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

D4.3: Training and Visualisation tools IO mechanisms v1 †

Abstract: This deliverable is the result of the first iteration of task T4.6 activities. It defines the technical means and type of interfaces for interconnecting the Training and Visualisation Tools with the relevant platform components such as with the Emulation Tool, the Simulation Tool and the Gamification Tool. The document is the first version of the means of communications. Its goal is to guide the Training and Visualisation Tools' integration activities in the second year of the project and 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	*Torsten Hildebrandt (SIMPLAN),* *George Bravos (ITML),* *Ludger Goeke (SEA).*
Quality Assurance	*Kristian Beckers (SEA),* *Georgios Leftheriotis (TUV),* *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. *Kristian Beckers (SEA),*
2. *Georgios Leftheriotis (TUV),*
3. *George Hatzivasilis (FORTH).*

Revisions

Version	Date	By	Overview
0.7	27/08/2019	Editor	Addressed the comments by FORTH and SEA from the internal quality review process
0.6	23/08/2019	Editor	Addressed the comments by TUV and SEA from the internal quality review process
0.5	07/08/2019	Editor, SIMPLAN	Editor minor corrections before internal review. SIMPLAN revision and corrections to Section 4
0.4	07/08/2019	ITML	ITML's contribution to Section 3.1
0.3	06/08/2019	Editor	ATOS' contribution to Sections 1, 3, 4, 0
0.2	28/06/2019	SEA	SEA's contribution to Section 5
0.1	20/05/2019	Editor	First Draft with ToC

Executive Summary

This document is the result of the first iteration of task T4.6 activities and reports the work performed under the task by month 12 of the project. It steps on and extends the results of the deliverable "D1.3 – THREAT-ARREST platform's initial reference architecture" to define the technical means and interfaces for interconnecting the Training and Visualisation Tools with the relevant platform components such as with the Emulation Tool, the Simulation Tool and the Gamification Tool.

The goal of this first version is to guide the Training and Visualisation Tools' integration activities in the second year of the project and proper interconnection with the other platform components. Particularly, this document is related with 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 deliverables "D2.4 – Emulation tool interoperability module v1" and "D5.3 – The Simulation component IO module 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.

Table of Contents

1 INTRODUCTION 9

2 MESSAGE BROKER AND REST COMMUNICATIONS.... 10

2.1 MESSAGE BROKER-ENABLED COMMUNICATIONS 10

2.1.1 Standard RabbitMQ Message Flow 13

2.1.2 RabbitMQ Topic Exchange 14

2.2 REST COMMUNICATIONS.... 14

3 TRAINING TOOL INTERCONNECTIONS 16

3.1 SCENARIO INITIALISATION SEQUENCE OF COMMUNICATIONS 17

3.2 INTERCONNECTION WITH THE EMULATION AND SIMULATION TOOLS 18

3.3 INTERCONNECTION WITH THE GAMIFICATION TOOL 20

3.4 INTERCONNECTION WITH THE ASSURANCE TOOL.... 23

4 VISUALISATION TOOL INTERCONNECTIONS.... 24

4.1 INTERCONNECTION WITH THE EMULATION AND SIMULATION TOOLS 25

4.2 INTERCONNECTION WITH THE SIMULATION TOOL ON USER ACTIONS 26

5 GAMIFICATION TOOL INTERCONNECTION AND REST API.... 29

5.1 PROTECT REST API.... 29

5.1.1 Functionality for creating a PROTECT instance.... 29

5.1.2 Functionality for querying game result(s).... 30

6 CONCLUSIONS.... 35

7 REFERENCES 36

List of Abbreviations

AMQP Advanced Message Queuing Protocol

API Application Programming Interface

CoAP Constrained Application Protocol

CoRE Constrained RESTful environments

CTTP Cyber Threat and Training Preparation

DTLS Datagram Transport Layer Security

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

Figure 1 Basic steps to create an application with RabbitMQ 11
Figure 2 Sequence diagram for discovery operation.. 12
Figure 3 Sequence diagram for event subscription operation 12
Figure 4. Standard RabbitMQ message flow 13
Figure 5 The REST architecture and the supported operations 15
Figure 6: THREAT-ARREST Platform Components Interconnection – Training Tool View 16
Figure 7: THREAT-ARREST Sequence Diagram Scenario Initialization and Trainees' Assessment 18
Figure 8: Training Tool Interconnection with Emulation and Simulation Tools..................... 19
Figure 9: Training Tool Interconnection with Gamification Tool on the Game Play Status ... 21
Figure 10: THREAT-ARREST Platform Components Interconnection – Visualisation Tool View 24
Figure 11: Interconnection of Visualisation Tool with Emulation and Simulation Tools 25
Figure 12: Interconnection of Visualisation Tool with Simulation Tool on User Actions 26
Figure 13: Example for an interconnection between the Training Tool and the Gamification Tool 29

List of Code Examples

Code Example 1: RabbitMQ Java API for Training Tool Creation of Queue and Receiving Messages of Cyber System Simulation State 20
Code Example 2: RabbitMQ Java API for Gamification Tool Topic Exchange Creation and Message Publishing 22
Code Example 3: RabbitMQ Java API for Training Tool Creation of Queue and Receiving Messages of Serious Games State 23
Code Example 4: RabbitMQ Java API for Visualisation Tool Topic Exchange Creation and Message Publishing on User Actions 27
Code Example 5: RabbitMQ Java API for Simulation Tool Creation of Queue and Receiving Messages of User Actions 28

1 Introduction

This deliverable defines the technical means and interfaces for interconnecting the Training and Visualisation Tools with the relevant platform components such as the Emulation Tool, the Simulation Tool, the Gamification Tool and the Assurance Tool. The document is the first version of the means of communications and it will be used as a guideline in the second year of the project to enable Training and Visualisation Tools interconnect with the other platform components.

The Training Tool is a central component of the THREAT-ARREST platform, in charge of offering functionality to both trainees, on selecting and performing training sessions, and trainers, on setting up training scenarios and configurations. The Training Tool's Dashboard offers an integral Graphical User Interface (GUI) to the various platform components' functionalities and specific interfaces. To achieve proper trainees' performance assessment, the Training Tool interconnects with the Simulation and Emulation Tools on the state of the cyber system, simulated or emulated, and with the Gamification Tool on the state and results of serious games played by trainees.

The Training Tool also interconnects with the Assurance Tool to initialise the monitoring of the trainee's actions against the actual cyber physical system of an organisation and get the necessary data for Cyber Threat and Training Preparation (CTTP) programmes' evaluation. The Assurance Tool may retrieve and update CTTP models stored in the platform for this purpose. It has the role of monitoring and assessing the security posture of the actual cyber system of an organisation where the THREAT-ARREST training platform is used at. The Assurance Tool has its own stand-alone GUI that will be integrated in the Dashboard in the next phase of the project.

The Visualisation Tool is designed as a JavaScript library offering a flexible mechanism for visualisation of the state of a cyber system in a training session. It allows (i) the definition of visual components for each user role and (ii) linking these components to real-time information from the simulated or emulated components of a cyber system. To this, the Visualisation Tool interconnects with the Simulation and Emulation Tools of the THREAT-ARREST platform.

The document also covers the interconnection of the Training Tool with the Gamification Tool. Section 3.3 presents message-broker-based communications and Section 5 REST-based communications. The goal is to provide a more complete and integral view of the Training Tool interconnections with the rest of the platform components.

As noted earlier, this deliverable has two other counterpart documents – deliverable D2.4 (THREAT-ARREST D2.4, 2019) and deliverable D5.3 (THREAT-ARREST D5.3, 2019) – and altogether the three documents provide an overall view of THREAT-ARREST platform interconnections for year 1 of the project.

For the 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 Representational State Transfer (REST) interfaces. Section 3 presents in detail the Training Tool interconnections with the other platform components, particularly with the Emulation, Simulation, Gamification and Assurance Tools. Section 4 details the interconnections of the Visualisation Tool with the Emulation and Simulation components. Section 5 describes the Gamification Tool REST API, particularly the API for the game PROTECT. Section 6 concludes the document and outlines next steps of activities.

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 Representational State Transfer (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)[1], 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[2] (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

[1] OASIS: "MQTT 3.1.1 specification," December 10, 2015, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
[2] 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[3] 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[4] 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[5]. Figure 1 illustrates the basic steps for creating an application with RabbitMQ (Lonescu, 2015).

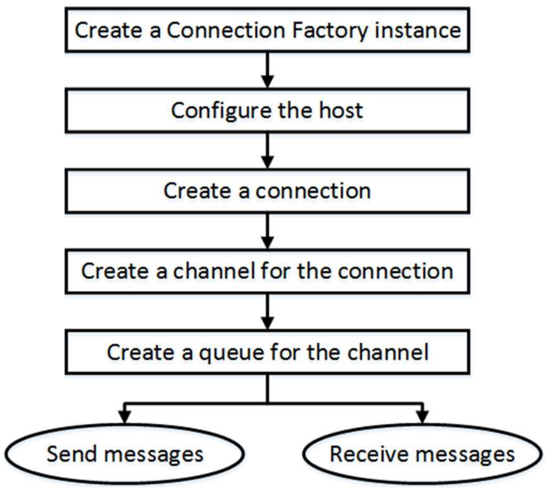

Create a Connection Factory instance
Configure the host
Create a connection
Create a channel for the connection
Create a queue for the channel
Send messages
Receive messages

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.

