are allowed to send information only to these servers, so any packet containing the CoAP or DTLS protocol that does not go to one of the specified servers might be a potential data leak.

Being this said, Montimage has implemented a rule that keeps this whitelist of IPs, examines the protocols contained on the network packets, and extracts the destination IP addresses of these. Using this information, the MMT-Security is capable of determining if the flow uses the involved protocols and raise an alert in case the information has not been sent to a trusted network. In this case, MMT will raise an alert and trigger the whole ANSTACIA reaction process, applying and appropriate mitigation for this potential security and privacy leak.

The second rule is one of the main innovations of the monitoring module. Montimage has worked with CNR in order to develop a SlowDoS detection algorithm and implement it in the MMT software. The following sections give more details about this development.

2.2.1.1 CNR Slow-DoS Detection Algorithms

Among all the methodologies used to successfully execute malicious cyber operations, denial of service attacks (DoS) are executed with the aim of exhaust victim's resources, compromising the targeted systems' availability, thus affecting availability and reliability for legitimate users. These threats are particularly dangerous, since they can cause significant disruption on network-based systems. The term Slow DoS Attack (SDA), coined by the CNR research group involved in the project, concerns a DoS attack which makes use of low-bandwidth rate to accomplish its purpose [2]. An SDA often acts at the application layer of the Internet protocol stack because the characteristics of this layer are easier to exploit to successfully attack a victim even by sending it few bytes of malicious requests. Moreover, under an SDA, an ON-OFF behaviour may be adopted by the attacker, which comprises a succession of consecutive periods composed of an interval of inactivity (called off-time), followed by an interval of activity (called on-time). Such behaviour is possible since specific server-side timeouts are exploited by the attacker [4].

In order to protect from Slow DoS Attacks, it is important to consider the following fact: *it is trivial to detect and mitigate a single attacking host, while it is extremely difficult to identify a distributed attack*. This fact derives from the fact that IP address filtering may be applied to detect and mitigate an SDA (see, for instance, tests on mod-security [4]), while in case of a distributed attack this concept may not be adopted with ease. Moreover, from the stealth perspective, if we consider Slow DoS Attacks like SlowComm or Slowloris, they are particularly difficult to detect while active, since log files on the server are often updated only when a complete request is received or a connection is closed: being our requests typically endless, during the attack log files do not contain any trace of attack. Therefore, different approaches should be adopted. In accordance to ANASTACIA deliverable D2.8, a possible slow DoS detection approach makes use of the concepts described in [3] to extract data able to characterize a Slow DoS Attack. Such intrusion detection framework may be applied with different algorithms like the one proposed in [1].

In particular, the model is specific to request-response protocols [3] acting at the application layer and making use of the TCP transport protocol. At this layer, the protocol is based on messages exchanged between two entities, commonly known as client and server. Some examples of request-reply protocols are HTTP, FTP, SMTP, SSH. If we consider such protocols, in normal conditions, after the client/server connection is established, the client sends a request to the server. Hence, the request is interpreted by the server in order to generate a response to send back to the client. After the first request-response exchange, two possible events could characterize the connection:

1. The connection is closed;
2. The connection is kept alive (we talk in this case of persistent connection), to reduce the connection overhead for any additional request-response between the same client/server pair.

By considering each TCP connection, we define a *connection slot* as the portion of a connection which refers to the time passing between the start of a request and the end of the relative response on the same stream. Hence, for each TCP connection, we first extract the number of connection slots, plus the delta_start value [3]. Hence, for each connection slot, we extrapolate the remaining delta values described in [3], along with other TCP values related to packets number and size, according to the assumption reported in the same scientific article.

Hence, at the end of the process, we have two separate tables, respectively known as connections table (including the number of connection slots and the related delta_start value, for each TCP connection) and connections slots table (referring to a set of connections slots, characterized by the delta values and other selected values computed from TCP packets). These tables are built by analysing the packets that transverse the network in the following manner: (1) the packet is associated to its TCP connection by using the IP addresses and the port numbers, (2) the connection slot is identified by either analysing the content or the direction of the packet, and (3) the statistics for the connection slot are updated using the metadata of the packet.

After such tables are derived, through live traffic analysis, the data analysis process is triggered. The adopted data analysis algorithm is based on the work reported in [3], making use of statistical properties to compare a potentially anomalous condition, obtained from live traffic data, with a pre-computed condition that is considered representative of all the legitimate conditions. In this case, through the 3-sigma rule [1], it is possible to identify relevant variations of selected delta parameters from normal situations, hence identify an attack and flag it as anomalous traffic.

2.2.1.2 Implementation of SlowComm Detection Algorithm in MMT

The last section describes a SlowDoS algorithm that can be effectively used by analysing the values of the TCP packets. To this end, an implementation of the connection slots table was implemented for the MMT software. The following sections go deeper about how data is extracted from TCP flows, how the connection slots table is maintained and how its data is used to detect SlowDoS attacks.

2.2.1.2.1 Network Traffic Caracterization

As mentioned before, the detection algorithm presented in [3] makes some assumptions about the type of the flows that can contain a SlowDoS attack:

1. Usage of TCP protocol: This assumption allowed us to use metadata about the TCP protocol in order to detect the start and end of both requests and responses.
2. Communication based on requests and responses: By assuming there is a request for each response, it is possible to model the communication as a state machine, whose actual state changes depending on how the flow behaves.
3. Absence of overlapping connection slots on the same flow: This simply means that there might not be a second request as long as the first one has not been answered by the server. In the case of the HTTP protocol, the application of this technique is called "request pipelining" (see Section 6.3.2 in [5]), and despite most of the major support this feature, its client-side implementation has been removed from the major browsers of the market[1]. It is important to remark that other protocols (in particular HTTP/2) might support request pipelining, which makes the detection technique more complex.

Considering the nature of the network traffic that will be analysed (an attack using HTTP 1.1 protocol), the aforementioned assumptions allow us to simplify the problem in order to propose a baseline for further extensions of the proposed implementation.

