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

D2.1: Emulated Components Generator Modules v1 †

Abstract: This deliverable provides a technical description of the design and the development of the functionalities of the Emulation Tool that support the generation of the emulated components. It represents the results of the first iteration of task T2.1 activities.

Contractual Date of Delivery	31/08/2019
Actual Date of Delivery	31/08/2019
Deliverable Security Class	Public
Editor	*Ernesto Damiani, Elvinia Riccobene, Stelvio Cimato, Chiara Braghin, Claudio Ardagna, Fulvio Frati, Sadegh Astaneh, Lara Mauri (UMIL)*
Contributors	*UMIL, FORTH, ATOS, ITML*
Quality Assurance	*Dirk Wortmann (SIMPLAN), Oleg Blinder (IBM)*

† *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. *Oleg Blinder (IBM)*

Revisions

Version	Date	By	Overview
1.0	30/08/2019	UMIL	Final version
0.7	28/08/2019	UMIL	Deliverable update after internal review
0.5	30/07/2019	UMIL	Deliverable ready for internal review
0.3	15/07/2019	UMIL	Sections 2-3 added
0.2	25/06/2019	FORTH	Appendix I and II
0.1	01/05/2019	UMIL	First Draft

Executive Summary

This deliverable provides a technical description of the design and the development of the functionalities of the Emulation Tool that support the generation of the emulated components.

The set of Virtual Machines needed to deploy the cyber range infrastructure are automatically created and configured starting from the Cyber Threat and Training Preparation (CTTP) model file defined for the specific training scenario that the tool receives as input.

This deliverable is the result of the first iteration of task T2.1 activities.

Table of Contents

List of Abbreviations

CTTP	Cyber Threat and Training Preparation
HOT	HEAT Orchestration Template
NaaS	Networking-as-a-Service
OS	Operating System
PAL	Platform Level software
QCOW	QEMU Copy-on-Write
REST	Representational State Transfer
SAL	Software Architecture Layer
SSH	Secure Shell
SVG	Scalable Vector Graphics
VM	Virtual Machine
WP	Work Package
XML	eXtensible Markup Language
XSD	XML Schema Definition

List of Tables

List of Figures

1 Introduction

The model-driven approach selected by the THREAT-ARREST Consortium enables the development of high quality cyber ranges having the characteristics of being rapidly reconfigurable and easily deployable. A completely manual configuration process is a complex activity often leading to errors, not always representing the target operating environment, with unpredictable timing (Pridmore, Lardieri, & Hollister, 2010). Furthermore, the combination of tools provided by THREAT-ARREST (Training Tool, Assurance Tool, Emulation Tool, Simulation Tool, Data Fabrication Platform) requires a complex preparation that can be done only if relying on a pre-defined grammar that can be executed and validated by the tools themselves. The interactions among the tools are already modelled in the language, and functionalities and interfaces have already been put in place to be exploited. Indeed, Model-Driven Development has emerged as one of the leading approaches for enabling rapid and collaborative development: in our case, it provides an excellent communication mechanism to align all the partners of the project, ensuring greater quality and more successful outcomes.

The objective of the work of task T2.1, described in this deliverable, is to provide the THREAT-ARREST platform with a concrete tool for the deployment of a virtual training infrastructure, based on the cyber range concept, which can be configured by trainers, by means of the Cyber Threat and Training Preparation (CTTP) models, and used by trainees for hands-on real-world cybersecurity exercise.

Indeed, with the Emulation Tool, the set of Virtual Machines (VMs) needed to deploy the cyber range infrastructure is automatically created and configured starting from an eXtensible Markup Language (XML) file defined for the specific training scenario that the tool receives as input.

In this deliverable we are going to provide a description of the technologies exploited for the implementation of the Emulation Tool, namely OpenStack and HEAT, and of the Emulation Tool package implemented by THREAT-ARREST in order to execute the automatic translation, execution and deployment of the training scenario starting from CTTP models.

The deliverable is organized as follows. Section 2 provides an overview of the technologies the Emulation Tool relies on to deploy the environment. Then, Section 3 describes our solution and the algorithm used to move from a temporary XML file, based on the CTTP Model, to a YAML file that can be used to deploy the training scenario. Section 4 provides an example of a use case that will be further developed in the second iteration of the task activities. Finally, Section 5 gives our conclusions and two Appendixes detail the utilized CTTP files.

2 Emulation Tool Infrastructure

In this Section, we provide an overview of the objectives and of emulation in a training environment based on a cyber-range infrastructure, and a technical description of the solution proposed as result of the activities of the task T2.1.

2.1 Emulation in a Cyber Range-based Training Environment

The TREATH-ARREST platform incorporates two technologies that covers different aspect of modern cyber-ranges environments: *simulation* and *emulation*. Those technologies overlaps in many aspects, and seminal works tried to differentiate between them. In (Davis & Magrath, 2013), cyber ranges are considered *simulations* if they use software models of real world objects to explore behaviour, and they are labelled as *emulations* if they run real software applications on dedicated hardware.