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

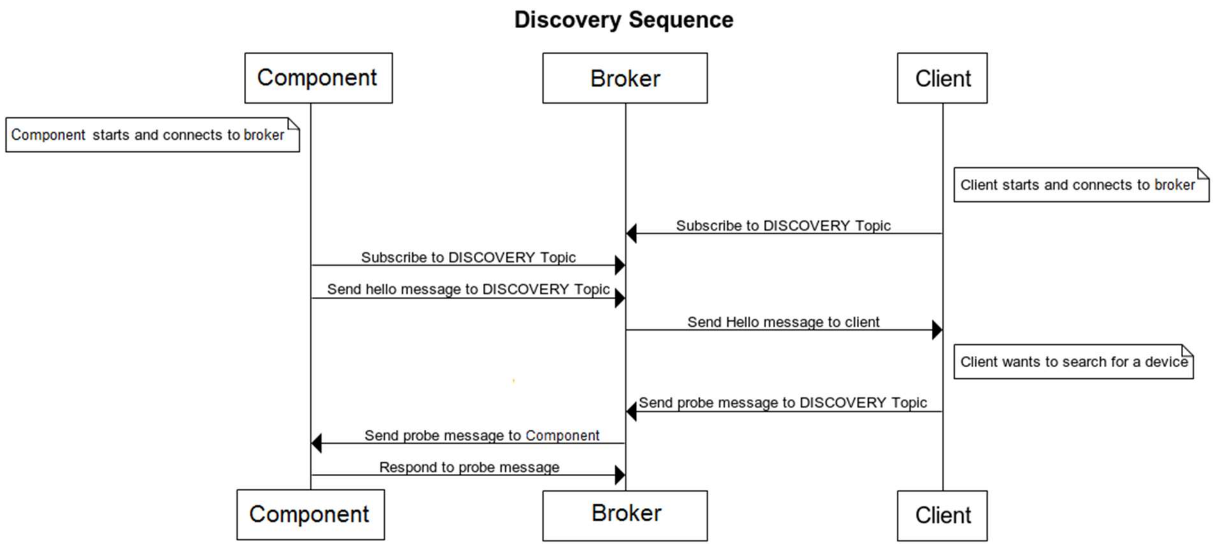

Discovery Sequence
Component
Broker
Client
Component starts and connects to broker
Client starts and connects to broker
Subscribe to DISCOVERY Topic
Subscribe to DISCOVERY Topic
Send hello message to DISCOVERY Topic
Send Hello message to client
Client wants to search for a device
Send probe message to DISCOVERY Topic
Send probe message to Component
Respond to probe message
Component
Broker
Client

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.

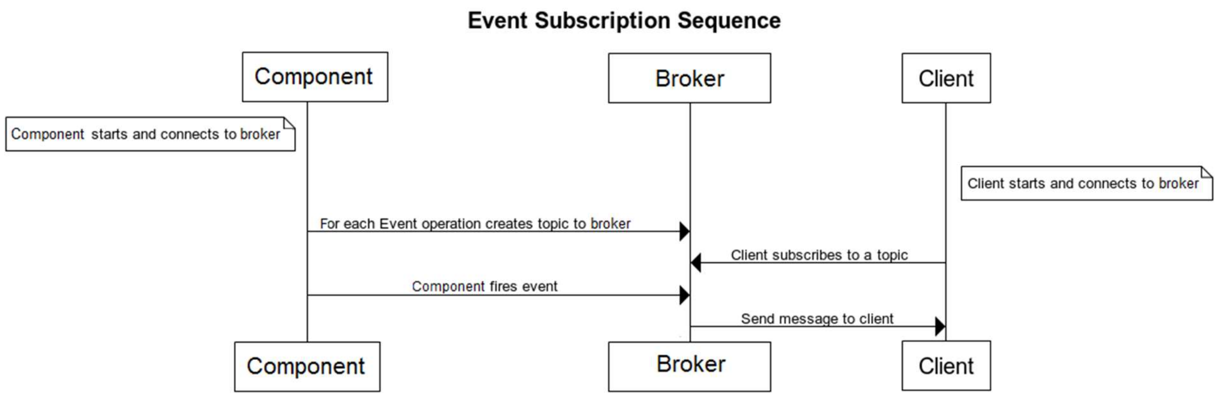

Event Subscription Sequence
Component
Broker
Client
Component starts and connects to broker
Client starts and connects to broker
For each Event operation creates topic to broker
Client subscribes to a topic
Component fires event
Send message to client
Component
Broker
Client

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[6] 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[7] 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).

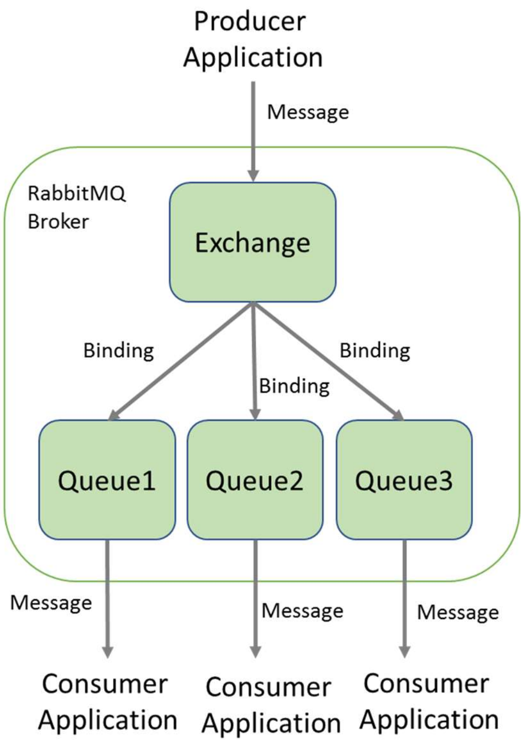


Figure 4. Standard RabbitMQ message flow

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

[6] https://www.rabbitmq.com/getstarted.html
[7] 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.

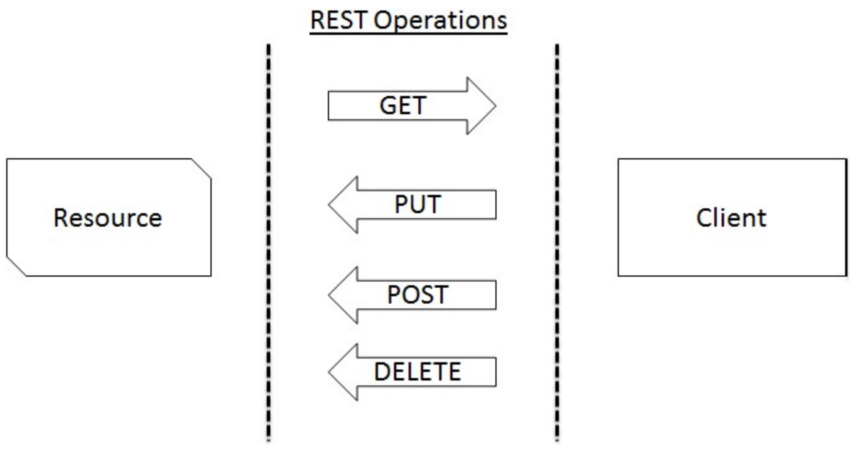


Figure 5 The REST architecture and the supported operations

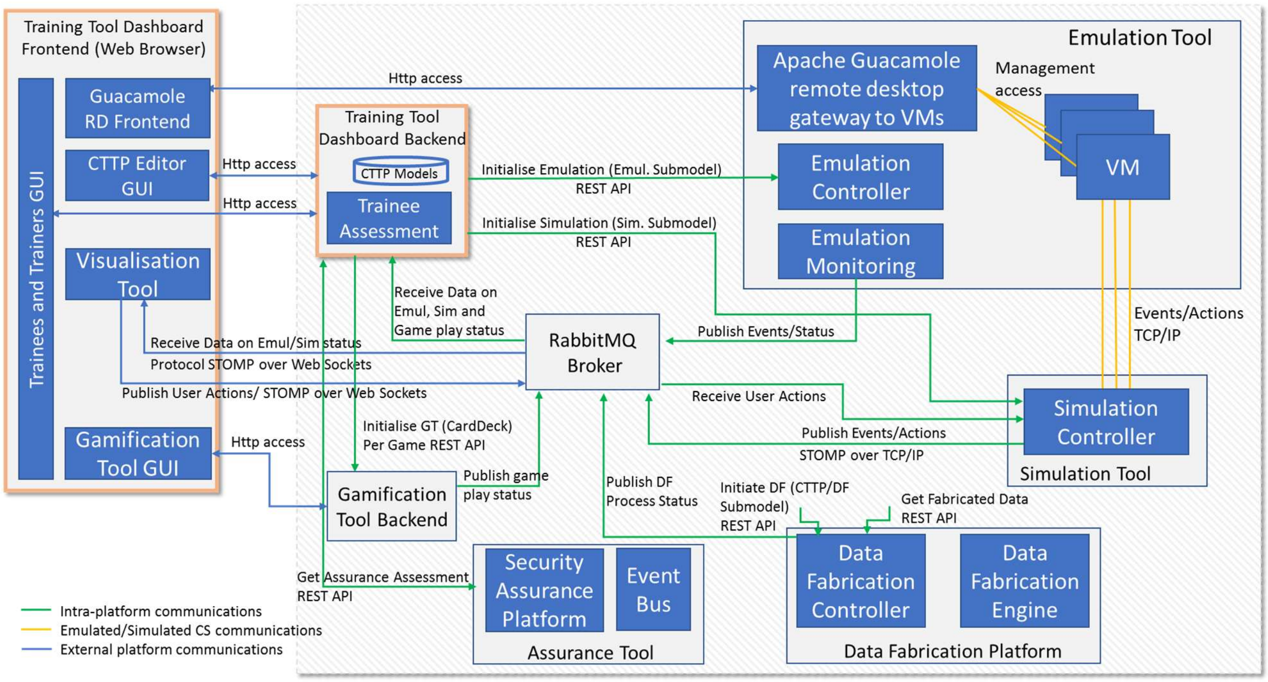
For secure information transmission, the extension of the Hypertext Transfer Protocol Secure (HTTPS) is widely-used. The pure communication protocol (HTTP) is safeguarded by encrypting the data with the TLS protocol. However, HTTP/HTTPs are not appropriate for lightweight applications with resource,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