2.2.1.2.2 Modelling Network Flows

By combining the first two assumptions mentioned in the last section, we can model the TCP connection as an Extended Finite State Machine (EFSM) [6], triggering its transitions depending on some characteristics of each TCP packet: the flags, the size of the payload, the direction in which the packet goes (client to server or vice versa). It is also noticing that these triggering transitions also depend on the flow's last packet for the same aforementioned values.

Moreover, by adding the third assumption and avoiding analysing pipelined connection slots, we can merge both of the table used to detect the attack into a single one. This is due to the fact that one connection will be established (thus computing a delta_start value) and then a single request-response pair will be transmitted before sending the next one. This allows keeping a single entry on the table per TCP flow, on which all the subsequent slots (i.e. request-response pairs) will share the same delta_start value.

Being this said, the table will contain the delta values only for the latest connection slot (request-response pair), and it will override the older values if the client decides to reuse the network flow to transmit a new connection slot.

Figure 2 shows the EFSM model of a TCP flow whose transitions are triggered by using the metadata of the TCP packets. To keep the figure clear, the states representing the closing of the connection are not shown, along with the transitions that lead to these states. It is also important to remark that this machine models the state of a single TCP flow, so an implementation will require to maintain an instance of this machine for each TCP flow detected.

[1] Internet Explorer: https://blogs.msdn.microsoft.com/ie/2005/04/11/internet-explorer-and-connection-limits/
Chrome: https://www.chromium.org/developers/design-documents/network-stack/http-pipelining
Mozilla: https://bugzilla.mozilla.org/show_bug.cgi?id=1340655

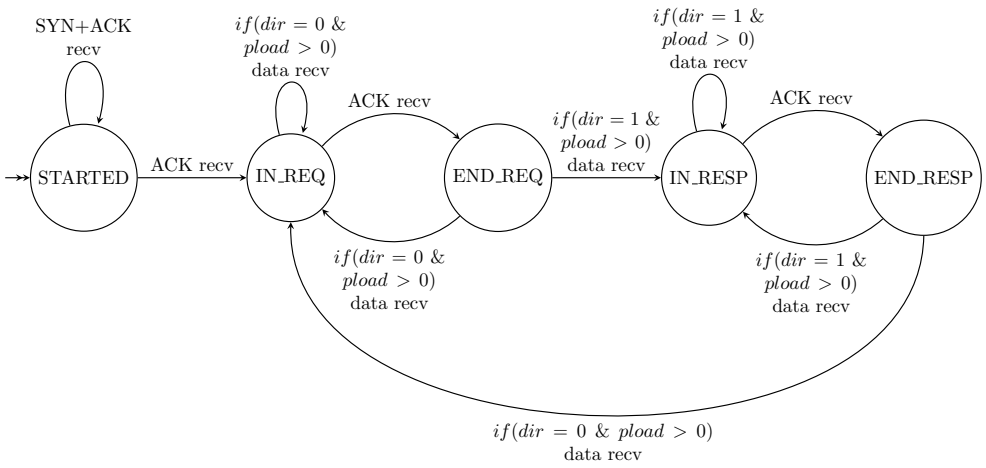

Figure 2 EFSM used to model a TCP-based request-response protocol.

To keep the figure clean, some of the information has been removed from the diagram: (1) each transition of the machine consumes a packet of the TCP flow; (2) depending on some metadata of the packet (TCP flags, payload size, direction of the flow), the machine decides the next state; (3) two states are not shown "CLOSING" and "CLOSED", which are used when the connection finishes or resets; (4) all the transitions going to the closing states are also hidden (each state two transitions that are executed with FIN and RST packets, leading to the state not shown); (5) the updates of the context variables (deltas) are also not shown, which are updated at each triggered transition; (6) after updating the context variables, the machine will emit a verdict depending on the values of the deltas; these outputs are also not shown.

The EFSM presented is a simplified version of the TCP state machine presented in [7], that has been adapted to support two types of "established" states; these new states represent the transfer of a request or a response, respectively, which helps measuring the times of the deltas for the current connection slot. These modifications go in line with the assumptions used to simplify the detection algorithm.

2.2.1.2.3 Design of the MMT Rule for the Detection Algorithm

The EFSM presented above follows the TCP flow in a per-packet basis, using metadata from each packet to keep track of the status of the network flow. To fulfil this requirement, the detection algorithm was implemented as an MMT attack rule, taking advantage of the MMT-DPI engine to extract the required metadata of each TCP flow. This rule was designed to analyse each TCP packet, keeping track of all the flows concerned. Table 1 shows the MMT rule that implements the detection algorithm.

Table 1 General schema of the SlowDoS detection rule

```
<beginning>
        <!-- Property 77: SlowComm DDos attack -->
        <embedded_functions>
        <!—Code used to emulate the EFSM execution -->
        </embedded_functions>
        <property value="THEN" delay_units="mms" delay_min="0" delay_max="+0" property_id="77"
type_property="ATTACK" description="SlowComm DDos attack">
                <!-- Detect any TCP packet as declared in the IP headers -->
                <event value="COMPUTE" event_id="1" description="Any TCP packet"
boolean_expression="((#is_exist(tcp.p_payload) != 2) &amp;&amp; (ip.proto_id == 6) &amp;&amp;
(ip.src != ip.dst))"/>
                <!-- Evaluate the values of the deltas using the data form the last event -->
```

```
            <event value="COMPUTE" event_id="2" description="Evaluate deltas"
boolean_expression="(#em_check_delta_request(ip.session_id.1, meta.utime.1, tcp.flags.1,
tcp.p_payload.1, tcp.payload_len.1, meta.direction.1))"/>
        </property>
</beginning>
```

The rule is composed of two events. The first (context) validates that the detected packet contains the TCP protocol by means of detecting the "protocol_ID" field in the IP headers. The other two network events are used to ensure that the rule is triggered even in the absence of payload and to print the IP addresses. The second event is a simple invocation to the EFSM presented above using the metadata from the first network event: a session_id (automatically generated by MMT using IP addresses and ports), a timestamp, the TCP flags, the TCP payload, the TCP payload length and the direction of the packet. In this case, the function *em_check_delta_request* implements the detection algorithm and returns the verdict as computed by the EFSM.

