European Commission | Horizon 2020 European Union funding for Research & Innovation

Cyber Security PPP: Addressing Advanced Cyber Security Threats and Threat Actors

Cyber Security Threats and Threat Actors Training - Assurance Driven Multi- Layer, end-to-end Simulation and Training

D5.3: The Simulation component IO module v1†

Abstract: This document is the result of the first iteration of task T5.4 activities. It defines the technical means and type of interfaces for interconnecting the Simulation Tool with the other platform components, namely the Training Tool, the Visualisation Tool, and the Data Fabrication Platform. The aim of the first version of the document is to guide the Simulation Tool's integration activities in the second year of the project and to ensure the proper interconnection with the other platform components.

Contractual Date of Delivery	31/08/2019
Actual Date of Delivery	31/08/2019
Deliverable Security Class	Public
Editor	*Hristo Koshutanski (ATOS)*
Contributors	*Oleg Blinder (IBM),* *Torsten Hildebrandt (SIMPLAN)*
Quality Assurance	*Dirk Wortmann (SIMPLAN),* *Fulvio Frati (UMIL),* *George Hatzivasilis (FORTH).*

† *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 786890.*

The *THREAT-ARREST* Consortium

Foundation for Research and Technology – Hellas (FORTH)	Greece
SIMPLAN AG (SIMPLAN)	Germany
Sphynx Technology Solutions (STS)	Switzerland
Universita Degli Studi di Milano (UMIL)	Italy
ATOS Spain S.A. (ATOS)	Spain
IBM Israel – Science and Technology LTD (IBM)	Israel
Social Engineering Academy GMBH (SEA)	Germany
Information Technology for Market Leadership (ITML)	Greece
Bird & Bird LLP (B&B)	United Kingdom
Technische Universitaet Braunschweig (TUBS)	Germany
CZ.NIC, ZSPO (CZNIC)	Czech Republic
DANAOS Shipping Company LTD (DANAOS)	Cyprus
TUV HELLAS TUV NORD (TUV)	Greece
LIGHTSOURCE LAB LTD (LSE)	Ireland
Agenzia Regionale Strategica per la Salute ed il Sociale (ARESS)	Italy

Document Revisions & Quality Assurance

Internal Reviewers

1. *Dirk Wortmann (SIMPLAN),*
2. *Fulvio Frati (UMIL),*
3. *George Hatzivasilis (FORTH).*

Revisions

Version	Date	By	Overview
0.6	27/08/2019	Editor	Addressed comments by FORTH from the internal quality review process
0.5	23/08/2019	Editor	Addressed comments by SIMPLAN and UMIL from the internal quality review process
0.4	22/08/2019	SIMPLAN, UMIL	Revision of message broker communications, and overall content
0.3	07/08/2019	Editor	ATOS' contribution to Sections 1, 3 and 5
0.2	01/07/2019	IBM	IBM's contribution to Section 4
0.1	20/05/2019	Editor	First Draft with ToC

Executive Summary

This document is the result of the first iteration of task T5.4 activities and reports the work performed by month 12 of the project. It steps on and extends the results of "D1.3 – THREAT-ARREST platform's initial reference architecture" to define the technical means and interfaces for interconnecting the Simulation Tool with the other platform components, namely the Emulation Tool, the Training Tool, the Visualisation Tool, and the Data Fabrication Platform.

The goal of this first version is to guide the Simulation Tool integration activities in the second year of the project and proper interconnection with the other platform components. Particularly, this document relates to the work package (WP) 6 activities on system integration starting in month 13 of the project.

Importantly, this document has two other counterpart documents – the deliverable "D2.4 – Emulation tool interoperability module v1" and the deliverable "D4.3 – Training and Visualisation tools IO mechanisms v1". These two other deliverables address in a similar but complementary way the interconnections of the other platform tools, and altogether provide an overall view of THREAT-ARREST platform interconnections for year 1 of the project. In addition, deliverable "D5.2 – Simulated components and network generator v1" describes how simulation components are instantiated based on the related Cyber Threat and Training Preparation (CTTP) sub-model.

Table of Contents

List of Abbreviations

AMQP Advanced Message Queuing Protocol

API Application Programming Interface

CoAP Constrained Application Protocol

CoRE Constrained RESTful environments

CSP Constraint Satisfaction Problem

CTTP Cyber Threat and Training Preparation

DFP Data Fabrication Platform

DTLS Datagram Transport Layer Security

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IEC International Electrotechnical Commission

IETF Internet Engineering Task Force

ISO International Organization for Standards

IoT Internet of Things

MQTT Message Queuing Telemetry Transport

OASIS Organization for the Advancement of Structured Information Standards

QoS Quality of Service

REST Representational State Transfer

STOMP Simple Text Oriented Messaging Protocol

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

VM Virtual Machine

WP Work Package

List of Figures

List of Code Examples

1 Introduction

This deliverable defines the technical means and interfaces for interconnecting the Simulation Tool with the rest of the platform components, such as with the Emulation, Training and Visualisation Tools, and the Data Fabrication Platform (DFP). The document is the first version of the means of communications that will be used as guidelines in the second year of the project to enable the Simulation Tool interconnect with the other platform components.

Simulation is an important part of a CTTP training session. In deliverable D5.2, the Jasima[1] discrete-event simulation library is described as the core engine of the THREAT-ARREST Simulation Tool. An important aspect of the Simulation Tool specification is the interconnection of the tool with the rest of the platform components, and its integration within the overall platform architecture and dataflow.

The Simulation Tool upon initialisation creates a simulation environment where the targeted cyber system components and functionalities are simulated. To do so, it offers a Representational State Transfer (REST) Application Programming Interface (API) for proper lifecycle management of cyber system simulation, such as initialisation and finalisation. The Training Tool initialises cyber system simulation by using the corresponding API. The Simulation Tool starts the initialisation by retrieving the instantiation scripts from the deployed CTTP models.

Upon successful cyber system emulation or simulation, the Training Tool interconnects with the Assurance Tool to initialise the monitoring of the trainee's actions against the actual cyber system of the organisation and get the necessary data for CTTP programmes' evaluation. The Assurance Tool may retrieve and update CTTP models stored in the platform for this purpose. The main role of the Assurance Tool is to monitor and assess the security posture of the actual cyber system of the organisation where the THREAT-ARREST training platform is used at. Given that, it has not been defined any direct communication between Simulation Tool and the Assurance Tool.

We note that this deliverable has two other counterpart documents – D2.4 (THREAT-ARREST D2.4, 2019) and D4.3 (THREAT-ARREST 4.3, 2019) – that altogether the three documents (D2.4, D4.3 and D5.3) provide the overall view of THREAT-ARREST platform interconnections for year 1 of the project. For convenience of readers and to facilitate material comprehension, we recall Section 2 across the three documents with the aim to have a more self-contained version of the documents.

The document is structured as following. Section 2 overviews the core means of communication supporting the various platform components – communications via either a message broker or REST interfaces. Section 3 presents in details the Simulation Tool interconnections with the other platform components, namely with Emulation Tool, with the Training and Visualisation Tools, and with the DFP. Section 4 overviews the functionality and API of the DFP, particularly the API for synthetic security event logs generation during a training session. Section 5 concludes the document and outlines next steps of activities.