The Training Tool is the central component of the platform offering cybersecurity training functionality to both trainees and trainers through a tailored to the THREAT-ARREST project needs Dashboard and GUI. The Dashboard follows a co-design approach which involves both technical partners in the project and use case partners to better address user experience and interactions with the platform for cybersecurity training.

Figure 6 shows the THREAT-ARREST platform components interactions with a focus on the Training Tool's front-end (Web browser) and backend communications. The front-end offers an integral view of the different platform components and their GUI. The GUI of the platform components communicate with different back-ends of the respective components. All these front-end communications are HTTP-based communications except for the Visualisation Tool's communications described in Section 4.



Training Tool Dashboard Frontend (Web Browser)
Trainees and Trainers GUI
Guacamole RD Frontend
CTTP Editor GUI
Visualisation Tool
Gamification Tool GUI
Http access
Training Tool Dashboard Backend
CTTP Models
Trainee Assessment
Emulation Tool
Apache Guacamole remote desktop gateway to VMs
Management access
VM
Initialise Emulation (Emul. Submodel) REST API
Emulation Controller
Initialise Simulation (Sim. Submodel) REST API
Emulation Monitoring
Receive Data on Emul, Sim and Game play status
Receive Data on Emul/Sim status
Protocol STOMP over Web Sockets
RabbitMQ Broker
Publish Events/Status
Events/Actions TCP/IP
Publish User Actions/ STOMP over Web Sockets
Receive User Actions
Initialise GT (CardDeck) Per Game REST API
Publish Events/Actions
STOMP over TCP/IP
Simulation Controller
Simulation Tool
Publish game play status
Gamification Tool Backend
Publish DF Process Status
Initiate DF (CTTP/DF Submodel) REST API
Get Fabricated Data REST API
Get Assurance Assessment REST API
Security Assurance Platform
Event Bus
Assurance Tool
Data Fabrication Controller
Data Fabrication Engine
Data Fabrication Platform
Intra-platform communications
Emulated/Simulated CS communications
External platform communications

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

The main Training Tool back-end interconnections are:

- Interconnection with the Emulation Tool on the state of the cyber system being emulated. Such real time state or event information facilitates the user performance assessment during hands-on training sessions.
- Interconnection with the Simulation Tool on the state of the cyber system being simulated. Similar to emulation, the aim is to facilitate user performance assessment with respect to the events/state of the simulated part of the cyber system.
- Interconnection with the Gamification Tool on the state and results of games played by the trainees. Results of game plays will form part of the overall user assessment for a given training scenario.
- Interconnection with the Assurance Tool to initiate the monitoring of trainee's actions against the real cyber system and obtain the information needed for the evaluation of CTTP programmes.

Given platform architecture results of year 1, the Training Tool's front-end becomes a user entry point to interact with other platform components, such as the Emulation Tool's Guacamole[8] Remote Desktop Gateway (for hands-on training), the Gamification Tool's serious games for cybersecurity training (namely for social engineering training), and the Visualisation Tool's JavaScript engine for progressive in-browser visualisation of the state of cyber system components (simulated/emulated).

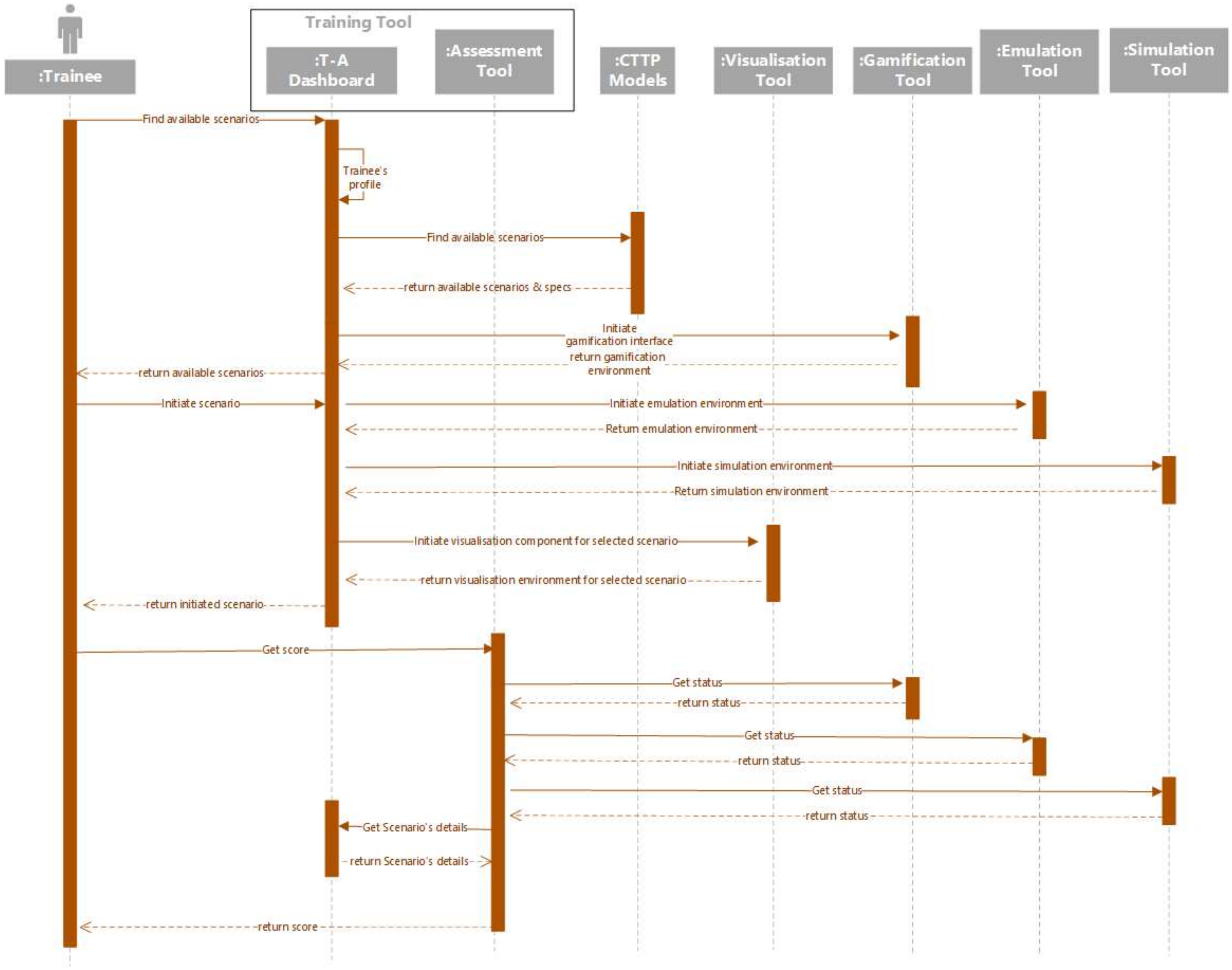
3.1 Scenario Initialisation Sequence of Communications

Upon training scenario initialization, the sequence of communications for the THREAT-ARREST platform has been analysed in detail, with respect to the sequence of the actions / messages to be exchanged between the various components of the THREAT-ARREST platform. The outcome of this analysis has been depicted in a detailed sequence diagram in Figure 7. The THREAT-ARREST dashboard, the main component of the THREAT-ARREST Training Tool, is responsible for the initialisation of all relevant components for each training session and, consequently, responsible for the aggregation of all information regarding the profiles and the assessment of the trainees.

In more detail, and according to the diagram, the first step in the communication process is related to the acquiring of all necessary information from the CTTP modeler. Following that, the Training Tool initializes the emulation and the simulation environments first; if needed, the gamification environment is also initialized. After that, the Visualisation Tool is also initialized, and lastly the Assessment Tool is initialized in order to provide real-time information to the trainees and the trainers.

With respect to the above sequence of communications, the initialization of each tool is performed by a dedicated REST API provided by each tool. In order to carry out the initialisation, a CTTP model (or relevant part of it) is used as an input, along with information about each training session; this information includes, among others, the session ID, the user ID and the role ID.