2.2.1.2.4 Execution of the Detection Algorithm

The embedded function *em_check_delta_request* has been designed to keep track of each network flow, contain the execution of its respective EFSM instance, and return the verdict for each packet analysed. To this end, the function contains a list of structures that: (1) identifies the flow to which this entry belongs to using the MMT session ID; (2) captures the execution state of the machine; (3) maintains metadata of the last package received; and (4) keeps track of the delta values for the session ID.

For each TCP packet, the *em_check_delta_request* function is invoked, performing the following process:

1. Check the list of known flows by comparing the session ID given by MMT.
    a. If no flow was found, create a new entry in the registry, assigning the state of the connection as "STARTED" (since the packet is – most likely – a SYN packet).
2. If an entry was found, change the state of the machine according to the conditions shown in Figure 2.
3. Use the state of the execution and the given timestamp to update the corresponding delta value.
4. Check that the values of the deltas do not violate the specified thresholds:
    a. In case a value of a delta exceeds the threshold, return the corresponding verdict.

With this implementation it has been possible to detect the SlowComm attack as defined in [2], defining the implementation of the SlowDoS detection algorithm that will be used in the ANASTACIA instance.

2.2.2 UTRC Data Analysis Engine

UTRC data analysis engine learns a constraint-based decision model to identify compromised cyber-physical devices. A novel approach to identify attacker by making use of Constraint Satisfaction Problem (CSP) and declarative programming frameworks has been proposed. The main contributions are twofold, listed as follows: a) learning constraint-based model for capturing the normal behaviour of a given cyber-physical system, b) an approach that provides explanation when a potential anomaly is detected by reporting which constraints fails to satisfy the model, c) an approach to incorporate user-defined constraints that can be easily integrated with the constraints learn from the data. The developed behaviour engine can handle multiple advanced attacks of different types, such as MiTM and flooding. We experimentally demonstrate the utility of our work on real datasets of cyber-physical device. This framework states constraints over network packets, describing relations between several packets, and modelled using Constraint Programming (CP) methodologies to find those intrusions. Although there are some Intrusion Detection Systems (IDSs) which provide native methods, characterized from scratch to allow the description of outliers or anomaly behaviour, which is found across several network packets, most IDSs either use signatures that only involve network features or allow constraints between several network features. However, these constraints are achieved in a very basic and limited way, often using ad-hoc tools. In contrast, our constraints are dynamic and learned over the time.

As described earlier in D4.3, our constraint model is composed of *variables* – extracted features from the data, and *constraints* – representing the allowed occurrence of each feature value happening simultaneously at any given point in time. The data format of the features can be of several types: continuous, discrete, events (on/off), etc. We choose to represent our model in terms of continuous variables and binary relations expressed in the form of table constraint. In the case of continuous data type, we perform clustering to create discrete domains. Note that the concept of over/underfitting applies when we generalise the data using clustering to create intervals for the variable domains to build the model for normal behaviour. Each interval is denoted by a symbolic integer value and the domain is the set of such integer values.

Algorithm Description:

Generally, most of the anomaly-based intrusion systems build a model of the normal behaviour of the monitored system. The built model is then used to compare a currently observed behaviour to the normal behaviour to determine whether the current state is a normal or an anomaly mode. Similarly, we have followed three steps to get the best detection model: a) building an initial model, b) learning the best model, and c) verification of the best model. In the first step, we create an initial model capturing all features and all possible relations using a historical dataset that describes the normal behaviour of the monitored system. Later in the second step, we use another labelled dataset that captures a mixture of normal and anomalous behaviour in order to select the best sub-model in terms of performance. In the third step, we use another unseen dataset to verify the performance of the selected sub-model.

Learning best model:

In order to learn the best model, we use an unseen labelled dataset (validation data) which contains data describing the normal and abnormal behaviours of the system. This dataset can be obtained either by applying attack functions on unseen data describing normal behaviour to insert malicious behaviour or for example from simulations. No matter how the validation data is obtained it is a labelled dataset where for each point in time we know whether the system describes a normal or abnormal behaviour.

We evaluate this dataset using the initial model. Evaluation is done by predicting a verdict (normal/anomaly) for each time point in the dataset and comparing this to the ground truth described by the labels of the dataset. Once we evaluated the dataset, we build an optimization problem, minimizing the error between the ground truth and predicted verdicts such that the optimal sub-set of variables and relations from the initial model is identified. In other words, we try to select a sub-set of features and relations such that the error between ground truth and expected verdicts is minimum.