[1] https://www.simplan.de/en/software-2/jasima/

2 Message Broker and REST Communications

This section describes the communication channels between the various platform components. Two main options are supported via either a message broker or REST interfaces (Fielding, 2000).

2.1 Message Broker-enabled Communications

Message-oriented protocols typically focus on providing asynchronous data transfers between distributed devices (Hatzivasilis et al., 2018a; Hatzivasilis et al., 2018b; Lakka et al. 2019). Their focus is on reliable messaging, including message buffers and Quality of Service (QoS) facilities, controlled by centralized entities. By using the message broker-enabled communication, messages are passed through a central server (the Broker), enabling *one-to-many* and *many-to-many* interactions. This offloads the computational power needed for a component to connect many different clients in order to exchange messages.

The Message Queuing Telemetry Transport (MQTT) (Banks and Gupta, 2014) is one such message-oriented protocol, introduced by IBM in 1999 and recently standardized by the Organization for the Advancement of Structured Information Standards (OASIS)[2], as the Internet of Things (IoT) developments brought it back into the limelight. It is also standardized as by the International Organization for Standards (ISO) and the International Electrotechnical Commission (IEC) as ISO/IEC 20922 (ISO/IEC, 2016). MQTT was designed as an extremely lightweight publish/subscribe messaging transport, for small sensors and mobile devices, optimized for high-latency or unreliable networks. A MQTT Broker is responsible for handling and organizing all communications between the various devices/components. Messages are published with specific *topics*, and each client can subscribe to various topics (though the Broker may require username/password authentication before allowing subscription). Topics are organized in a hierarchical manner, like the folder structure in a file system (e.g. "THREAT-ARREST/CTTP/models" could be a topic where a component can subscribe to get updates on the CTTP models). When a client publishes a message, the Broker then relays this message to all clients which are subscribed to the message's topic. Thus, all interactions are asynchronous and clients only communicate directly with the Broker. MQTT relies on the Transmission Control Protocol (TCP) and secure deployments support the use of the Transport Layer Security (TLS) protocol. The protocol is designed to be used even on lightweight devices, like mobile devices and embedded systems where bandwidth is costly and minimum overhead required. It uses a 2-byte fixed header to control everything and exchange data as byte stream. Therefore, MQTT is being used widely in IoT settings.

The Simple Text Oriented Message Protocol[3] (STOMP) is a simple text-based protocol with a main goal to interoperate with message-oriented middleware. The protocol wire format is suitable to allow any STOMP client to communicate with any message broker which supports the protocol. The protocol runs on any TCP-enabled communications following well-defined commands such as CONNECT, SEND, SUBSCRIBE, UNSUBSCRIBE, BEGIN, COMMIT, ABORT, etc. Importantly, STOMP is designed for *asynchronous* message passing between *lightweight* entities/clients coming from scripting languages such as Ruby, Python, Perl or JavaScript. In such a client environment, simple operations are typically carried reliably such as reliably sending single messages or consume messages on a given destination. STOMP can be seen as an alternative to other open messaging protocols, such as the Advanced Message Queuing Protocol (AMQP) (Luzuriaga et al., 2015), but covering a small subset of commonly used messaging operations. Given its deign principles, STOMP has been a definitive choice for

[2] OASIS: "MQTT 3.1.1 specification," December 10, 2015, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
[3] https://stomp.github.com/ ; http://stomp.github.io/stomp-specification-1.2.html

some THREAT-ARREST components' communications such as those of the Visualisation Tool.

The Advanced Message Queuing Protocol (AMQP) (Luzuriaga et al., 2015) is an open standard for passing business messages between applications or organizations. AMQP is designed for reliable communication via message delivery guarantee primitives, like *at-most-one*, *at-least-once*, and *exactly-one* delivery, and it is built upon a reliable transport protocol, such as TCP. The protocol consists of two core components that handle communication: the exchanges and the message queues. Based on pre-defined rules, the exchanges route the messages to appropriate queues, which can store the data and later send it to the receivers. It connects systems, feeds business processes with the information they need and reliably transmits onward the instructions that achieve their goals. The protocol is designed with more advanced features in mind and has more overhead than MQTT. For this reason, AMQP is not preferred for lightweight devices (e.g. mobile), while MQTT can be used almost anywhere. But in real world application development, we may need AMQP for reliable message queue while having lightweight devices to work with. Here is the point where RabbitMQ[4] comes in.

RabbitMQ (Richardson, 2014; Lonescu, 2015) is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols (e.g. MQTT and AMQP). It can be deployed in distributed and federated configurations to meet high-scale and high-availability requirements. This implementation can be run on a wide variety of platforms. RabbitMQ can potentially run on any platform that provides a supported Erlang[5] version, from multi-core nodes and cloud-based deployments to embedded systems. In particular, OpenStack supports RabbitMQ as message queue service and use it in many of its modules[6]. Figure 1 illustrates the basic steps for creating an application with RabbitMQ (Lonescu, 2015).

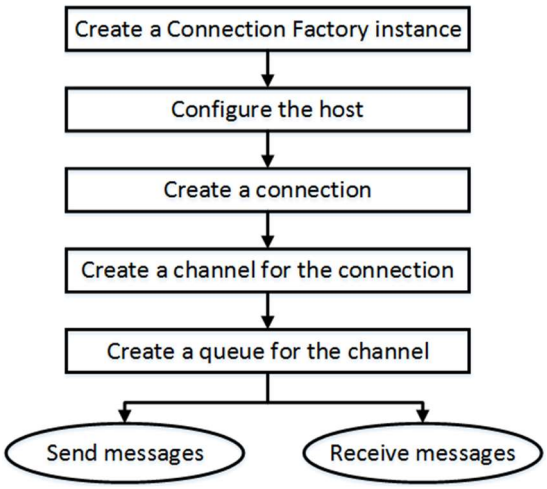

Figure 1 Basic steps to create an application with RabbitMQ

First, the components publish their profile information to the broker, including the relevant IP address. The broker can be either local or remote, enabling cross-domain interaction. In order to discover a component or service, the actuator sends a request message to all public components through the broker, which implements the corresponding multicasting functionality. The compatible entities respond to the request by sending descriptive metadata.