[8] https://guacamole.apache.org



Training Tool
:Trainee
:T-A Dashboard
:Assessment Tool
:CTTP Models
:Visualisation Tool
:Gamification Tool
:Emulation Tool
:Simulation Tool
Find available scenarios
Trainee's profile
Find available scenarios
return available scenarios & specs
Initiate gamification interface
return gamification environment
return available scenarios
Initiate scenario
Initiate emulation environment
Return emulation environment
Initiate simulation environment
Return simulation environment
Initiate visualisation component for selected scenario
return visualisation environment for selected scenario
return initiated scenario
Get score
Get status
return status
Get status
return status
Get status
return status
Get Scenario's details
return Scenario's details
return score

Figure 7: THREAT-ARREST Sequence Diagram Scenario Initialization and Trainees' Assessment

3.2 Interconnection with the Emulation and Simulation Tools

The Training Tool's interconnections with the Emulation and Simulation Tools are particularly characterised by the need for *asynchronous* and *real time feedback* on the state and events of the simulated and/or emulated cyber system.

The RabbitMQ message broker was chosen as an architectural solution to the needs of the platform. In the following, we will present how the Training Tool receives information states from the cyber system emulation/simulation through the RabbitMQ-supported communications channels.

We recall that both the Emulation and Simulation Tools each have a predefined *Exchange* agent of type *Topic* at the RabbitMQ broker of the platform. These Exchanges are predefined and created at deployment and set-up of the platform, and are in charge of routing all communications of the corresponding tool (for all scenarios and training sessions) to other platform components. We refer to deliverable D2.4 for details on the Emulation Tool message routing through the broker, and to deliverable D5.3 for those of the Simulation Tool.

In contrast to the predefined (static) Exchanges for publishing messages to the platform, the Training Tool *dynamically* upon training scenario initialisation declares (creates) Queues at the broker to listen to specific training session messages. Thus, Queues are dynamically declared for each training session and bound to the corresponding predefined Exchanges of the Emulation and Simulation Tools. Importantly, the binding key (pattern) of each Queue must *correctly* refer to the structure of the routing key of the messages published to the selected Exchange (refer to subsections 2.1).

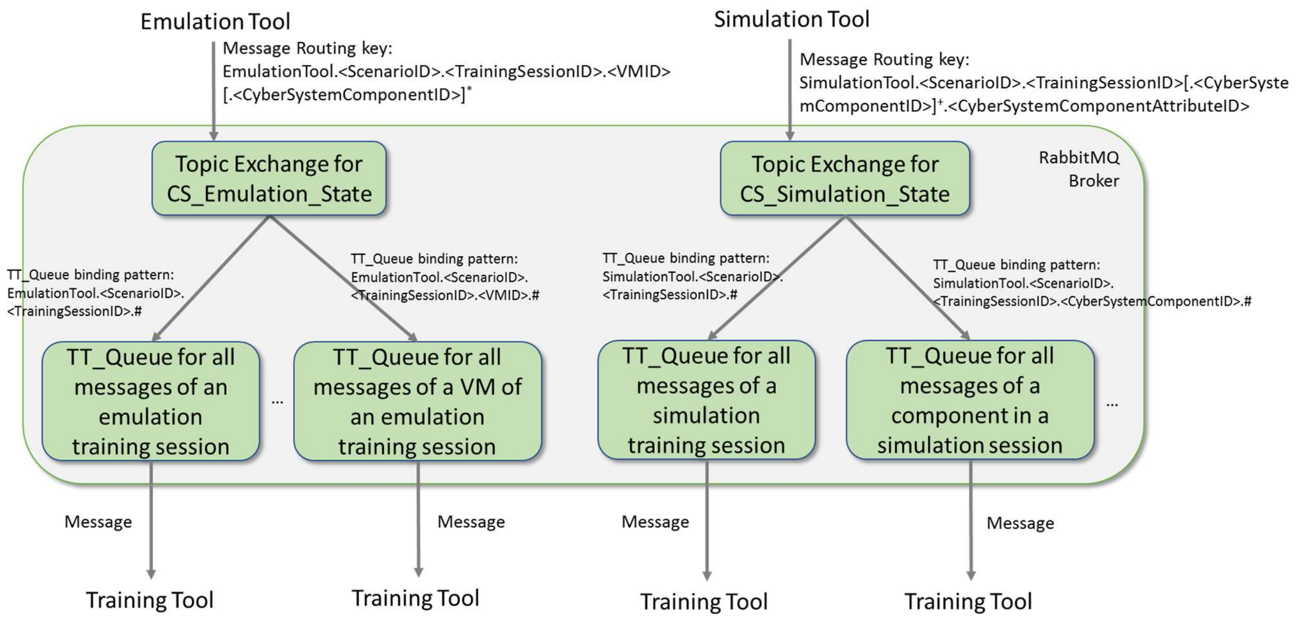

Emulation Tool
Message Routing key:
EmulationTool.<ScenarioID>.<TrainingSessionID>.<VMID>
[.<CyberSystemComponentID>]*
Simulation Tool
Message Routing key:
SimulationTool.<ScenarioID>.<TrainingSessionID>[.<CyberSystemComponentID>]+.<CyberSystemComponentAttributeID>
RabbitMQ Broker
Topic Exchange for CS_Emulation_State
Topic Exchange for CS_Simulation_State
TT_Queue binding pattern: EmulationTool.<ScenarioID>.<TrainingSessionID>.#
TT_Queue binding pattern: EmulationTool.<ScenarioID>.<TrainingSessionID>.<VMID>.#
TT_Queue binding pattern: SimulationTool.<ScenarioID>.<TrainingSessionID>.#
TT_Queue binding pattern: SimulationTool.<ScenarioID>.<TrainingSessionID>.<CyberSystemComponentID>.#
TT_Queue for all messages of an emulation training session
...
TT_Queue for all messages of a VM of an emulation training session
TT_Queue for all messages of a simulation training session
TT_Queue for all messages of a component in a simulation session
...
Message
Message
Message
Message
Training Tool
Training Tool
Training Tool
Training Tool

Figure 8: Training Tool Interconnection with Emulation and Simulation Tools

Figure 8 shows how the Training Tool interconnects with the Emulation and Simulation Tools through the message broker and how different queue binding keys can be used depending on the needs or granularity of the referred cyber system components to receive messages on state or events of the cyber system.

The essential aspect of such communications is the *matching* of the routing keys used by the Emulation and Simulation Tools when publishing messages to the broker, and the pattern of the binding key of each queue declared by the Training Tool.

We recall that the hash/pound symbol '#' is a place holder for zero or more words of the matching process, while the asterisk '*' symbol is a placeholder for one word. Refer to the RabbitMQ documentation[9] for more details on the use of Topics.

[9] https://www.rabbitmq.com/getstarted.html

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

public class ReceiveCSSimulationState {

  private static final String EXCHANGE_NAME = "CS_Simulation_State";

  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 = "SimulationTool.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 1: RabbitMQ Java API for Training Tool Creation of Queue and Receiving Messages of Cyber System Simulation State

Code Example 1 shows how to use Application Programming Interface (API) of the RabbitMQ Java library[10] to declare a Queue bound to the Exchange for state of cyber system simulation and receive messages from the queue.

After a queue is declared (`channel.queueDeclare().getQueue()`), the queue is bound to the Exchange predefined for cyber system simulation state (`channel.queueBind(queueName, EXCHANGE_NAME, bindingKey)`). The binding key used in the example is the following: `String bindingKey = "SimulationTool.Energy_BruteForceSSH.TrSess_98374767.mainNet.sensor1.#"`

This binding key will make RabbitMQ route all messages on the state of sensor1 at the main net simulated for the Scenario `Energy_BruteForceSSH` and for a training session `TrSess_98374767`.

Similarly, one can use the example of Code Example 1 to create a queue for receiving messages on the state of cyber system emulation using the predefined Exchange for the Emulation Tool and a proper binding key.

3.3 Interconnection with the Gamification Tool

The Training Tool interconnects with the Gamification Tool for initialisation of games upon training scenario initialisation and for getting results of games played by trainees for their overall assessment. A REST API is defined by the Gamification Tool for these needs. Section 5 presents functionality provided by the REST API of the game PROTECT as available in the first year of the project. Similar activities of other games' API will follow in the second year of the project.

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

Information on the status that a game has been finished is identified relevant for the Training Tool in order to make trainers timely informed on the trainee's results. The RabbitMQ broker is used to enable such asynchronous communication on the final state of a game that has been played by the trainee.