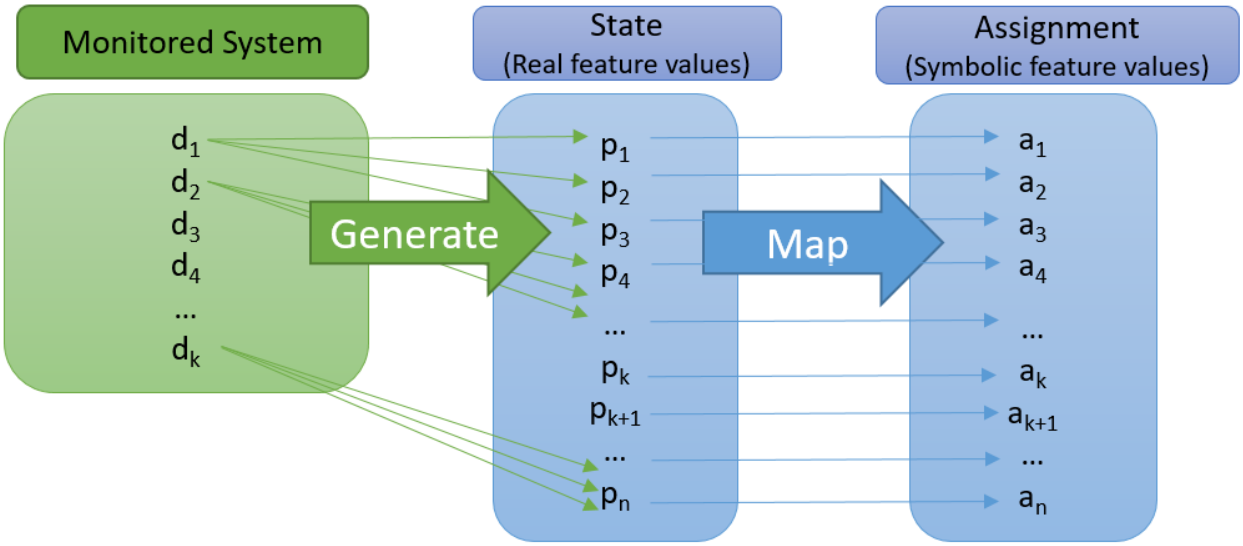

Figure 3 Dataset Evaluation Using the Initial Model

In order to evaluate a dataset, we follow the steps shown in Figure 3. In here k denotes the number of raw features and n denotes the number of raw and auxiliary features, thus k < n. Given a monitored system that has d_k raw features we observe a value for each raw feature for each time-point. Generate step shows the process of generating the required features of the model for each time point. We further refer to this as the state of the monitored system with respect to the model. This state consists of set of values for all features in a given point in time, S = {p_1, p_2, .., p_n} where p_i is the value associated to a feature for a given a time instant. An assignment A = {a_1, a_2, …, a_n}, where a_i is a symbolic label associated to a feature that is calculated for each state. This is done at the mapping step in Figure 3. An assignment has an assignment cost associated to it which consists of the sum of feature costs for each feature and relation costs for each relation for a given time-point. Once we obtain an assignment and its respective cost for all time-points in the validation data following the steps shown in Figure 3 the following steps are taken:

1) Create cost table: We collect all assignment costs for all time points in the validation data with respect to the initial model. Verdicts are obtained by comparing assignment costs to the threshold of the model. The cost table serves as a tabular data structure that is used to get the best model. The cost table contains cost information about all features and all relations in the initial model for every time point from validation data analysed as shown above.

2) Compare predictions with ground truth: Since we have a labelled dataset in order to learn from the predictions the initial model provided, we compare these predictions to the ground truth represented by the labels in the dataset.

3) Select best model: We use an optimization model to select the best model. The optimization model takes as input the following: cost table along with features, relations and threshold value of the initial model, the verdicts associated to each time point and the respective ground truth. It outputs a sub-set of features, relations and a threshold value, that will be used by the best model. A CP model is built to select the features and relations with the following objective: select the minimum number of features and relations between them, such that the difference between ground truth and the predicted verdict depending on the assignment costs of the sub-selection is minimal. The difference between ground truth and verdict is obtained using the notion of attack windows as described above. Once we obtain the learned threshold value and the set of features and relations, we build the best model. We use a third unseen labelled dataset to test the performance of the best model with regard to unseen data.

Once we obtain the best model, we deploy the learned model on monitoring agent. In the end, the developed monitoring agent is able to detect following attacks, a) MiTM, b) Flooding, and c) Slow DoS. However, for the project demonstration, we have deployed our monitoring agent from the perspective of MiTM attack only.

2.2.3 ATOS XL-SIEM

The XL-SIEM has been extended in the second iteration of the ANASTACIA platform in two ways:

1. Support to process and alert about additional attacks detected by the ANASTACIA probes deployed in the IoT infrastructure.
2. Integration with the Verdict and Decision Support System (VDSS) at the Reaction module to calculate risk levels associated to an incident.

The following subsections detail these advancements.

2.2.3.1 Integration of new attacks at the XL-SIEM

During the second iteration of the ANASTACIA platform the MMT probe has been extended with the capability to provide with evidences of anomalous activities with the support of detecting SlowDDoS attacks and detection of anomalous communication with untrusted destination that might infer a potential data leakage. To be able to process the new information the following activities has been carried out at the XL-SIEM:

- Adaptation of the MMT plugin at the XL-SIEM agent to process the new evidences.
- Adaptation of the XL-SIEM server with new correlation rules.