[4] RabbitMQ: http://www.rabbitmq.com
[5] https://www.erlang.org
[6] https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-messaging.html

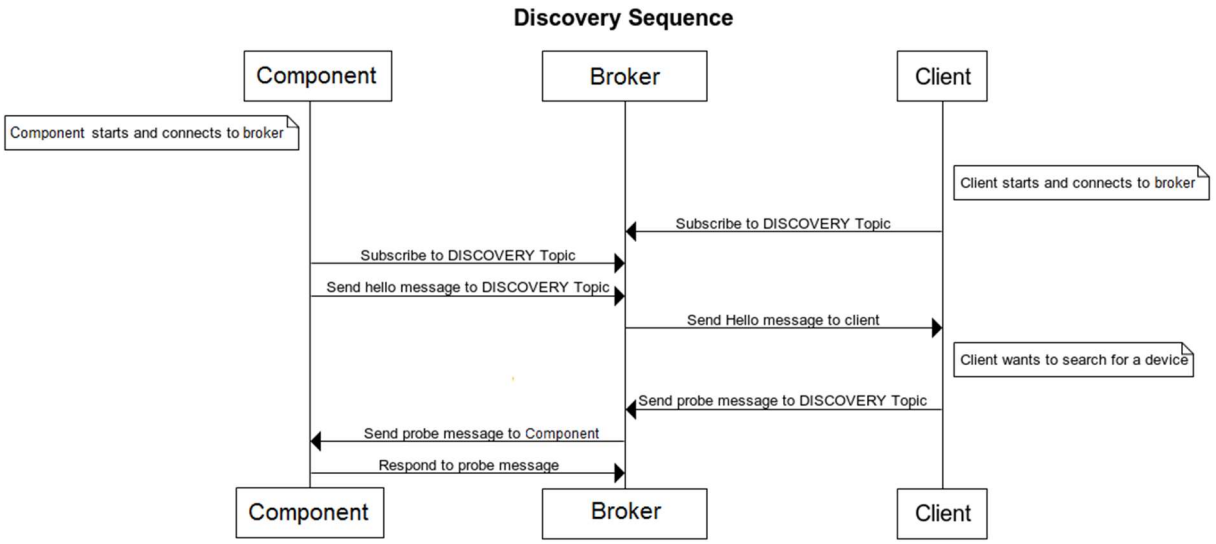

Figure 2 Sequence diagram for discovery operation

Figure 2 illustrates a sequence diagram of the discovery operation. For asynchronous operation, subscribe or eventing, the messages are passed through the broker. Figure 3 illustrates a sequence diagram of the event subscription operation.

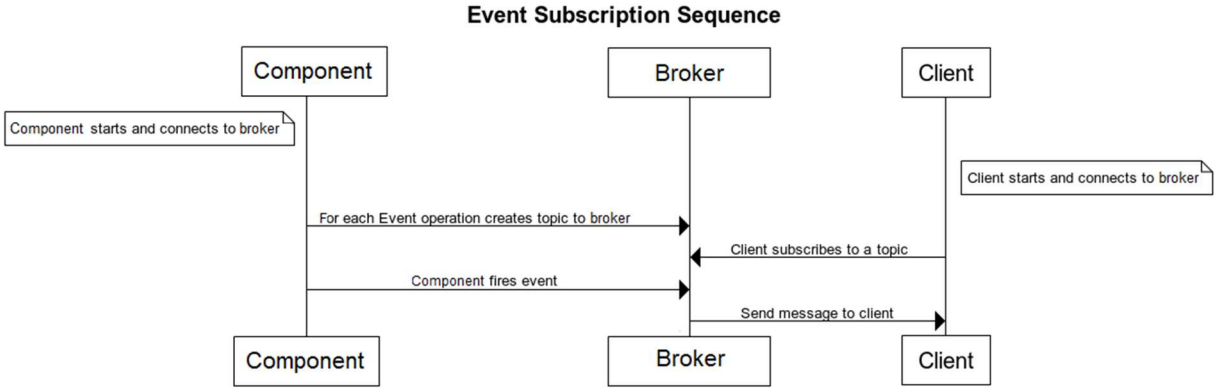

Figure 3 Sequence diagram for event subscription operation

RabbitMQ supports AMQP, MQTT, STOMP and WEBSOCKETS as message delivery protocols. This means that consumer and producer services can be implemented not only by using different platforms and languages, but also by different messaging protocols. It has a wide community and we can find a rich documentation on many different programming languages, such as Python, Java, PHP, JavaScript, Go, etc. (Richardson, 2014; Lonescu, 2015).

The most important features of RabbitMQ for the THREAT-ARREST project include the guaranteed delivery and the message queue implementation (Lakka et al. 2019; Hatzivasilis et al., 2019). To sum-up, we choose the RabbitMQ broker for the internal THREAT-ARREST platform communications, as:

- It is an open source message queuing system.
- It constitutes an ideal choice for interoperability between applications and tools of different protocols and between different programming languages.

- The fact that we can publish messages into one environment via one protocol and consume them via one or more other protocols (simultaneously if necessary).
- It is a popular open source message queuing system that implements the AMQP.
- It well describes all supported protocols and their purpose.
- There is an active community and RabbitMQ has been utilized in very different application areas.
- RabbitMQ offers libraries/APIs available in many programming languages[7] allowing, with just a few lines of code, the creation of communication channels to a broker, the creation of queues, and publishing and receiving messages on channels and queues respectively.
- It is fully supported by OpenStack[8] the underpinning technology of the THREAT-ARREST Emulation Tool.

2.1.1 Standard RabbitMQ Message Flow

In the following, we will overview the basic message flow concept of RabbitMQ to facilitate the presentation in the following sections. In RabbitMQ, the producer's messages are not published directly to a consumer but instead, the producer sends messages to an *Exchange*. An Exchange is a message routing agent responsible for routing of messages to different queues. An Exchange accepts messages from the producer application and routes them to message queues with the help of header attributes, bindings, and routing keys (Johansson, 2015).

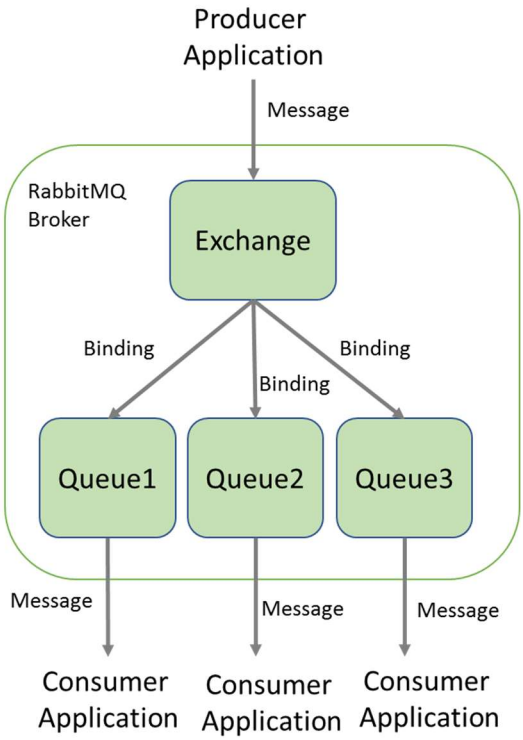

Figure 4. Standard RabbitMQ message flow

Figure 4 shows the standard RabbitMQ message flow. A producer application publishes a message to a given (selected) Exchange. When the Exchange receives the message, it is responsible for routing the message to an appropriate Queue(s). A Binding has to be set up

[7] https://www.rabbitmq.com/getstarted.html
[8] https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-messaging.html

between a Queue and a given Exchange. In our case, there are bindings to three different Queues from the given Exchange. The Exchange routes the message to the Queues according to the Bindings specified. The messages stay in a Queue until they are handled by a consumer application.

A Binding is a "link" that is set up to bind a Queue to an Exchange. A routing key is a message attribute set up by the producer that allows an Exchange to look at this key and decide how to route the message to Queues depending on the Exchange type.

There are four different types of Exchange that route messages differently using different parameters and bindings setups. The most relevant to the THREAT-ARREST needs is the Exchange of type *Topic*.