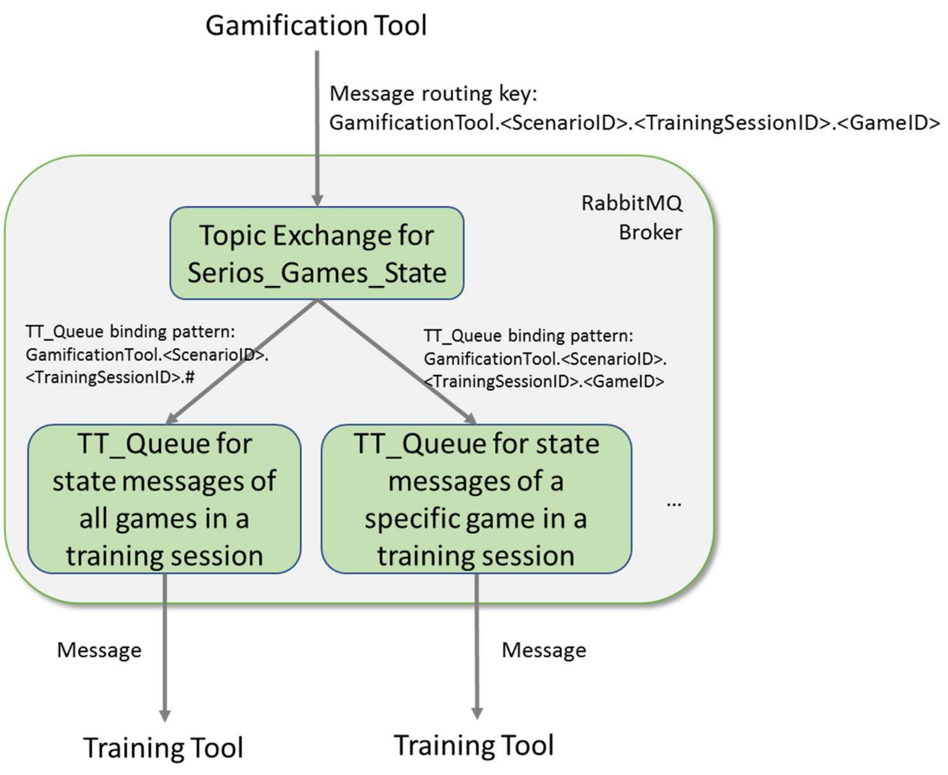

Gamification Tool
Message routing key: GamificationTool.<ScenarioID>.<TrainingSessionID>.<GameID>
RabbitMQ Broker
Topic Exchange for Serios_Games_State
TT_Queue binding pattern: GamificationTool.<ScenarioID>.<TrainingSessionID>.#
TT_Queue binding pattern: GamificationTool.<ScenarioID>.<TrainingSessionID>.<GameID>
TT_Queue for state messages of all games in a training session
TT_Queue for state messages of a specific game in a training session
...
Message
Message
Training Tool
Training Tool

Figure 9: Training Tool Interconnection with Gamification Tool on the Game Play Status

Figure 9 shows how the Training Tool interconnects with Gamification Tool through the message broker. An Exchange of type Topic is predefined for the Gamification Tool that will interface all communications of the Gamification Tool on the state of serious games played by trainees.

The message routing key for all game state messages sent by the Gamification Tool has the following structure:

```
GamificationTool.<ScenarioID>.<TrainingSessionID>.<GameID>
```

The constant `GamificationTool` is used to indicate the name of the THREAT-ARREST platform component that is the 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. The `<GameID>` refers to the *instance* of the game the status message refers to.

Upon a training session initialisation for a given scenario, the Training Tool dynamically declares a Queue bound to that Exchange with a proper binding key. For instance, the binding key (pattern) can be exactly the one following the routing key:

```
GamificationTool.<ScenarioID>.<TrainingSessionID>.<GameID>
```

In such case, all messages for a game ID in a given training session are received on the queue.

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

public class EmitSeriousGameState {

  private static final String EXCHANGE_NAME = "Serious_Game_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 = "GamificationTool.EmailPhishing_RootKit.TrSess_98374767.PRO-
TECT_6352";
        String message = "GAME_PLAY_FINISHED";

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

Code Example 2: RabbitMQ Java API for Gamification Tool Topic Exchange Creation and Message Publishing

Code Example 2 shows how the Gamification Tool can use the RabbitMQ Java API to declare an Exchange and publish messages to this exchange using the routing key structure defined above.

The following routing key example is used: `String routingKey = "GamificationTool.EmailPhishing_RootKit.TrSess_98374767.PROTECT_6352"`. Messages with that routing key indicate the state info of a game instance `PROTECT_6352` for a training session `TrSess_98374767`. In the example, the message of the game state is `GAME_PLAY_FINISHED`.

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

public class ReceiveSeriousGameState {

  private static final String EXCHANGE_NAME = "Serious_Game_State";

  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 = "GamificationTool.EmailPhishing_RootKit.TrSess_98374767.PRO-
TECT_6352";

    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 3: RabbitMQ Java API for Training Tool Creation of Queue and Receiving Messages of Serious Games State

Code Example 3 shows how the Training Tool can use the RabbitMQ Java API for dynamic declaration of a queue bound to the Exchange predefined for the serious games with a particular binding key. The binding key used in the example is `String bindingKey = "GamificationTool.EmailPhishing_RootKit.TrSess_98374767.PROTECT_6352"` that matches all messages sent with the API example shown in Code Example 2.

3.4 Interconnection with the Assurance Tool

The Training Tool interconnects with the Assurance Tool through a REST API to initialise the monitoring of a trainee's actions against a real cyber system and obtain the information needed for the evaluation of CTTP programmes. A preliminary version of the Assurance Tool's REST API is presented in the deliverable "D1.3 – THREAT-ARREST platform's initial reference architecture". The current version of the Assurance Tool addresses mostly the monitoring aspects of a target cyber system and produces the CTTP models stored in the Training Tool repository. Given the project timeline, a more advanced version of API and interconnections with the Assurance Tool will be defined in the second year of the project.

4 Visualisation Tool Interconnections

The Training Tool's Dashboard is the central component of the platform offering cybersecurity training functionality to both trainees and trainers through a tailored to the project needs GUI. The Dashboard is a user entry point to interact with other platform components such as with the Emulation Tool's Guacamole[11] Remote Desktop Gateway (for hands-on training), the Gamification Tool's serious games for cybersecurity training (namely for social engineering training), and the Visualisation Tool for the state of cyber system components, either simulated or emulated. We note that both the Guacamole Desktop and the Gamification Tool's serious games have their own (managed) GUI which will be part of the overall Dashboard design and layout. No specific visualisation support is required for those components. We refer to deliverable D4.2 (THREAT-ARREST D4.2, 2019) for details on the GUI and concept of the serious games.

The Visualisation Tool is a JavaScript engine for progressive in-browser visualisation of the state of cyber system components. As such, the Visualisation Tool interconnects with the Simulation Tool and Emulation Tool to receive real time information on the state of the cyber system. We note that in case other state information is identified necessary for visualisation during second year of the project from other components of the platform, these will follow the interconnection means established in this document.

The Visualisation Tool's interconnections with the Simulation and Emulation Tools are particularly characterised by the need for *asynchronous* and *real-time feedback* on the state and events of the simulated and/or emulated cyber system.

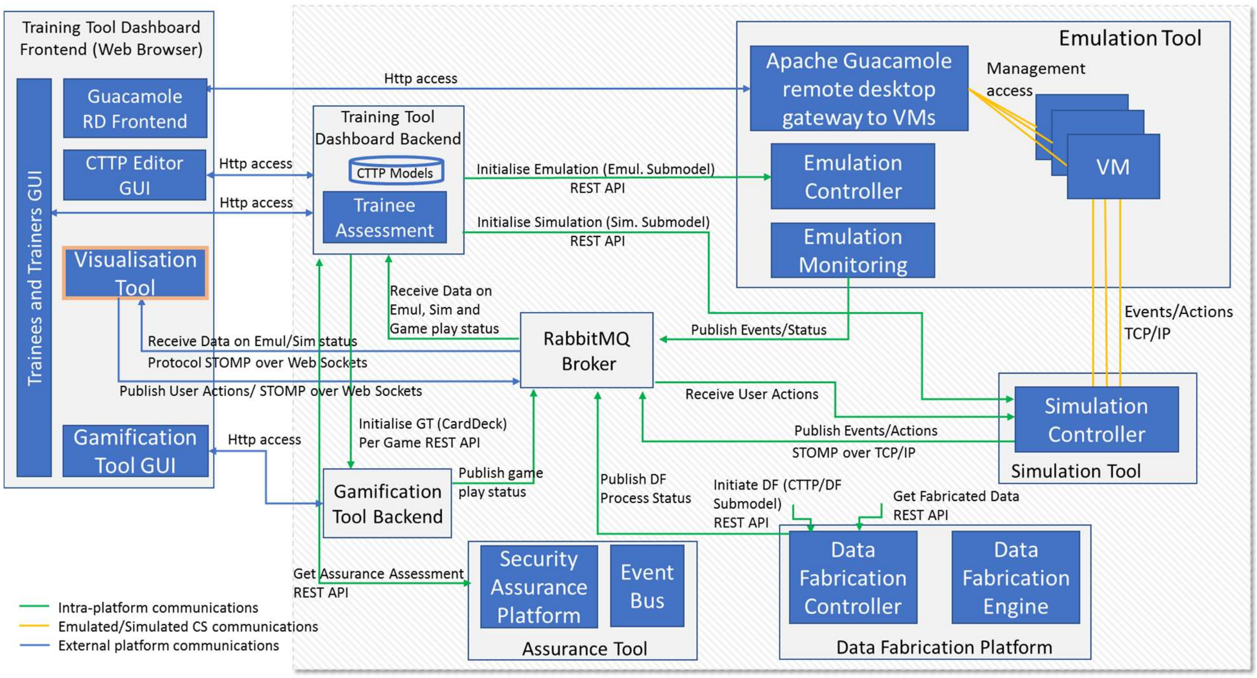

Training Tool Dashboard Frontend (Web Browser)
Trainees and Trainers GUI
Guacamole RD Frontend
CTTP Editor GUI
Visualisation Tool
Gamification Tool GUI
Http access
Http access
Http access
Http access
Training Tool Dashboard Backend
CTTP Models
Trainee Assessment
Initialise Emulation (Emul. Submodel) REST API
Initialise Simulation (Sim. Submodel) REST API
Emulation Tool
Apache Guacamole remote desktop gateway to VMs
Management access
VM
Emulation Controller
Emulation Monitoring
Events/Actions TCP/IP
Receive Data on Emul, Sim and Game play status
Receive Data on Emul/Sim status
Protocol STOMP over Web Sockets
Publish User Actions/ STOMP over Web Sockets
RabbitMQ Broker
Publish Events/Status
Receive User Actions
Publish Events/Actions
STOMP over TCP/IP
Simulation Controller
Simulation Tool
Initialise GT (CardDeck) Per Game REST API
Gamification Tool Backend
Publish game play status
Publish DF Process Status
Initiate DF (CTTP/DF Submodel) REST API
Get Fabricated Data REST API
Get Assurance Assessment REST API
Security Assurance Platform
Event Bus
Assurance Tool
Data Fabrication Controller
Data Fabrication Engine
Data Fabrication Platform
Intra-platform communications
Emulated/Simulated CS communications
External platform communications