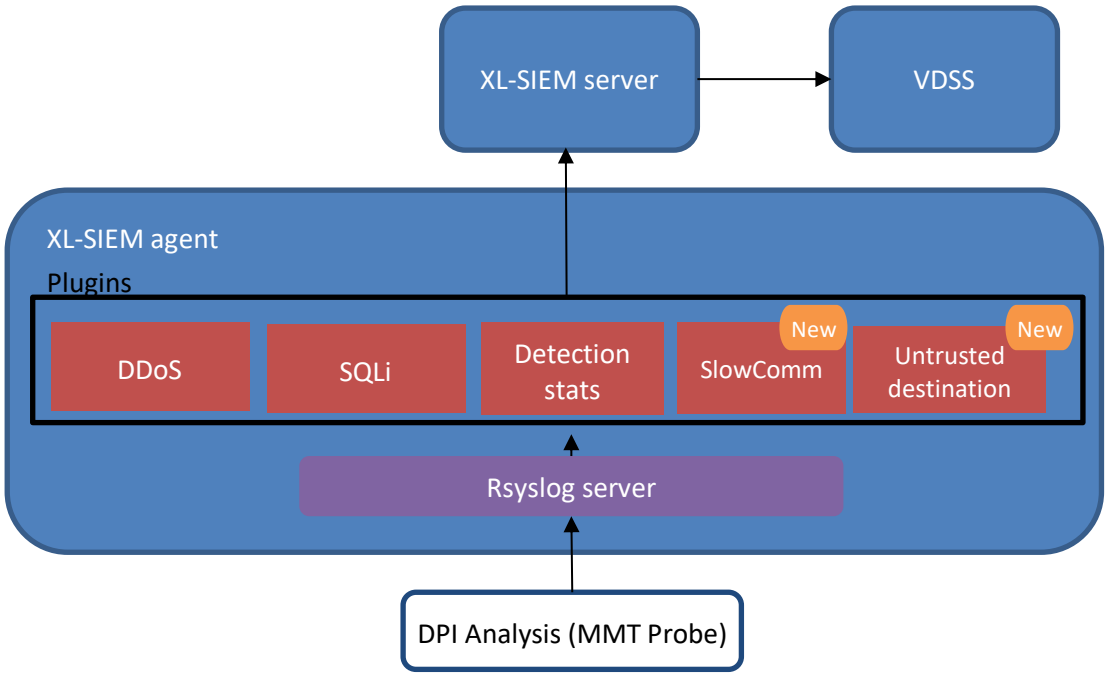

Figure 4. XL-SIEM plugin schema for the MMT Probe after the second iteration

Adaptation of the MMT plugin at the XL-SIEM agent to process the new evidences

As described in D4.1, the XL-SIEM task is to normalize the events received, through its rsyslog server, in a common format that is processable by the XL-SIEM server. The events normalized by the XL-SIEM agent is a JSON which is described in Table 1 of D4.1. Among others, it contains the source IP of the identified event, the destination IP of the targeted device or the date of the event. It is important to highlight the plugin_id and plugin_sid, which identifies the type of message so that it can be understood by the XL-SIEM server. Custom fields (called user data) allows to insert additional information contained in the event that is worth to be preserved and transmitted to the XL-SIEM server.

In order to know what the information is to extract from the event received we need to analyse the taxonomy of the event and the information contained. New events have been introduced in the second iteration of the development of the ANASTACIA platform. The following is an example of event received from the MMT Probe representing a SlowDoS incident.

```
2019-09-18T08:05:05.019Z hephasto.local MMT-Probe - - -
{"reportType":"security","probeID":3,"source":"/home/diego/capture_SlowComm.pc
ap","timestamp":1.568726462E9,"propertyID":77,"verdict":"detected","securityTy
pe":"attack","cause":"SlowComm DDos
attack","sourceIP":"150.145.21.204","destIP":"150.145.11.130","sourceMAC":"AA-
AA-AA-AA","destMAC":"BB-BB-BB"}
```

The following table represents the matching between the fields received in the MMT probe and the normalized event fields:

Table 2. Matching of fields between MMT Probe and normalized event for "SlowComm" events

MMT Probe Event field	Normalized event field	Value in example

not applicable	plugin_id	32000
not applicable	plugin_sid	133
2019-09-18T09:08:05.019Z	date	2019-09-18T09:08:05.019Z
sourceIP	src_ip	150.145.21.204
destIP	dst_ip	150.145.11.130
sourceMAC	userdata1	AA-AA-AA-AA
destMAC	userdata2	BB-BB-BB
verdict	userdata3	detected
probeID	userdata5	3
source	userdata6	/home/diego/capture_SlowComm.pcap
propertyID	userdata7	77
timestamp	userdata8	1.568726462E9
cause	not applicable. Used to identify type of event	SlowComm DDos attack

The normalized event results in the following JSON:


```
2019-09-18 09:08:05,019 Output [INFO]:
{"event":{
       "type":"detector",
       "date":"1568797685",
       "device":"10.0.2.4",
       "interface":"enp0s3",
       "plugin_id":"32000",
       "plugin_sid":"133",
       "src_ip":"150.145.21.204",
       "dst_ip":"150.145.11.130",
       "userdata1":"QUEtQUEtQUE=",
       "userdata2":"QkItQkItQkI=",
       "userdata3":"ZGV0ZWN0ZWQ=",
       "userdata5":"Mw==",
       "userdata6":"L2hvbWUvZGllZ28vY2FwdHVyZV9TbG93Q29tbS5wY2Fw",
       "userdata7":"Nzc=",
       "userdata8":"MS41Njg3MjY0NjJFOQ==",
       "log":"U2VwIDE4IDA5OjA4OjA1IDEyNy4wLjAuMSBNTVQtUHJvYmUgLSAtIC0geyJyZXBvcnRUeXBlI
joic2VjdXJpdHkiLCJwcm9iZUlEIjozLCJzb3VyY2UiOiIvaG9tZS9kaWVnby9jYXB0dXJlX1Nsb3dDb21tLnBj
YXAiLCJ0aW1lc3RhbXAiOjEuNTY4NzI2NDYyRTksInByb3BlcnR5SUQiOjc3LCJ2ZXJkaWN0IjoiZGV0ZWN0ZWQ
iLCJzZWN1cml0eVR5cGUiOiJhdHRhY2siLCJjYXVzZSI6IlNsb3dDb21tIEREb3MgYXR0YWNrIiwic291cmNlSV
AiOiIxNTAuMTQ1LjIxLjIwNCIsImRlc3RJUCI6IjE1MC4xNDUuMTEuMTMwIiwic291cmNlTUFDIjoiQUEtQUEtQ
UEiLCJkZXN0TUFDIjoiQkItQkItQkIifSA=",
       "fdate":"2019-09-18 09:08:05",
       "event_id":"d9f311e9-a789-0800-27ea-052cd2e7c882"
}}
```


It is worth noticing that the userdata fields are coded using base64. The log field contains the original event received by the MMT Probe while the event_id uniquely identifies this event.

Similarly, for the event received from the MMT Probe representing a potential Data Leak, being it the following:

```
2019-09-18T09:08:04.513Z hephasto.local MMT-Probe - - -
{"reportType":"security","probeID":3,"source":"enp0s8","timestamp":1.568725931
E9,"propertyID":99,"verdict":"not_respected","securityType":"security","cause"
:"Data sent to untrusted destination - potential data
leakage","sourceIP":"10.0.3.90","destIP":"10.0.3.80","sourceMAC":"AA-AA-
AA","destMAC":"BB-BB-BB"}
```

The following table represents the matching between the fields received in the MMT probe and the normalized event fields:

Table 3. Matching of fields between MMT Probe and normalized event for "Unstrusted destination" events