2.1.2 RabbitMQ Topic Exchange

A Topic Exchange routes messages to Queues based on wildcard matches between the *routing key* specified in the message header and the *routing pattern* specified by the Queue *binding* (Johansson, 2015). Given the routing pattern of each Queue binding, messages are routed to one or many Queues.

The consumer indicates in which Topics is interested in, such as subscribing to a feed of a specific THREAT-ARREST platform tool. The consumer creates a Queue and sets up a binding with a given routing pattern to the selected Exchange. All messages with a routing key that match the routing pattern are routed to the Queue and stay there until the consumer consumes the message.

The routing key is a period (‘.’) delimited list of words, such as *EmulationTool.ehealthscenario1.vm1* which identifies all events of cyber system emulation that are monitored at the Virtual Machine (VM) ‘*vm1’* of the eHealth scenario.

A routing pattern of a Queue binding can contain an asterisk ‘*’ to indicate a match of words in a specific position of the routing key. For instance a routing pattern for a Queue1 can be **.ehealthscenario1.** indicating all events from the cyber system of the eHealth scenario regardless of whether these are from simulation or emulation and regardless of what particular VMs they originated from.

A hash/pound symbol ‘#’ indicates match on zero or more words. For instance a routing pattern *EmulationTool.#* will match any routing keys beginning with *EmulationTool* resulting in capturing all events from cyber system emulation regardless of the specific scenarios currently used.

2.2 REST Communications

Nevertheless, except from the asynchronous communication through a broker, we also need synchronous communication options where the various modules can exchange data directly. Protocols that follow the REST architecture are adopted for this. RESTful implementations typically use the Hypertext Transfer Protocol (HTTP). In general, the REST solutions follow a request/response model, where a client may interact with the server using a subset of the HTTP methods, namely using GET, PUT, POST and DELETE on the server's resources (see Figure 5 below). RESTful systems target interoperability, performance and scalability for increasing resources consumption. They target reusability of components which can be managed or updated without affecting the system as a whole, even while it is running.

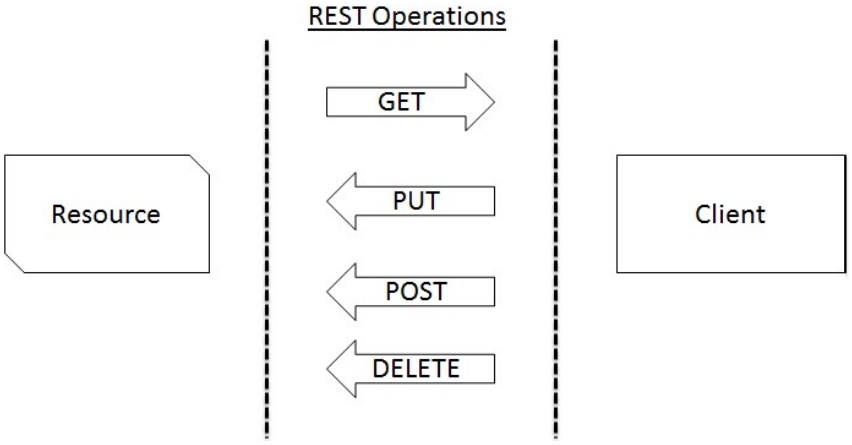

Figure 5 The REST architecture and the supported operations

For secure information transmission, the extension of the Hypertext Transfer Protocol Secure (HTTPS) is widely-used. The pure communication protocol (HTTP) is safeguarded by encrypting the data with the TLS protocol. However, HTTP/HTTPs are not appropriate for lightweight applications with resource, bandwidth, and/or energy restrictions.

Thus, the Internet Engineering Task Force (IETF) Constrained RESTful environments (CoRE) Working Group presented the Constrained Application Protocol (CoAP), now an IETF standard (Shelby et al., 2014). CoAP is a specialized web transfer protocol for use with constrained nodes and constrained networks in the IoT, aiming to maintain compatibility with the existing Internet infrastructure, through simple proxies. The protocol is often referred to as "the HTTP for the Internet of Things". CoAP messages are transported over the User Datagram Protocol (UDP). Moreover, basic publish/subscribe interactions are also supported, as, by extending the HTTP GET method, a client can *observe* a specific resource. For security, CoAP applications support the Datagram Transport Layer Security (DTLS).

3 Simulation Tool Interconnections

We will overview the communications of the Simulation Tool with the relevant platform components. Particularly, the Simulation Tool interconnects with the:

- Training Tool on the state of the simulated cyber system components necessary for the assessment of trainees' performance;
- Emulation Tool to trigger actions/events on a cyber system emulation environment, and enable the generation and realisation of hybrid networks of simulated and emulated components;
- Visualisation Tool on the state of the simulated cyber system for progressive in-browser visualisation during a training session;
- Data Fabrication Platform to request fabrication of synthetic security event logs for the needs of a cyber system simulation, and consequently access those logs during a training session.

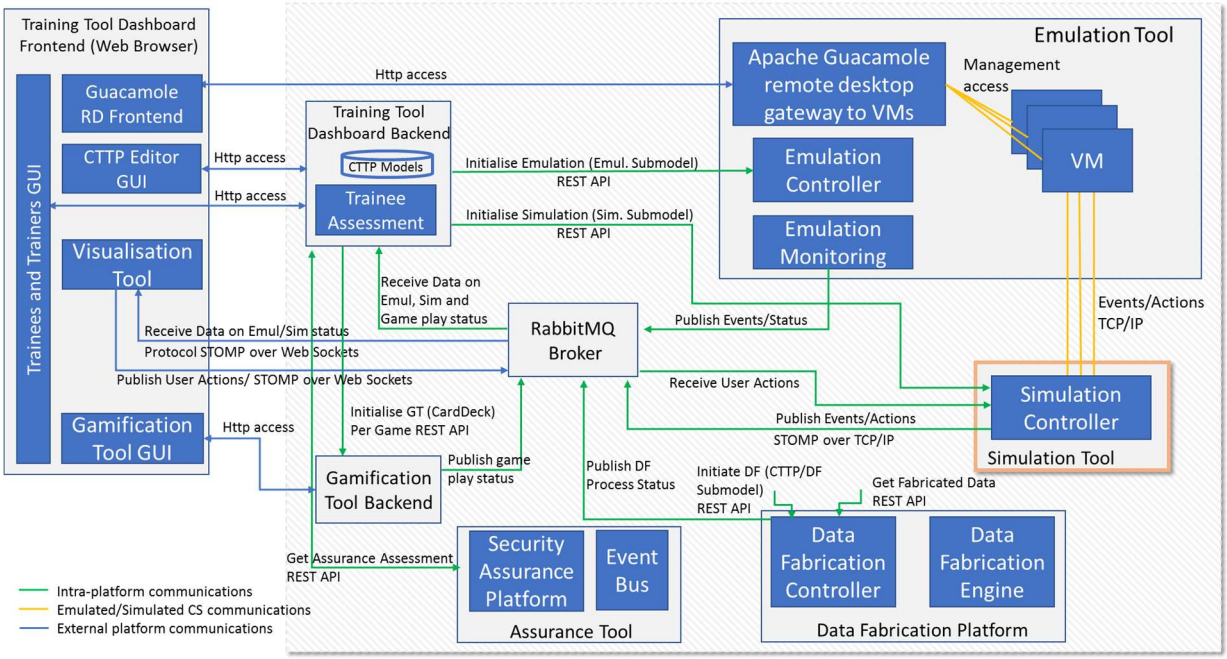

Figure 6: THREAT-ARREST Platform Components Interconnection – Simulation Tool View

Figure 6 shows the THREAT-ARREST platform communications with a particular focus on the Simulation Tool communications. RabbitMQ is selected as the Message Queue Broker for the THREAT-ARREST platform to enable all asynchronous communication needs among the platform components. Particularly, all communications of the Simulation Tool with the Training and Visualisation Tools, on the state of a simulated cyber system, are asynchronous and mediated through the broker.