In the THREAT-ARREST project, we support this definition and we consider as *simulated* all the components that do not need a direct interaction with the user in the fulfilling of their tasks. "D5.2 – Simulated Components and network generator v1", due at M12, will provide a complete description of the architecture.

On the other side, all the nodes that require a direct interaction with the trainees, via Secure Shell (SSH) console or Remote Desktop connections, are considered *emulated* and managed by the Emulation Tool.

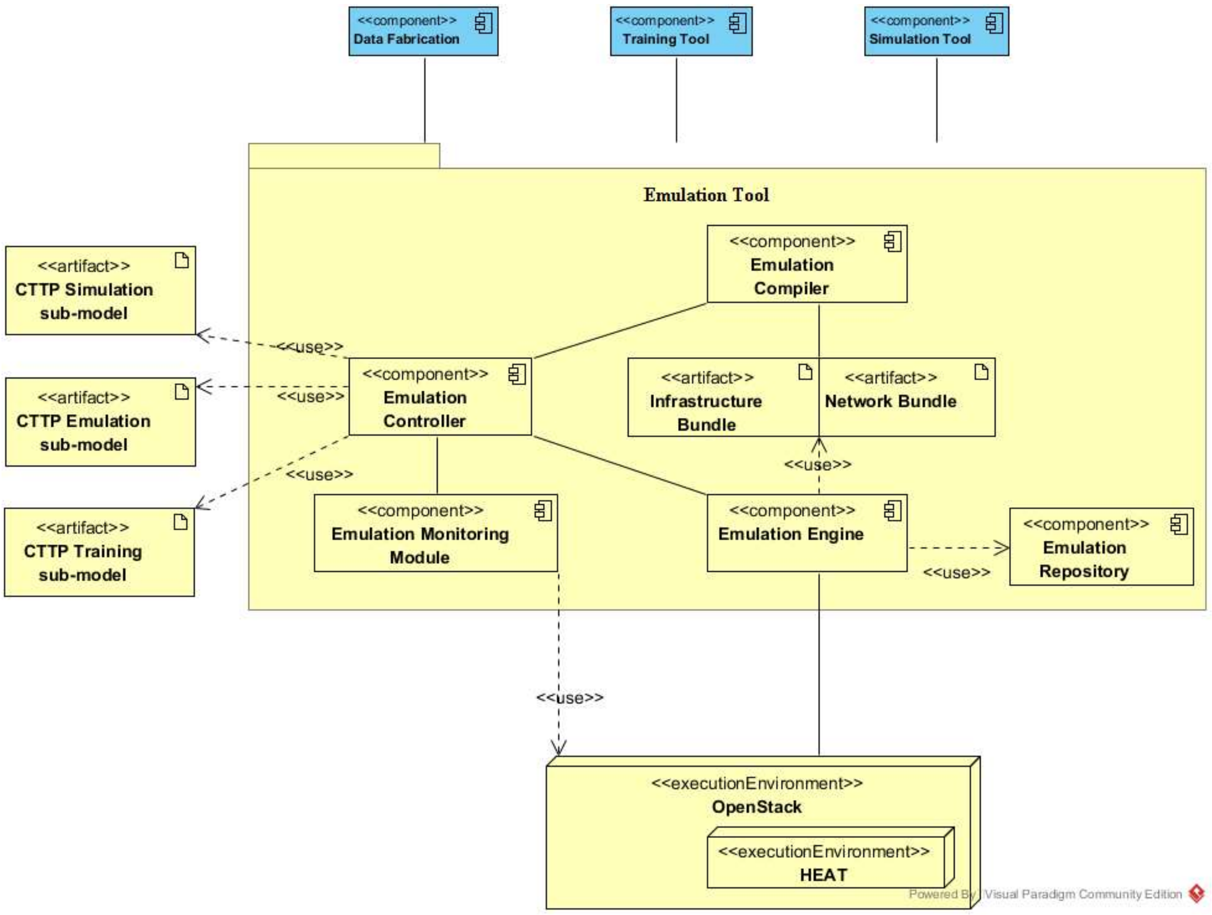
This approach allows to manage a complete and comprehensive set of training scenarios, where all the approach (simulation and emulation) can be exploited individually or at the same time. As for instance, in a mixed training scenario simulated nodes can create network traffic and simulate attacks that trainees, accessing and controlling an emulated machine, can monitor and apply countermeasures.

In the following sections we describe our solution for defining and deploying a cyber range basing on emulated resources. This environment has also designed to incorporated and interact with simulated nodes.

2.2 Emulation Tool Infrastructure

The Emulation Tool infrastructure allows the automatic compiling and deployment of the CTTP model describing the training scenario in actual VMs to be used and accessed by trainers and trainees. Furthermore, it also deploys VMs running the simulation software and provides a control and monitoring infrastructure interacting with the other tools composing the THREAT-ARREST platform.

The Emulation Tool connects via the APIs provided by OpenStack and HEAT frameworks. In the following sections, we provide a description of the two frameworks, focusing on the presentation of the HEAT Orchestration Templates (HOT) used to deploy the scenario. The next figure outlines the main components of the Emulation Tool that will be detailed below.