MMT Probe Event field	Normalized event field	Value in example
not applicable	plugin_id	32000
not applicable	plugin_sid	144
2019-09-18T09:09:08.513Z	date	2019-09-18T09:09:08.513Z
sourceIP	src_ip	150.145.21.204
destIP	dst_ip	150.145.11.130
sourceMAC	userdata1	AA-AA-AA-AA
destMAC	userdata2	BB-BB-BB
verdict	userdata3	detected
securityType	userdata4	security
probeID	userdata5	3
source	userdata6	/home/diego/capture_SlowComm.pcap
propertyID	userdata7	99
timestamp	userdata8	1.568725931E9
cause	not applicable. Used to identify type of event	Data sent to untrusted destination - potential data leakage

The normalized event results in the following JSON:

```
2019-09-18 09:08:04,513 Output [INFO]:
{"event":{
      "type":"detector",
      "date":"1568797684",
      "device":"10.0.2.4",
      "interface":"enp0s3",
      "plugin_id":"32000",
      "plugin_sid":"144",
      "src_ip":"10.0.3.90",
      "dst_ip":"10.0.3.80",
      "userdata1":"QUEtQUEtQUE=",
      "userdata2":"QkItQkItQkI=",
```

```
      "userdata3":"bm90X3Jlc3BlY3RlZA==",
      "userdata4":"c2VjdXJpdHk=",
      "userdata5":"Mw==",
      "userdata6":"ZW5wMHM4",
      "userdata7":"OTk=",
      "userdata8":"MS41Njg3MjU5MzFFOQ==",
      "log":"U2VwIDE4IDA5OjA4OjA0IDEyNy4wLjAuMSBNTVQtUHJvYmUgLSAtIC0gey
JyZXBvcnRUeXBlIjoic2VjdXJpdHkiLCJwcm9iZUlEIjozLCJzb3VyY2UiOiJlbnAwczgiL
CJ0aW1lc3RhbXAiOjEuNTY4NzI1OTMxRTksInByb3BlcnR5SUQiOjk5LCJ2ZXJkaWN0Ijoi
bm90X3Jlc3BlY3RlZCIsInNlY3VyaXR5VHlwZSI6InNlY3VyaXR5IiwiY2F1c2UiOiJEYXR
hIHNlbnQgdG8gdW50cnVzdGVkIGRlc3RpbmF0aW9uIC0gcG90ZW50aWFsIGRhdGEgbGVha2
FnZSIsInNvdXJjZUlQIjoiMTAuMC4zLjkwIiwiZGVzdElQIjoiMTAuMC4zLjgwIiwic291c
mNlTUFDIjoiQUEtQUEtQUEiLCJkZXN0TUFDIjoiQkItQkItQkIifSA=",
      "fdate":"2019-09-18 09:08:04",
      "event_id":"d9f311e9-a789-0800-27ea-052cd29a77bc"
}}
```


Adaptation of the XL-SIEM server with new correlation rules

The next step in the process of processing new events is the configuration of the XL-SIEM server to trigger the corresponding alerts when those events are received. As described above, two important fields are used at the XL-SIEM server to identify the new events: the plugin_id, which identifies the probe that sent the event (in this case it is 32000 which corresponds to the events received from the MMT Probe) and the plugin_sid which identifies the type of event (in this case it is 133 for the SlowComm event and 144 for the untrusted destination event).

With this information new rules are created at the XL-SIEM to trigger alerts associated to those events. There are many possibilities to create those rules, adding conditions that allows to fine tune the generation of the alert to certain context, associated to aspects such as the value of the fields received, the number of events received in certain period of time, etc. In this case, the rules are created to trigger an alert when two events received from the same probe. The rule specifies that for a time window of 60 seconds, no alerts are repeated even if they are received with the same source IP. This is used to avoid repetition of the same alert. The rule also force that the events received must have the same source IP and destination IP in order to trigger an alert. This is done not to mix events associated to different targets, which probably might denote a different incident. The following table details the alert using the Esper notation used at the XL-SIEM:

Table 4. New rules created at the XL-SIEM server

Name	Rule	Priority	Reliability
MMT Probe - Potential data leakage detected	pattern [every-distinct(a.src_ip, 60 seconds) a=MMT_Probe_Untrusted -> b=MMT_Probe_Untrusted ((b.src_ip=a.src_ip) and (b.dst_ip=a.dst_ip))]	2	8
MMT Probe - Slow DDoS Attack Detected	pattern [every-distinct(a.src_ip, 60 seconds) a=MMT_Probe_SlowComm -> b=MMT_Probe_SlowComm ((b.src_ip=a.src_ip) and (b.dst_ip=a.dst_ip))]	5	9

The following screenshot represents the ATOS XL-SIEM dashboard showing the new alerts triggered when new events are received from the MMT Probe.