The Simulation Tool communicates with the message broker using the STOMP[9] protocol over TCP/IP. We note that one of the aspects for choosing the RabbitMQ broker was the multi-protocol support provided by the recent versions of the broker[10]. This decision addressed an important interoperability issue on the protocol level (e.g. (Soultatos et al., 2019)) between the THREAT-ARREST platform components.

[9] STOMP: http://stomp.github.io/
[10] RabbitMQ protocols: https://www.rabbitmq.com/protocols.html

There is a REST API provided by the Simulation Tool at the Simulation Controller to allow cyber system initialisation for a given scenario. Particularly, the Training Tool upon a scenario initialisation, initialises all platform components in a given order through their dedicated REST APIs. Input to the Simulation Tool's initialisation API is the simulation model defined in a given CTTP and a training session information including among other things the Session ID, User ID, User Role, etc. For more details, the CTTP simulation sub-model, which is generated by the Assurance Tool, is documented in the deliverable D3.1 "CTTP Models and Programmes Specification Language", while the extraction of the instantiation scripts from the sub-model and the initialization of simulation components is presented in the deliverable D5.2 "Simulated components and network generator v1".

3.1 Interconnection with the Emulation Tool

For a given scenario and training session, the Simulation Tool is interconnected with the emulated (part of) a cyber system through direct TCP/IP-based communications, as depicted in Figure 6. In this way, the Simulation Tool (i.e., the Simulation Controller) can interact with the emulated cyber system environment and perform/trigger corresponding actions on the environment necessary for the training scenario. For instance, triggering specific network-level connections to an IoT Gateway (hosted in a VM of the emulated environment) coming from the simulation of a compromised IoT device.

To realise such interconnections, as already discussed in D2.4, upon a training session initialisation, the Emulation Tool is initialised first. The Emulation Tool instantiates not only the environment (infrastructure) for cyber system emulation but also the VM of the Simulation Tool in the *same context* (i.e. subnet or network) of the VMs of the emulation environment. We note that the Emulation Controller does not initialise the cyber system simulation for a given scenario but only the VM with the corresponding network, compute and storage resources. Such initialisation is necessary to ensure an environment of the Simulation Tool properly interconnected with the emulated cyber system (i.e. interconnected with the VMs of the emulated cyber system environment) and the external network.

We note that in case of a training scenario with cyber system simulation only, the Emulation Controller is still invoked to initialise the VM of the simulation engine but without any emulation environment initialisation and interaction.

3.2 Interconnection with the Training and Visualisation Tools

The Simulation Tool interconnects with the Training and Visualisation Tools on the state of the cyber system being simulated. These are asynchronous communications through the RabbitMQ message broker. All related events and state information of the simulated cyber system are communicated to both the Training Tool and the Visualisation Tool through the message broker.

These events and state information are used for different purposes – the Training Tool for user performance assessment and the Visualisation Tool for progressive in-browser visualisation of the cyber system environment. Given the different purposes, each tool needs to filter and process only those events relevant to its scope.

To address the different levels of events/state information needed by the platform tools for their operation, it was decided to use RabbitMQ Exchanges of type Topic. Similar to the discussed means in D2.4 and D4.3, *one* Exchange of type Topic is created for the Simulation Tool to serve *all* communications of state information for *all* training scenarios and training sessions. This Topic Exchange is created on set up of the platform and its initial configuration.

Importantly, all messages sent by the Simulation Tool to its predefined Exchange are to bear a *well formed* routing key. It was agreed to use the following structure of the routing key:

```
SimulationTool.<ScenarioID>.<TrainingSessionID>[.<CyberSystemComponentID>]+.<Cyber
SystemComponentAttributeID>
```

The string `SimulationTool` is a constant used to indicate the name of the THREAT-ARREST platform component source of the message. The `<ScenarioID>` refers to the scenario identifier from the CTTP model. The `<TrainingSessionID>` refers to the identifier of the training session as managed by the Training Tool/Dashboard. Upon Simulation Tool initialisation, the actual training session ID is given along the CTTP simulation sub-model.

The `<CyberSystemComponentID>` refers to the identifier of a component of the cyber system that is simulated in a training session, such as an IoT device, sensor, actuator, IoT gateway, network switch, etc. The brackets with an upper index plus "`[..]+`" indicate the expression "`.<CyberSystemComponentID>`" can be repeated one or more times depending on the complexity of the cyber system simulated.

The `<CyberSystemComponentAttributeID>` refers to the identifier of an attribute of a component of the cyber system that is simulated in a training session, such as an IoT device's state (critical/normal/aborted), or sensors' current temperature or humidity measured by it, etc. The routing key of a message can refer to only one attribute of a component.

Given first-year discussions on the topic, the `<CyberSystemComponentAttributeID>` is mandatory as it identifies (qualifies) which aspect of the cyber system component is simulated, and which of the routed messages are referred to it.

It is important to note that the identifier information for the `<CyberSystemComponentID>` and `<CyberSystemComponentAttributeID>` are obtained from the CTTP (sub-)model. In the next version of the deliverable a formal Backus-Naur Form[11] (BNF) specification will be provided for the routing key structure.

As we said, there is one Exchange (*CS_Simulation_State*) pre-defined for all messages sent by the Simulation Tool that will outreach all relevant components of the platform, including the Training and Visualisation Tools.

To do so, it was agreed that upon initialisation of the Training and Visualisation Tools for a given scenario and training session, each tool *dynamically* creates Queues bound to the *CS_Simulation_State*. It is essential that the binding for each Queue follows the structure of the routing key discussed above. Asterisk '*' and pound '#' can be used as placeholders for a single word, or zero or more words, respectively.

For instance, the binding (matching pattern) for a Queue of the Visualization Tool, would be the following one:

```
SimulationTool.<ScenarioID>.<TrainingSessionID>.#
```

The `<ScenarioID>` refers to the identifier of the scenario in the CTTP model. The `<TrainingSessionID>` refers to the identifier of the current training session the Visualisation Tool is initialised for. The binding above means that the Queue will receive all messages sent

[11] https://en.wikipedia.org/wiki/Backus–Naur_form

by the Simulation Tool (to *CS_Simulation_State*) for a given scenario and session ID regardless of what simulated cyber system components they relate to. In this case, the Visualisation Tool will receive all messages and parse those on the application level to visualise their value in the Dashboard.