Figure 10: THREAT-ARREST Platform Components Interconnection – Visualisation Tool View

Figure 10 shows the THREAT-ARREST platform communications with a particular focus on the Visualisation Tool communications with the message broker of the platform. The Visualisation Tool communications with the message broker use the STOMP[12] over WebSocket. Refer to Section 2 for an introduction to STOMP protocol and rationale of adoption. WebSocket enables web applications to handle bidirectional communications based

[11] https://guacamole.apache.org
[12] http://stomp.github.io/

on variable length frames (of messages), which makes it suitable for the use of STOMP protocol on top[13]. We note that one of the aspects for choosing the RabbitMQ broker was the multi-protocol support provided by the broker's recent versions[14]. This decision addressed an important interoperability issue on the protocol level (e.g. (Soultatos et al., 2019)) among the THREAT-ARREST platform components.

4.1 Interconnection with the Emulation and Simulation Tools

The Visualisation Tool interconnects with the Emulation and Simulation Tools through the means established by the message broker, similarly to the means established for the Training Tool in Section 3.2. Upon a training session initialisation, the Visualisation Tool declares Queues bound to the predefined Exchanges of the Emulation and Simulation Tools.

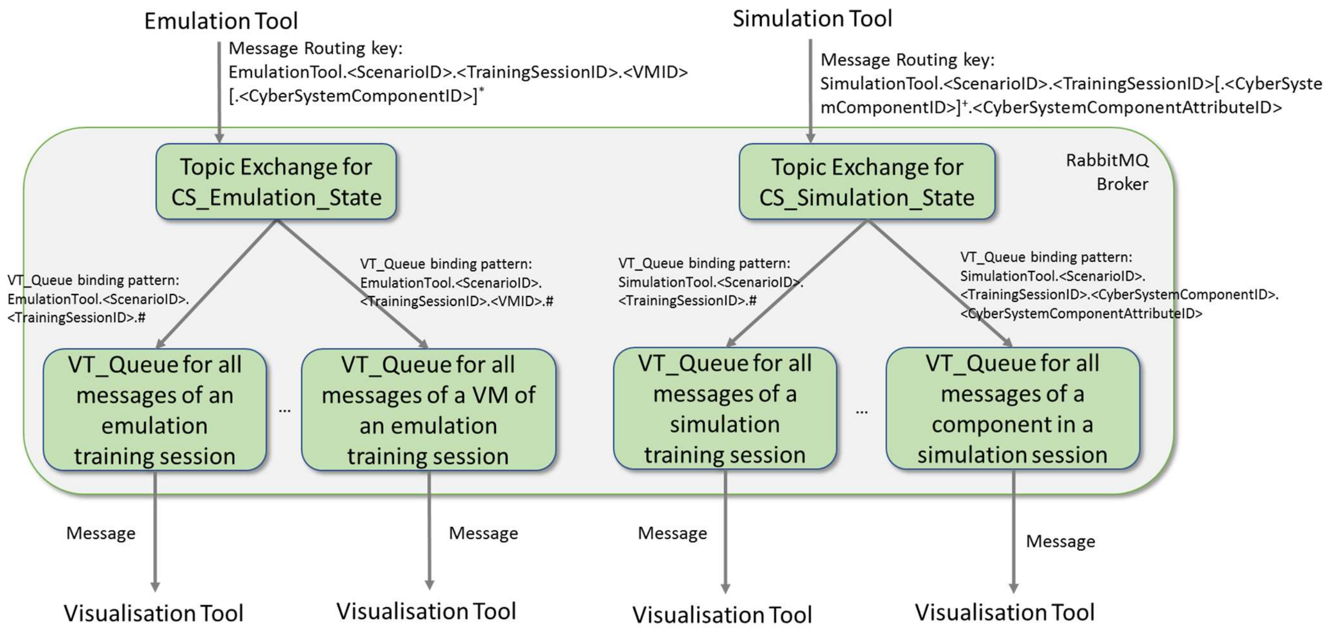

Emulation Tool
Message Routing key:
EmulationTool.<ScenarioID>.<TrainingSessionID>.<VMID>
[.<CyberSystemComponentID>]*
Simulation Tool
Message Routing key:
SimulationTool.<ScenarioID>.<TrainingSessionID>[.<CyberSystemComponentID>]+.<CyberSystemComponentAttributeID>
RabbitMQ Broker
Topic Exchange for CS_Emulation_State
Topic Exchange for CS_Simulation_State
VT_Queue binding pattern: EmulationTool.<ScenarioID>.<TrainingSessionID>.#
VT_Queue binding pattern: EmulationTool.<ScenarioID>.<TrainingSessionID>.<VMID>.#
VT_Queue binding pattern: SimulationTool.<ScenarioID>.<TrainingSessionID>.#
VT_Queue binding pattern: SimulationTool.<ScenarioID>.<TrainingSessionID>.<CyberSystemComponentID>.<CyberSystemComponentAttributeID>
VT_Queue for all messages of an emulation training session
...
VT_Queue for all messages of a VM of an emulation training session
VT_Queue for all messages of a simulation training session
...
VT_Queue for all messages of a component in a simulation session
Message
Message
Message
Message
Visualisation Tool
Visualisation Tool
Visualisation Tool
Visualisation Tool

Figure 11: Interconnection of Visualisation Tool with Emulation and Simulation Tools

Figure 11 shows the Visualisation Tool interconnections with the Emulation and Simulation Tools through the message broker, and in particular the different possible queue binding keys depending on the needs or granularity of the referred cyber system components. The essential aspect of such communications is the *matching* of the routing keys used by the Emulation and Simulation Tools when publishing messages to the broker, and the pattern of the binding key of each queue declared by the Visualisation Tool. The hash/pound symbol '#' is a place holder for zero or more words of the matching process, while the asterisk '*' symbol is a placeholder for one word exactly.

We refer to the Java API example of Code Example 1 of how the Visualisation Tool can create dynamic queues and key bindings for the various cyber system components of a simulation run, and receive messages of their state. For instance, the `bindingKey` of Code Example 1 could be further detailed to include the `currentTemperature` attribute so that specific messages regarding this attribute are routed to this queue.

```
String bindingKey =
"SimulationTool.Energy_BruteForceSSH.TrSess_98374767.mainNet.sensor1.currentTempe
rature";
```

[13] http://jmesnil.net/stomp-websocket/doc/
[14] https://www.rabbitmq.com/protocols.html

4.2 Interconnection with the Simulation Tool on User Actions

It has been identified in the course of the first year, that there is a need for the Simulation Tool to receive actions performed by trainees on specific components/aspects of the cyber system being simulated. The Visualisation Tool enables such user actions to be visualised, captured and communicated to the Simulation Tool. The platform's message broker means are used to enable the Visualisation Tool to interconnect with the Simulation Tool.