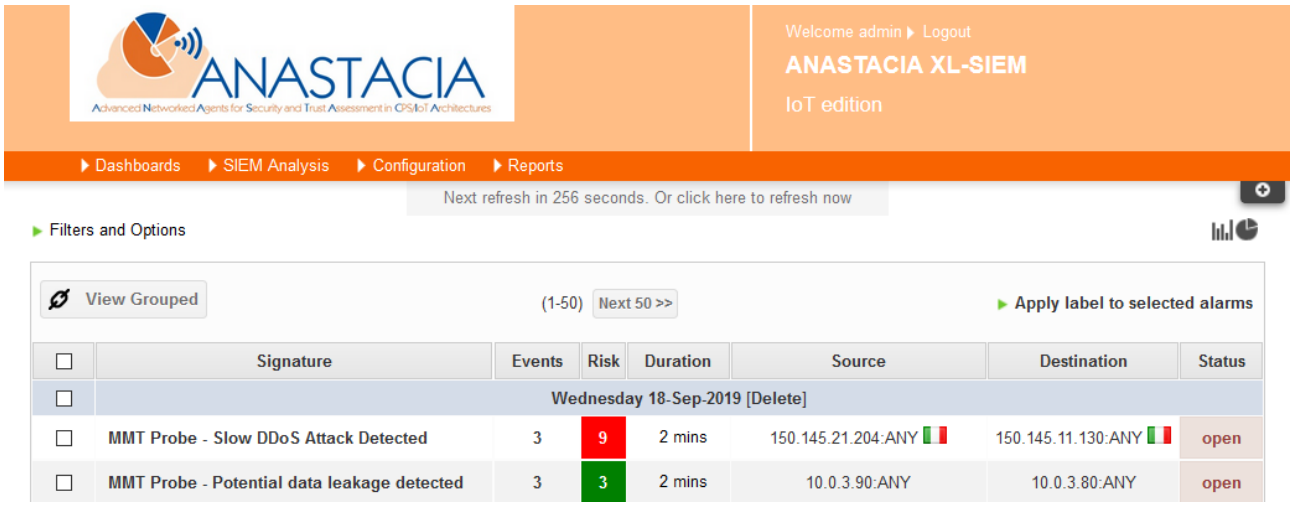

Figure 5. XL-SIEM dashboard showing alerts for the new events

2.2.3.2 Integration with the VDSS

As it can be seen in Figure 5, every alert there is a Risk level which is calculated by the XL-SIEM by using different values associated to the events received and to the device affected. As described in D4.1, this risk level is calculated using the following formula:

$$Risk = \frac{Reliability * Priority * Asset_{importance}}{25}$$

Reliability is a value that represents the accuracy of the detection, while the Priority represents the importance of the event received. Both values are given by the CISO using the XL-SIEM dashboard. The third variable is the Asset importance, which represents the criticality of the device affected by the attack. During the first iteration of the ANASTACIA implementation this value was inserted by the CISO using the XL-SIEM dashboard. After its integration with the VDSS (see D4.5), this value is calculated automatically based on the location and the type of device. The XL-SIEM server is capable of requesting to the Security Model Service this information, which is retrieved directly from the Orchestrator. The CISO might have set at the VDSS levels of importance to the different locations of the IoT infrastructure, and also to the different type of devices.

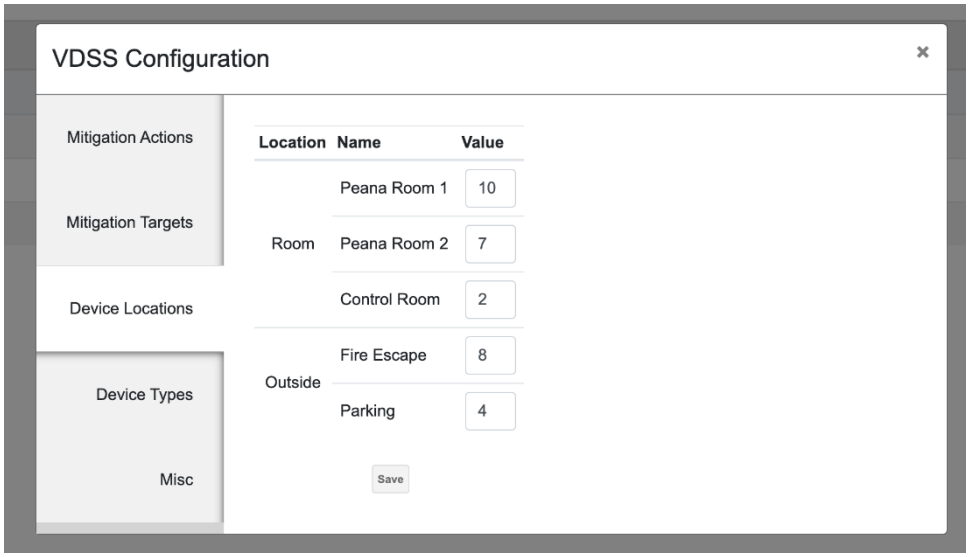

Figure 6. GUI used by the CISO to set location and device type importance levels

This information is used by the XL-SIEM to define the level of risk, resulting in the following formula. Being LI and DTI the Location Importance and the Device Type Importance scores respectively:

$$Risk = \frac{Reliability * Priority * Asset_{importance}\,(LI, DTI)}{25}$$

Being

$$Asset_{importance} = \frac{LI + DTI}{2}$$

With this we can obtain a more accurate risk level that is tailored to the characteristics of the IoT infrastructure to monitor. The alert generated by the XL-SIEM is sent to a RabbitMQ server (using the queue exchange_alarms). Further details of the rest of the process is given in D4.5 as part of the VDSS activities at the Reaction Module.

2.2.4 Resource and QoS Monitoring Component

The orchestration system provides an internal component, named "Resource and QoS monitoring", that serves to monitor the resource utilization at the network level in terms of end-to-end delay and bandwidth. This component also serves for monitoring different resources at deployed Virtual Network Functions (VNFs) in terms of CPU, RAM, and storage. The security orchestrator uses this component for ensuring life cycle management (LCM) of deployed services. The collected information enables the security orchestrator to ensure the efficient LCM of VNFs by either deploying or destroying VNFs according to the VNF and network states. This strategy helps for ensuring the KPIs in terms of security, delay, bandwidth, and QoS by mitigating the overload of VNFs and minimizing the cost by deploying the minimum number of VNFs at the right locations. In the literature, many tools have been suggested to monitor the network and the resources at different VNFs.

2.2.4.1 Open-Source Resource and Network Monitoring Tools

2.2.4.1.1 Nagios Core

Nagios[2] is one of the oldest and best monitoring systems that has been used to monitor the network. The open-source version of Nagios is Nagios Core that can enable different users to get real-time visibility about different hosts and services belonging to the network. This software also offers several alerts that could be used for optimizing and customizing the network and resources at each VNF. Nagios can monitor RAM, CPU and disk loads, and the number of currently running processes at each VNF. Nagios also can monitor services, such as SMTP, POP3, HTTP and other common network protocols used at each VNF.

2.2.4.1.2 Icinga

Icinga[3] is a tool that is developed as a branch from Nagios to offer more functionality to the Nagios Core suite. Icinga is designed to be easy to install and use. It offers more functionalities than Nagios for monitoring the network. This tool utilizes text-based configuration files for configuring different parameters. From the server, different agents would be installed at different hosts, and then the server periodically keeps monitoring the resources at each host.

2.2.4.1.3 Zabbix

Zabbix is one of the first tools that has been proposed for monitoring the network and services. This tool provides many out-of-the-box features that make it easy to manage and extend. The users do not have to deal with a glut of plugins. It can also offer real-time monitoring metrics across large-scale networks.