However, the Visualisation Tool may also subscribe for messages specific to a component of the cyber system, for instance with the following binding key for a Queue:

```
SimulationTool.<ScenarioID>.<TrainingSessionID>.<CyberSystemComponentID>.#
```

In this case, messages for all attributes of a cyber system component will be received. The Visualisation Tool may even subscribe to messages regarding a specific attribute of a simulated cyber system component. In such granularity, it needs to declare a queue per each component and its attribute of interest.

In the same way, upon initialisation, the Training Tool dynamically declares (creates) Queues either per training session or per component or per attribute of a component of the simulated cyber system.

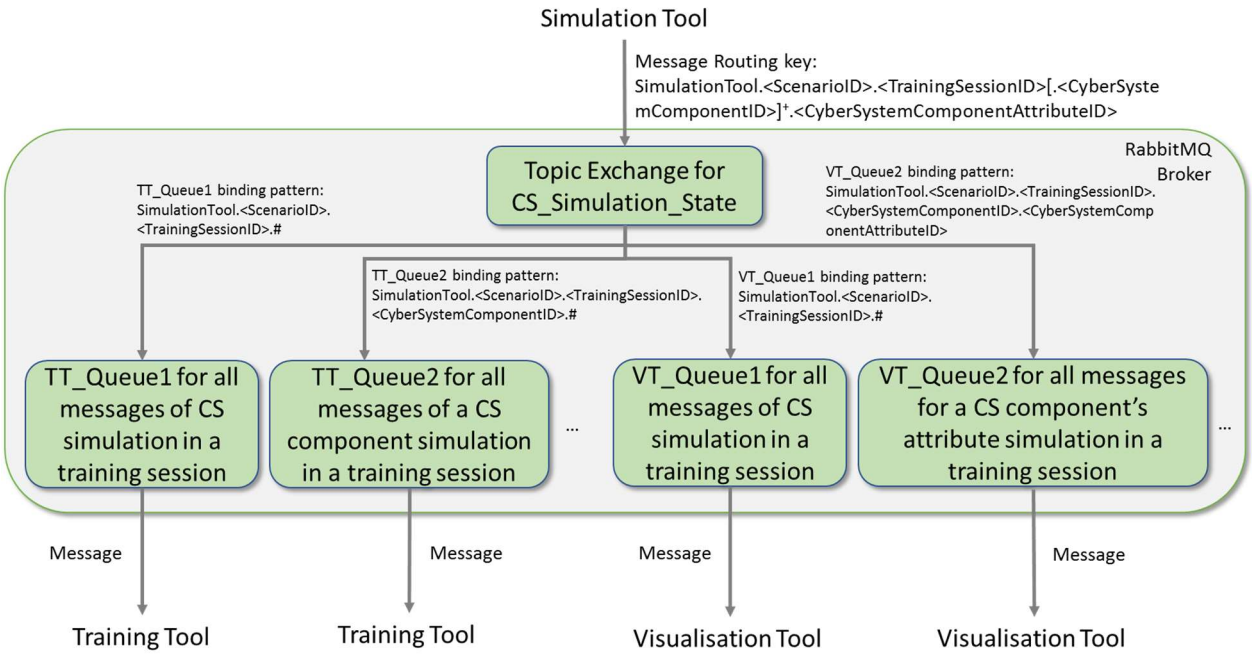

Figure 7: Simulation Tool Interconnection with the Training and Visualisation Tools

Figure 7 illustrates the broker-based communications as discussed above with possible queue binding keys. The abbreviation TT stands for Training Tool, VT for Visualisation Tool, and CS for cyber system. As we emphasized, it is essential to define the scope of each queue and its proper binding keys for the matching.

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

public class EmitCSEmulationState {

  private static final String EXCHANGE_NAME = "CS_Simulation_State";

  public static void main(String[] argv) throws Exception {

    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    try (Connection connection = factory.newConnection();
         Channel channel = connection.createChannel()) {

        channel.exchangeDeclare(EXCHANGE_NAME, "topic");

        String routingKey = "SimulationTool.Energy_BruteForceSSH.TrSess_98374767.Main-
Net.Sensor1.CurrentTemperature";
        String message = "25.5";

        channel.basicPublish(EXCHANGE_NAME, routingKey, null, message.getBytes("UTF-
8"));
    }
  }
}
```

Code Example 1: RabbitMQ Java API for Simulation Tool Topic Exchange Creation and Message Publishing

Code Example 1 shows how the Simulation Tool can use the Java library/API provided by RabbitMQ[12] to create an Exchange of type Topic and publish a message with a routing key following the format presented above.

In the example, the following routing key is used:

```
SimulationTool.Energy_BruteForceSSH.TrSess_98374767.MainNet.Sensor1.CurrentTemperat
ure
```

The message in the example is a temperature measured `25.5` Celsius degrees.

Give the example above, a possible binding key for a queue for all messages of a training session of cyber system simulation would be:

```
SimulationTool.Energy_BruteForceSSH.TrSess_98374767.#
```

An example of a possible binding key of a queue for all messages regarding a specific simulated component of a training session would be:

```
SimulationTool.Energy_BruteForceSSH.TrSess_98374767.MainNet.Sensor1.#
```

An example of a possible binding key of a queue for all messages regarding a specific attribute of a cyber system component simulation would be:

```
SimulationTool.Energy_BruteForceSSH.TrSess_98374767.MainNet.Sensor1.CurrentTemperat
ure
```

We refer to the deliverable D4.3 for more details on how the Training and Visualisation Tools subscribe to the broker and receive messages from the Simulation Tool. This deliverable defines the complementary view of how the Simulation Tool publishes messages to the broker.

3.3 Interconnection with the Visualisation Tool on User Actions

The deliverable D4.3 defines the message broker means to enable the Visualisation Tool to communicate with the Simulation Tool when user actions are performed via the Graphical User

[12] RabbitMQ tutorials: https://www.rabbitmq.com/tutorials/tutorial-five-java.html

Interface (GUI) to the simulated cyber system components. In the rest of this subsection, we recall these communications with a focus on the Simulation Tool.

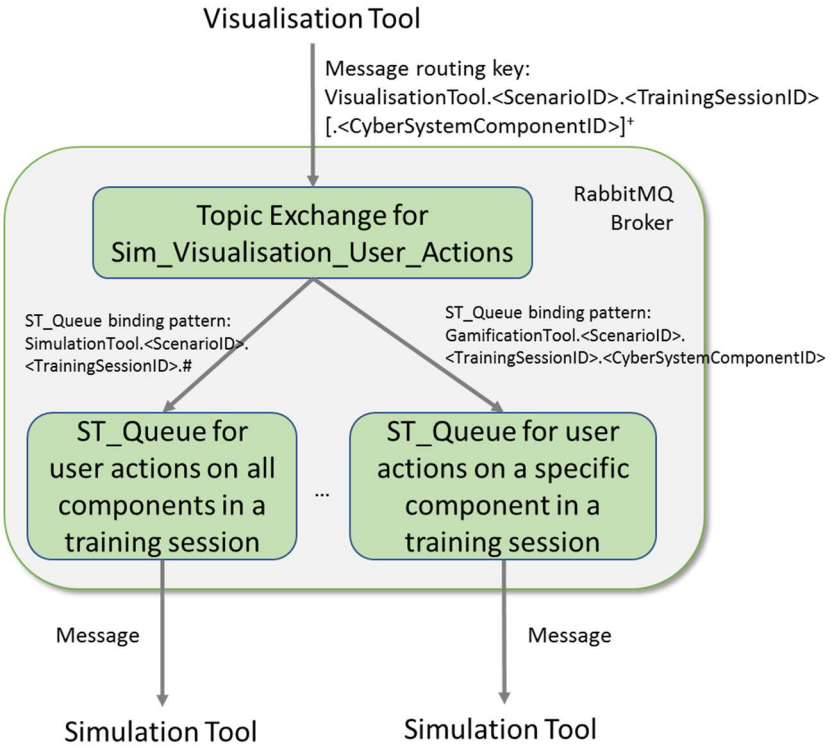

Figure 8: Interconnection of Simulation Tool with Visualisation Tool on User Actions

Figure 8 shows how the Simulation Tool interconnects with the Visualisation Tool on user actions performed. An Exchange of type Topic is predefined for the Visualisation Tool that interfaces all communications of user actions by the Visualisation Tool for all scenarios and training sessions. Upon Simulation Tool initialisation per training session, the Simulation Tool dynamically declares queues (subscribes) to messages sent by the Visualisation Tool. Possible binding key options are shown in the figure depending on the needs of the Simulation Tool.

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.DeliverCallback;

public class ReceiveSimVisUserActions {

  private static final String EXCHANGE_NAME = "Sim_Visualisation_User_Actions ";

  public static void main(String[] argv) throws Exception {

    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    String queueName = channel.queueDeclare().getQueue();

    String bindingKey = "VisualisationTool.Energy_BruteForceSSH.TrSess_98374767.Main-
Net.Sensor1";

    channel.queueBind(queueName, EXCHANGE_NAME, bindingKey);

    System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

    DeliverCallback deliverCallback = (consumerTag, delivery) -> {
        String message = new String(delivery.getBody(), "UTF-8");
        System.out.println(" [x] Received '" +
            delivery.getEnvelope().getRoutingKey() + "':'" + message + "'");
    };
    channel.basicConsume(queueName, true, deliverCallback, consumerTag -> { });
  }
}
```

Code Example 2: RabbitMQ Java API for Simulation Tool Creation of Queue and Receiving Messages of User Actions

Code Example 2 shows the use of RabbitMQ Java library/API to connect to a broker, declare a queue bound to the Exchange for user actions and receive messages routed to the queue. Particularly, the following binding key is used `VisualisationTool.Energy_BruteForceSSH.TrSess_98374767.MainNet.Sensor1`. In this case, the Simulation Tool will receive all messages sent by the Visualisation Tool for user actions regarding *Sensor1* at the main net simulation.

Following this example and upon initialisation, the Simulation Tool will need to dynamically declare a queue per each component of the simulated cyber system of interest in order to receive messages of user actions for these components. Alternatively, it may declare one queue per all user actions of all simulated components in a training session as shown in Figure 8.

3.4 Interconnection with the Data Fabrication Platform

The Simulation Tool interconnects with the Data Fabrication Platform (DFP) for the needs of dynamic security event logs generation during a training session. In such cases, similar to the needs of the Emulation Tool as discussed in D2.4, the Simulation Tool uses the REST API of the DFP to request and access the generated synthetic data. Section 4 overviews the DFP functionality and API.

It is under discussion whether to interface the results of data fabrication for each scenario with all other platform components, including the Simulation Tool, through a Git repository accessible by all platform components.

In any case, the RabbitMQ broker will be used to notify all relevant platform components when the data fabrication process is completed, and where the data is available in the Git repository

(under a specific URI) so that the other components can access it. Alternatively, the DFP can use the broker to notify other tools when the data is ready along with the specific identifier, so that the tools can use the dedicated REST API to obtain the data.

Similar to the above described message broker means, a dedicated Exchange will be created for the DFP so that all other components upon initialisation, including the Simulation Tool, dynamically create a Queue bound to this Exchange to be informed when the requested data is fabricated.

4 Data Fabrication Platform Functionality and API

IBM's Data Fabrication Platform (DFP) (IBM, 2017) is a web-based central platform for generating high-quality structured data for testing, development, and training. The methodology used is termed "model-based rule-guided fabrication". DFP consumes data declaration directives (data model or metadata) along with user-defined rules as input, creates a Constraint Satisfaction Problem (CSP), and solves the problem using a proprietary CSP Solver, which has been used for verifying IBM hardware systems for over a decade. The solver finds a consistent pseudo-random solution satisfying all the data definition and constraint requirements. Finally, the generated data is populated in target databases and/or files. A detailed description of the DFP's functionality for the generation of synthetic event logs is given in deliverable D5.1 (THREAT-ARREST D5.1, 2019).

4.1 IBM Data Fabrication Platform enhancement

To support the THREAT-ARREST requirements, IBM Data Fabrication Platform is being enhanced with the ability to generate sequences of simulated cyber-events in general, and synthetic security event log files in particular. For such a use, the DFP needs to be properly initialized (presumably, based on definitions found in the CTTP model) before a user or a client sub-system can get any synthetic log file.

4.1.1 Cyber Network and Scenario Definition

First, a virtual network topology should be defined according to a given CTTP model. This includes the declaration of network-attached computers, switches, and other relevant hardware. Each hardware node should be decorated with properties and "installed" software applications and services. User-provided rules and constraints should complement the network definition to guide the fabrication engine, how to choose values for hardware and software properties.

Then, an Event Scenario, built of connected Actions and Activities, should be defined. In case a cyber-attack log is required, an attack Scenario should be defined over the virtual Network.

4.1.2 Log Fabrication

After being properly configured with the Network topology and Scenario definition, DFP creates a CPS based on those definitions, solves the Problem with the CSP Solver, producing pseudo-random property and function call parameter values, satisfying all the definitions, rules, and constraints.

Finally, DFP simulates the Scenario, calling in application functions, declared by the Scenario actions, propagating events from one network node to another, and stores the resulting event messages down to some persistent storage, producing event log files.

4.2 THREAT-ARREST Requirements for Synthetic Data

Two types of synthetic data, required for the THREAT-ARREST project, have been identified so far:

(i) static general-purpose synthetic data, such as health records, for the needs of setting/performing a given training scenario;

(ii) static or dynamic (interactive) security (event) logs for cybersecurity training in the context of a training scenario, such as security logs regarding malicious (anomalous) accesses to a server hosting a database of health records.

There is no need of any special DFP API to fabricate static data for case (i). Such data can be modelled in advance via the DFP web-based user interface and fabricated off-line, even before

any training session starts. Fabricated data can be populated in well-known databases and/or predefined file locations to be consumed by other THREAT-ARREST components.

As far as security event logs are concerned, the DFP will be enhanced and the new functionality accessible via a log fabrication REST API is described in the following section.

4.3 Data Fabrication Platform – Log Fabrication API

Before one can get any security log, the DFP needs to be properly initialized and configured as it is described in the subsection 4.1.1. Then, a fabrication and simulation process as described in the subsection 4.1.2 should be controlled and monitored. Finally, after fabrication has been successfully completed, the fabricated log files need to be fetched out and consumed by a client sub-system. We note that the APIs described below will be initialised based on a given CTTP model specification.