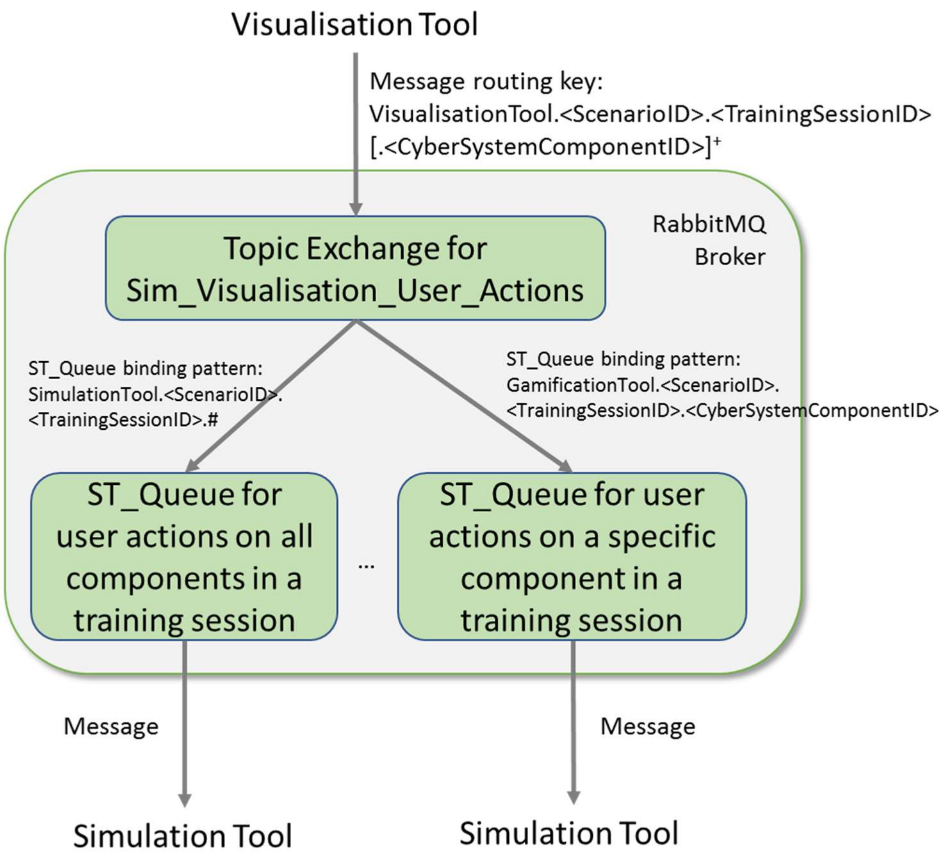

Visualisation Tool
Message routing key:
VisualisationTool.<ScenarioID>.<TrainingSessionID>
[.<CyberSystemComponentID>]+
RabbitMQ Broker
Topic Exchange for Sim_Visualisation_User_Actions
ST_Queue binding pattern:
SimulationTool.<ScenarioID>.<TrainingSessionID>.#
ST_Queue binding pattern:
GamificationTool.<ScenarioID>.<TrainingSessionID>.<CyberSystemComponentID>
ST_Queue for user actions on all components in a training session
...
ST_Queue for user actions on a specific component in a training session
Message
Message
Simulation Tool
Simulation Tool

Figure 12: Interconnection of Visualisation Tool with Simulation Tool on User Actions

Figure 12 shows how the Visualisation Tool interconnects with the Simulation Tool on the performed user actions. An Exchange of the 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. This Topic Exchange is created during set up of the platform and its initial configuration.

In this context, it is important for the Visualisation Tool that all messages sent to the predefined Exchange bear a *well formed* routing key. It was agreed to use the following structure of the routing key:

```
VisualisationTool.<ScenarioID>.<TrainingSessionID>[.<CyberSystemComponentID>]+
```

All four elements of the routing key are *mandatory* and essential to determine the *namespace* of the message.

The constant `VisualisationTool` is 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 and is managed by the Training Tool/Dashboard.

The `<CyberSystemComponentID>` refers to the identifier of a component of the cyber system that is simulated, such as a sensorID, IoTHubID, NetworkNodeID, 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 to be simulated. It is important to note that identifier information for the `<CyberSystemComponentID>` is obtained from the CTTP model. In the next version of the deliverable a formal Backus-Naur Form[15] (BNF) specification will be provided.

The Simulation Tool, upon training session initialisation, dynamically declares Queues and binds those to the predefined Exchange to receive messages of user actions performed. The binding key of each Queue should properly refer to the routing key pattern, as discussed above.

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

public class EmitSimVisUserAction {

  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");
    try (Connection connection = factory.newConnection();
         Channel channel = connection.createChannel()) {

        channel.exchangeDeclare(EXCHANGE_NAME, "topic");

        String routingKey = "VisualisationTool.Energy_BruteForceSSH.TrSess_98374767. Main-
Net.Sensor1";
        String message = "USER_ACTION_DISABLE_SENSOR";

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

Code Example 4: RabbitMQ Java API for Visualisation Tool Topic Exchange Creation and Message Publishing on User Actions

Code Example 4 shows the use of the RabbitMQ Java library/API to connect to a message broker, declare an Exchange of type Topic and publish messages on user actions. In the example, a message for a user action `USER_ACTION_DISABLE_SENSOR` is sent with a routing key `VisualisationTool.Energy_BruteForceSSH.TrSess_98374767.MainNet.Sensor1` indicating that a trainee performed an action to disable sensor1 at the main net of the simulated cyber system.