[2] https://www.nagios.org/projects/nagios-core/
[3] https://github.com/Icinga/icinga2

2.2.4.1.4 Prometheus

Similar to the previous tools, Prometheus is also an open-source system that offers the network and resource monitoring and alerting toolkit. Prometheus has been adopted by many companies and organizations to monitor their networks. Recently, in 2016, Prometheus joined the Cloud Native Computing Foundation. Prometheus has the following features. IT offers a multi-dimensional data model with time series data identified by metric names and key/value pairs. The time series collection happens via a pull model over HTTP. It adopts also a flexible query language, named PromQL, which offers more flexibility for retrieving different resource information. The hosts can be discovered using a service discovery or through manual configuration. Finally, it offers multiple modes of graphing and dashes boarding tools.

2.2.4.1.5 Psutil

Psutil is a Python library that provides process and system utilities for monitoring different hosts and networks. Psutil is a cross-platform library for retrieving information on running processes and system utilization (CPU, RAM, disk, network, sensors). It is useful mainly for system monitoring, profiling and limiting process resources and management of running processes. It implements many functionalities offered by UNIX command-line tools including ps, top, lsof, netstat, ifconfig, who, df, kill, free, nice, ionice, iostat, iotop, uptime, pidof, tty, taskset and pmap. It supports many platforms including Linux, Windows, macOS and others.

2.2.4.2 ANASTACIA QoS and Resource Monitoring Tool.

Using the Psutil library and Python language, we have developed the ANASTACIA QoS and resource monitor tool that monitors the resources at different VNFs and the delay and bandwidth of the network. To take the right decisions at the security orchestration plane, this component is leveraged for collecting the information about the resources of different deployed VNFs and the network in terms of end-to-end delay and bandwidth. When a VNF is instantiated, the security orchestrator injects an ANASTACIA QoS and resource monitor daemon into that instance, this daemon keeps sending the resource usage to the security orchestrator. The monitoring daemon keeps collecting in real-time the resources usage including the CPU usage percentage, memory details available, used and total. The monitoring agent also provides information about the network in terms of delay and bandwidth, as well as the security levels of different paths. We plan in the future to explore the ONOS API that exposes REST API for the net metering and JAVA API for QoS providing (Type, rate, and size).

3 CONCLUSIONS

This document presented the development advances of the Monitoring Module of the ANASTACIA platform. These changes are grouped into two principal axes.

First, the design of the Monitoring Module suffered minor modifications in order to support the addition of the new *Resource and QoS Monitoring* component. Since this new member has been conceived as an extension of the Security Orchestrator, it does not impact on the security monitoring functionality provided by the components already present in the module.

Along with the general design changes, the document also exposes the major changes on the four main members of the monitoring module:

- The Montimage Monitoring Tool (MMT): This software has been extended to support the detection of Slow DoS attacks. This was achieved by implementing an EFSM-based detection rule which analyses the network floes and follows the state of the client-server connection. To detect the attack, the rule keeps track of the partial times spent in transferring the data, raising an alert when the data transfer time exceeds a threshold.
- UTRC Data Analysis Engine: This component was extended with a refined model crafted to monitor the data measured form IoT sensors. The new model was build using constrained-based programming and following a three-steps training: first, a first model is trained using a historical dataset of normal behaviour, aiming to model all the possible parameters and relations among them; secondly, a better model is learned by means of refining the previous one using mixed datasets (containing both normal and tampered values), aiming to reduce the number of parameters used to identify the presence of an attack; finally, the model is validated using an third dataset
- ATOS XL SIEM: During the second part of the project, this component has seen two principal changes. On one hand, the XL SIEM was extended to support the security alerts corresponding to the novel attacks supported (e.g. SlowDoS). On the other hand, it has also been extended to integrate it with the Verdict and Decision Support System (VDSS) and calculate the risk levels associated with the detected security issues.
- Resource and QoS Monitoring: This Security Orchestrator extension was implemented to allow the SO ensuring the lifecycle management of all the deployed services. This component has been implemented using open-source monitoring tool, named psutil, inserting instances of this tools in any new deployed instance in order to monitor the used resources and ensure the QoS of the whole platform.

This deliverable finishes the activities of the T4.1, providing the ANASTACIA implementors with a simple yet flexible design, allowing them to include multiple security solutions into a single, federated security platform.

4 REFERENCES

[1] Aiello, M., Cambiaso, E., Scaglione, S., & Papaleo, G. (2013, July). A similarity based approach for application DoS attacks detection. In *2013 IEEE Symposium on Computers and Communications (ISCC)* (pp. 000430-000435). IEEE.

[2] Cambiaso, E., Papaleo, G., Chiola, G., & Aiello, M. (2013). Slow DoS attacks: definition and categorisation. *International Journal of Trust Management in Computing and Communications*, *1*(3-4), 300-319.

[3] Cambiaso, E., Papaleo, G., Chiola, G., & Aiello, M. (2016). A Network Traffic Representation Model for Detecting Application Layer Attacks. *International Journal of Computing and Digital Systems*, *5*(01).

[4] Cambiaso, E., Papaleo, G., & Aiello, M. (2017). Slowcomm: Design, development and performance evaluation of a new slow DoS attack. *Journal of Information Security and Applications*, *35*, 23-31.

[5] Fielding, R. & Reschke, Ed. (2014). Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. [Online. Accessed: 19/09/2019]. https://tools.ietf.org/html/rfc7230.

[6] A. Petrenko, S. Boroday and R. Groz, "Confirming configurations in EFSM testing," in *IEEE Transactions on Software Engineering*, vol. 30, no. 1, pp. 29-42, Jan. 2004. doi: 10.1109/TSE.2004.1265734

[7] Zaghal, R. Y., & Khan, J. I. (2005). EFSM/SDL modeling of the original TCP standard (RFC793) and the Congestion Control Mechanism of TCP Reno. *URL: http://www. medianet. kent. edu/technicalreports. html*.