4.3.1 Cyber Network Definition API

In the following we overview the API defined for a specification of a cyber network.

- *add/edit/delete sub-network* – functions enabling the creation, modification, and removal of sub-nets (folder-like entities, enabling grouping of network nodes and other sub-nets in a hierarchical manner)
- *add/edit/delete network node* – functions enabling the creation, modification, and removal of a network node
- *add/edit/delete property* – functions enabling the creation, modification, and removal of hardware or software properties (in addition to predefined ones)
- *add/edit/delete constraint* – functions enabling the creation, modification, and removal of a constraint over hardware and/or software properties
- *connect/disconnect nodes* – functions enabling the connection and disconnection of network nodes to create and modify a network graph of connected nodes

4.3.2 Scenario Definition API

In the following we overview the API defined for a specification of a targeted scenario.

- *add/edit/delete activity* – functions enabling the creation, modification, and removal of scenario activities (folder-like entities, enabling grouping of actions and other sub-activities in a hierarchical manner)
- *add/edit/delete action* – functions enabling the creation, modification, and removal of a scenario action
- *add/edit/delete constraint* – functions enabling the creation, modification, and removal of a constraint over action function call parameters
- *link/unlink actions* – functions enabling the connection and disconnection of scenario actions and activities to create and modify a scenario flow (a graph of connected scenario items)

4.3.3 Log Fabrication API

In the following we overview the API defined to generate and access fabricated data.

- *fabricate* – a function starting the fabrication session
- *get fabrication status* – a function providing status of the current fabrication session

- *get fabrication data* – a function providing output data generated in the last session

5 Conclusions and Next Steps

This document is a first deliverable documenting the work done in task T5.4. It presents the technical means used by the Simulation Tool to interconnect with the other platform components primarily using a REST API or message-broker-enabled communications. The results of this first version will guide the Simulation Tool integration activities in the second year of the project. There are two other deliverables that complement this document, the deliverables D2.4 and D4.3. All three deliverables together define the overall view of components' interconnections of the THREAT-ARREST platform.

The adoption of REST interfaces and the RabbitMQ message broker allowed us to address interoperability not only on the API level but on the protocol level as well (see Cameron, 2012).

Next steps in the second year of the project target will address:

- Technical specification of interfaces (APIs) for the Simulation Tool communications with other platform components both through REST API and those through RabbitMQ broker. We note that the technical specification of such API is subject to the design and technical development of individual component's functionalities.
- Interoperability on the message level to ensure the syntax and semantics of messages (e.g., a component's state simulated, or specific sensor's simulated measurements, or any other components' simulated attributes, etc.) sent by the Simulation Tool are understandable by the Training and Visualisation Tools.

We note that the concluding remarks and next steps are common to the three documents D2.4, D4.3, and D5.3 as they altogether (in a complementary way) address the mechanisms and interfaces for interconnecting all THREAT-ARREST platform components.

The steps above will be particularly driven by the activities of WP6 on platform integration and interconnection, which officially starts in Month 13 of the project.

6 References

[1] Banks, A. and Gupta, R. (2014) OASIS Message Queuing Telemetry Transport (MQTT), version 3.1.1, OASIS, pp. 1-81, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf .

[2] Cameron, B. (2012) The Polyglot Rabbit: Examples of Multi-Protocol Queues in RabbitMQ. Available at http://assortedrambles.blogspot.com/2012/11/the-polygot-rabbit.html

[3] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". *Architectural Styles and the Design of Network-based Software Architectures* (Ph.D.). University of California, Irvine. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[4] Hatzivasilis, G., et al. (2019). Secure Semantic Interoperability for IoT Applications with Linked Data. IEEE Global Communications Conference (GLOBECOM 2019), IEEE, Waikoloa, HI, USA, 9-13 December 2019, pp. 1-7.

[5] Hatzivasilis, G., Fysarakis, K., Soultatos, O., Askoxylakis, I., Papaefstathiou, I. and Demetriou G. (2018a) The Industrial Internet of Things as an enabler for a Circular Economy Hy-LP: A novel IIoT Protocol, evaluated on a Wind Park's SDN/NFV-enabled 5G Industrial Network, Computer Communications – Special Issue on Energy-aware Design for Sustainable 5G Networks, Elsevier, vol. 119, pp. 127-137.

[6] Hatzivasilis, G., et al., (2018b). The Interoperability of Things. 23rd IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD 2018), IEEE, Barcelona, Spain, 17-19 September 2018, pp. 1-7.

[7] IBM, "Create high-quality test data while minimizing the risks of using sensitive production data." *IBM InfoSphere Optim Test Data Fabrication*, IBM, 2017, https://www.ibm.com/il-en/marketplace/infosphere-optim-test-data-fabrication

[8] ISO/IEC 20922 (2016). "Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1," June 15, 2016, https://www.iso.org/standard/69466.html .

[9] Johansson, L. (2015) RabbitMQ Exchanges, routing keys and bindings. CloudAMQP Blog. Available at https://www.cloudamqp.com/blog/2015-09-03-part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html .

[10] Lakka, E., et al. (2019). End-to-End Semantic Interoperability Mechanisms for IoT. 24th IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD 2019), IEEE, Limassol, Cyprus, 11-13 September 2019, pp. 1-6.

[11] Lonescu, V. M. (2015). The analysis of the performance of RabbitMQ and ActiveMQ, 14th RoEduNet International Conference – Networking in Education and Research (RoEduNet NER), IEEE, Caiova, Romania, Sept. 24-26, pp. 132-137.

[12] Luzuriaga, J. E., Perez, M., Boronat, P., Cano, J. C., Calafate, C. and Manzoni, P. (2015). A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks, 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), IEEE, pp. 1-6.

[13] Richardson, A. (2014) RabbitMQ Essentials, PACKT Publishing, pp. 1-182. http://www.spooch.dk/Ebooks/Programming/RabbitMQ%20Essentials%20%5BeBook%5D.pdf

[14] Shelby, Z., Hartke, K. and Bormann, C. (2014). The constrained application protocol (CoAP), IETF, RFC 7252. https://tools.ietf.org/html/rfc7252 .

[15] Soultatos, O., et al., 2019. Pattern-Driven Security, Privacy, Dependability and Interoperability Management of IoT Environments. 24th IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD 2019), IEEE, Limassol, Cyprus, 11-13 September 2019, pp. 1-6.

[16] THREAT-ARREST D1.3. (2019). THREAT-ARREST platform's initial reference architecture. THREAT-ARREST Project. Available at https://www.threat-arrest.eu/

[17] THREAT-ARREST D2.4 (2019). Emulation tool interoperability module v1. THREAT-ARREST Project. Available at https://www.threat-arrest.eu/

[18] THREAT-ARREST D4.3 (2019). Training and Visualisation tools IO mechanisms v1. THREAT-ARREST Project. Available at https://www.threat-arrest.eu/

[19] THREAT-ARREST D5.1 (2019). Real event logs statistical profiling module and synthetic event log generator v1. THREAT-ARREST Project. Available at https://www.threat-arrest.eu/