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

```
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 5: RabbitMQ Java API for Simulation Tool Creation of Queue and Receiving Messages of User Actions

Code Example 5 shows the use of RabbitMQ Java library/API to connect to a broker, declare a queue bound to the Exchange of user actions and receive messages routed to the queue. Particularly, the following binding key is used `VisualisationTool.Energy_BruteForceSSH.TrSess_98374767.MainNet.Sensor1` that matches the routing key of the example in Code Example 4. 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.

5 Gamification Tool Interconnection and REST API

The Gamification Tool provides a REST API for the interconnection with the Training Tool. An example for such an interconnection is shown in Figure 13 that outlines a communication of the Training Tool with the gaming tool PROTECT. The first HTTP request represents the functionality for the creation of a new PROTECT instance. The result or intermediate result of a PROTECT game can be queried with help of the second HTTP request. A more detailed description of the REST API functionality for PROTECT is provided in the subsection 5.1.

At the time this document is written, the second gaming tool AWARENESS QUEST is in its conceptual phase. Because of that, no additional information regarding its API, compared to D1.3 (THREAT-ARREST D1.3, 2019), can be provided at the moment. The specification of the REST API of AWARENESS QUEST will be provided in a later deliverable.

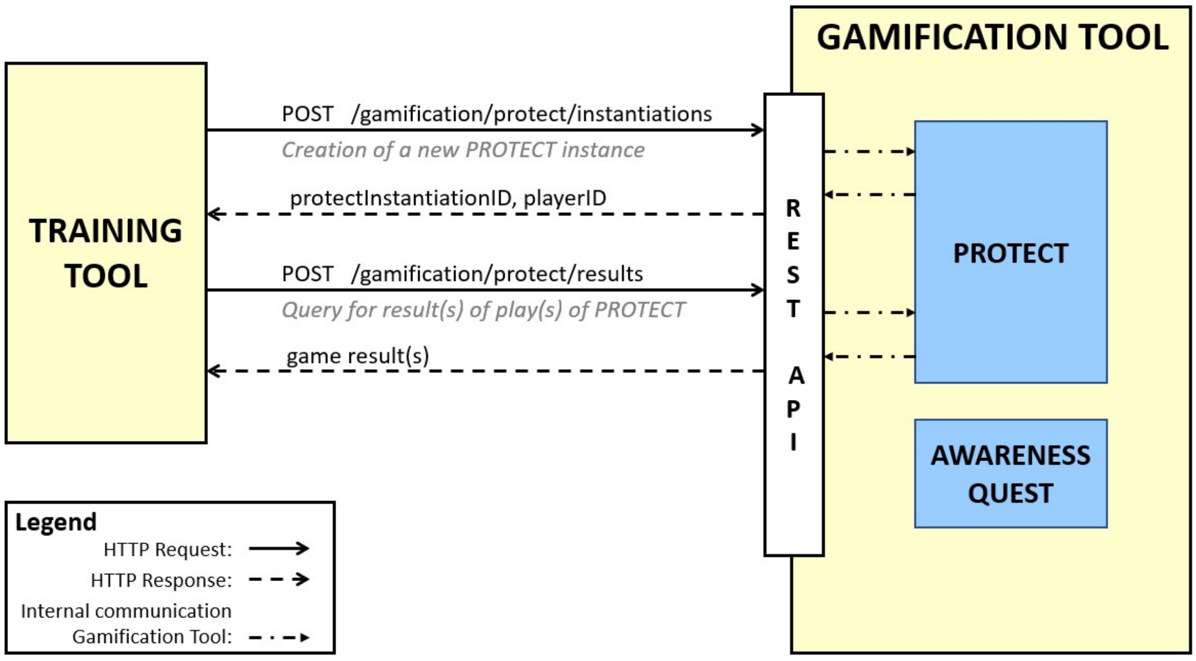

GAMIFICATION TOOL
TRAINING TOOL
POST /gamification/protect/instantiations
Creation of a new PROTECT instance
protectInstantiationID, playerID
POST /gamification/protect/results
Query for result(s) of play(s) of PROTECT
game result(s)
REST API
PROTECT
AWARENESS QUEST
Legend
HTTP Request:
HTTP Response:
Internal communication Gamification Tool:

Figure 13: Example for an interconnection between the Training Tool and the Gamification Tool

5.1 PROTECT REST API

This section discusses the REST API functionality for the gaming tool PROTECT. In this context, Section 5.1.1 describes the functionality for the creation of a PROTECT instance. Section 5.1.2 considers the querying of game results with different filter parameters. It should be pointed out that these specifications of the REST API functionalities will be refined and concretized in further stages of development of the Training and Gamification Tools.

5.1.1 Functionality for creating a PROTECT instance

In the following, the REST API functionality for the creation of a PROTECT instance is described.

Provided functionality

Functionality	Resource	URI (informal)	HTTP method
Creation of a new instance of PROTECT	Instantiations of the game PROTECT	/gamification/protect/instantiations	POST

HTTP request content data

Parameter name	Type	Necessity	Description
senarioID	String	REQUIRED	Scenario that defines the features for a game of PROTECT.
trainingsSessionID	String	REQUIRED	Trainings session in which a game of PROTECT is played.
playerID	String	REQUIRED	Unique identifier of the player
playerName	String	REQUIRED	Name of the player
gameTime	Integer	REQUIRED	Game time in minutes
difficultyLevel	Integer	REQUIRED	Difficulty level with which the game is played. The value of the difficulty level corresponds to a certain configuration of the game within PROTECT.
cardDeckID	String	REQUIRED	Unique identifier of the card deck that shall be played.
specialPractice	Boolean	REQUIRED	Defines if Attack cards that have been solved incorrectly in previous games of PROTECT and the appropriate Defense cards shall appear multiple times in the card deck. • TRUE: The relevant card pairs shall appear multiple times in the card deck. • FALSE: The relevant card pairs shall **not** appear multiple times in the card deck.

HTTP response content data

Parameter name	Type	Necessity	Description
protectInstantiationID	String	REQUIRED	Unique identifier of the created PROTECT instance. This identifier corresponds to the GameID that is used by the Message Broker for the identification of the finish message regarding a certain game of PROTECT (see Section 3.3)
playerID	String	REQUIRED	Unique identifier of the player for whom the PROTECT instance has been created.

5.1.2 Functionality for querying game result(s)

This section considers the REST API functionality for querying game results of PROTECT games. The provided functionality is described in the following table. The appropriate content data for HTTP requests and HTTP responses that correspond to queries with different filter parameters are described in the subsections 5.1.2.3 to 5.1.2.2.

Provided functionality

Functionality	Resource	URI (informal)	HTTP method
Query for results of finished games of PROTECT	Results of finished games of PROTECT	/gamification/protect/results	POST

5.1.2.1 Content data for querying the game results for a certain game

In this section, the content data for a query regarding the game result for a certain game of PROTECT is specified. The Training Tool can perform such a query after it has been notified by the Gamification Tool via the Message Broker (see subsection 3.3) that a certain game of PROTECT has been finished.

HTTP request content data

Parameter name	Type	Necessity	Description
protectInstantiationID	String	REQUIRED	Unique identifier of an instantiation of PROTECT for that the result shall be returned.

HTTP response content data

Parameter name	Type	Necessity	Description
gameResult	Object Data Type	REQUIRED	Result of a game of PROTECT that corresponds to the transferred ID of a PROTECT instantiation.
playerID	String	REQUIRED	Unique identifier of the player of the game
playerName	String	REQUIRED	Name of the player of the game
difficultyLevel	Integer	REQUIRED	Difficulty level of the game.
score	Integer	REQUIRED	Points scored in the game
outcome	Boolean	REQUIRED	Information if the game has been won or lost by the player
unforcedErrors	Array	REQUIRED	Identifiers of Attack cards that have been solved incorrectly, although a correct solution was possible.
attackCardID	String	NOT REQUIRED	Identifier of an Attack card that has been solved incorrectly, although a correct solution was possible.
numberCorrect Solutions	Integer	REQUIRED	Number of Attacks that have been solved correctly during the game.
numberIncorrect Solutions	Integer	REQUIRED	Number of Attacks that have been solved incorrectly during the game.

5.1.2.2 Content data for querying the game results for multiple games

This section considers the content data for a query according the game results for several finished games of PROTECT. The Training Tool gets the information which games of PROTECT have been finished by the Gamification Tool via the Message Broker (see subsection 3.3).

HTTP request content data

Parameter name	Type	Necessity	Description
listProtectInstantiationIDs	Array	REQUIRED	List of unique identifiers of PROTECT instantiations for that the game results shall be returned.
protectInstantiationID	String	REQUIRED	Unique identifier of an instance of PROTECT

HTTP response content data

Parameter name	Type	Necessity	Description
listResults	Array	REQUIRED	List of all game results that correspond to the transferred IDs of PROTECT instantiations.
gameResult	Object Data Type	REQUIRED	Result of a game of PROTECT that corresponds to a transferred PROTECT instantiation ID.
playerID	String	REQUIRED	Unique identifier of the player of the game.
playerName	String	REQUIRED	Name of the player of the game.
difficultyLevel	Integer	REQUIRED	Difficulty level of the game.
score	Integer	REQUIRED	Points scored in the game.
outcome	Boolean	REQUIRED	Information if the game has been won or lost by a player.
unforcedErrors	Array	REQUIRED	Identifiers of Attack cards that have been solved incorrectly, although a correct solution was possible.
attackCardID	String	NOT REQUIRED	Identifier of an Attack card that has been solved incorrectly, although a correct solution was possible.
numberCorrect Solutions	Integer	REQUIRED	Number of Attacks that have been solved correctly during the game.
numberIncorrect Solutions	Integer	REQUIRED	Number of Attacks that have been solved incorrectly during the game.

5.1.2.3 Content data for querying the game result(s) for a certain player

The following specifications of content data consider a query of the game result(s) for a certain player.

HTTP request content data

Parameter name	Type	Necessity	Description
playerID	String	REQUIRED	Unique identifier of a player for whom the game results shall be returned.

HTTP response content data

Parameter name	Type	Necessity	Description
listResults	Array	REQUIRED	List of all game results that correspond to the transferred player ID.
gameResult	Object Data Type	NOT REQUIRED	Result of a game of PROTECT that corresponds to the transferred player ID.
playerID	String	REQUIRED	Unique identifier of the player of the game
playerName	String	REQUIRED	Name of the player of the game
difficultyLevel	Integer	REQUIRED	Difficulty level of the game
score	Integer	REQUIRED	Points scored in the game
outcome	Boolean	REQUIRED	Information if the game has been won or lost by a player
unforcedErrors	Array	REQUIRED	Identifiers of Attack cards that have been solved incorrectly, although a correct solution was possible.
attackCardID	String	NOT REQUIRED	Identifier of an Attack card that has been solved incorrectly, although a correct solution was possible.
numberCorrect Solutions	Integer	REQUIRED	Number of Attacks that have been solved correctly during the game.
numberIncorrect Solutions	Integer	REQUIRED	Number of Attacks that have been solved incorrectly during the game.

5.1.2.4 Content data for querying the game results for multiple players

This section specifies the content data for a query of the game results for multiple players.

HTTP request content data

Parameter name	Type	Necessity	Description
listPlayerIDs	Array	REQUIRED	List of unique identifiers of players for whom the game results shall be returned.
playerID	String	REQUIRED	Unique identifier of a player of the game.

HTTP response content data

Parameter name	Type	Necessity	Description
listResults	Array	REQUIRED	List of all game results that correspond to the transferred player IDs.

Parameter name	Type	Necessity	Description
gameResult	Object Data Type	NOT REQUIRED	Result of a game of PROTECT that corresponds to a transferred player ID.
playerID	String	REQUIRED	Unique identifier of the player of the game.
playerName	String	REQUIRED	Name of the player of the game.
difficultyLevel	Integer	REQUIRED	Difficulty level of the game.
score	Integer	REQUIRED	Points scored in the game.
outcome	Boolean	REQUIRED	Information if the game has been won or lost by a player.
unforcedErrors	Array	REQUIRED	Identifiers of Attack cards that have been solved incorrectly, although a correct solution was possible.
attackCardID	String	NOT REQUIRED	Identifier of an Attack card that has been solved incorrectly, although a correct solution was possible.
numberCorrect Solutions	Integer	REQUIRED	Number of Attacks that have been solved correctly during the game.
numberIncorrect Solutions	Integer	REQUIRED	Number of Attacks that have been solved incorrectly during the game.

6 Conclusions

This document is version 1 of the “Training and Visualization tools IO mechanisms” deliverable (version 2 will be included in the future D4.11 deliverable). It presents the means of the Training and Visualisation Tools interconnect with the other platform components. Different types of communication needs have been addressed such as synchronous vs asynchronous through the REST API and message-broker-enabled communications, respectively. The selected technologies realise the needed communication concepts and enable a loose coupling between the tools which leads to an easier integration of the tools into the THREAT-ARREST platform.

The goal of this first version is to guide the Training and Visualisation Tools integration activities in the second year of the project, particularly how interconnections with the other platform components will be addressed. We recall that this document has two other counterparts that complement the overall view of components interconnections of the THREAT-ARREST platform – deliverables D2.4 and D5.3.

The adoption of REST and RabbitMQ message broker allows us to address interoperability on the API level but also interoperability on the protocol level (Cameron, 2012). For instance, thanks to RabbitMQ, the Visualisation Tool (a JavaScript library running in a trainee’s browser) can interconnect and exchange messages with the Emulation Tool which is running a different message protocol. The Visualisation Tool adopted STOMP over WebSockets for browser-based messages exchanges, while the Emulation Tool the AMQP (see Deliverable 2.4). Both tools can seamlessly exchange messages even using different protocols.

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

- Technical description and specification of interfaces (APIs) for Training Tool and Visualisation Tool communications with other platform components both through REST API and those through RabbitMQ broker. We note that the technical specification of such APIs 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., log data from VMs of the emulation environment, or actions/events from cyber system simulation, etc.) sent by the Emulation and Simulation Tools are processable 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 start in month 13 of the project.

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

[8] 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 .

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

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

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

[17] THREAT-ARREST D4.2 (2019). THREAT-ARREST serious games v1. THREAT-ARREST Project. Available at https://www.threat-arrest.eu/

[18] THREAT-ARREST D5.3 (2019). The Simulation component IO module v1. THREAT-ARREST Project. Available at https://www.threat-arrest.eu/