<<component>> Data Fabrication
<<component>> Training Tool
<<component>> Simulation Tool
Emulation Tool
<<component>> Emulation Compiler
<<artifact>> CTTP Simulation sub-model
<<artifact>> CTTP Emulation sub-model
<<artifact>> CTTP Training sub-model
<<use>>
<<component>> Emulation Controller
<<artifact>> Infrastructure Bundle
<<artifact>> Network Bundle
<<component>> Emulation Monitoring Module
<<component>> Emulation Engine
<<component>> Emulation Repository
<<executionEnvironment>> OpenStack
<<executionEnvironment>> HEAT
Powered By Visual Paradigm Community Edition

Figure 1: Emulation Tool Infrastructure

2.3 OpenStack

OpenStack (OpenStack, 2019) is an open source cloud computing platform. It consists of a collection of interrelated software tools that control pools of processing, storage and networking resources, which can be accessed through RESTful APIs with different language bindings (e.g., Python, and Java) and with common authentication mechanisms.

OpenStack has been chosen as reference architecture for the Emulation Tool thanks to its widespread acceptance and its recognized leadership in the market of open source cloud management infrastructures.

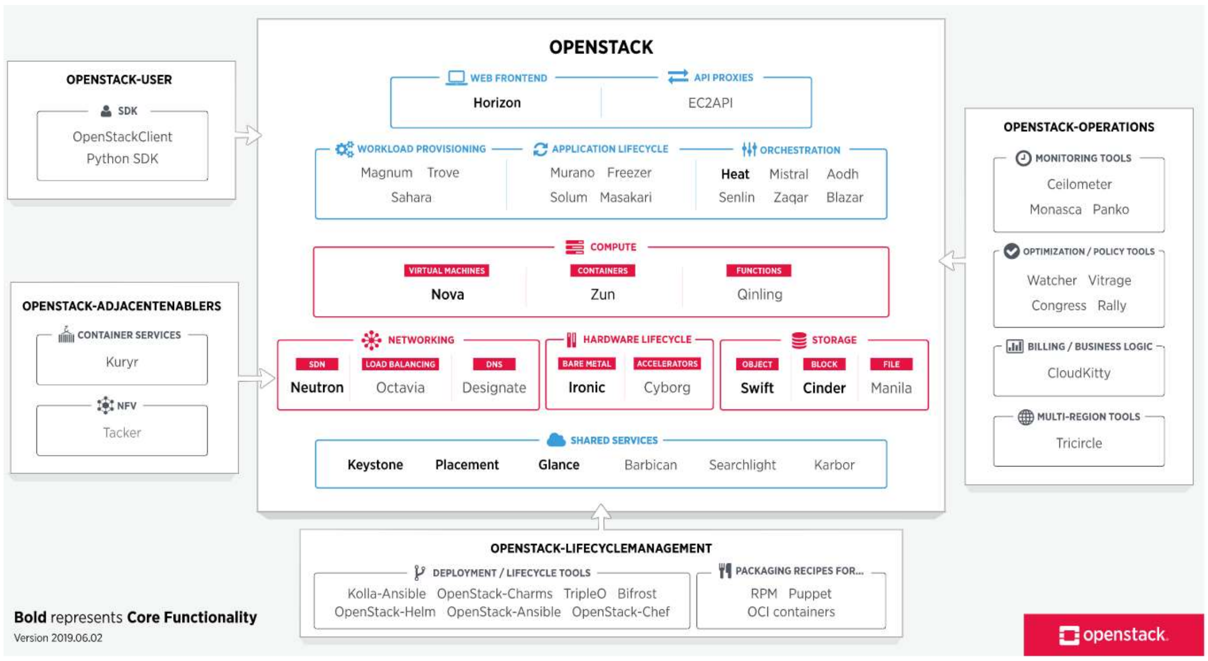


Figure 2: OpenStack core functionalities (OpenStack, 2019e)

The modular composition of OpenStack allows its deployment in different infrastructures and technical requirements. Different projects are developed independently to manage different aspects of Cloud management, and they integrate together into a single product. Figure 2 depicts the projects stack that composes the overall infrastructure.

In particular, the main projects that compose a basic OpenStack infrastructure are the following (OpenStack, 2019e):

- **Nova**: it implements services and associated libraries to provide massively scalable, on demand, self-service access to compute resources, including bare metal, virtual machines, and containers.
- **Neutron**: it is a Software-Defined Networking project focused on delivering networking-as-a-service (NaaS) in virtual computing environments. The exploitation of Neutron as provider of the scenario's virtual network will be provided in deliverable D2.3.
- **Horizon**: it is the canonical implementation of OpenStack's dashboard, extensible and providing a web based user interface to most OpenStack services.
- **Keystone**: it is the OpenStack service that provides API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack's Identity API. It supports LDAP, OAuth, OpenID Connect, SAML and SQL.
- **Swift**: it is a highly available, distributed, eventually consistent object/blob store. Organizations can use Swift to store lots of data efficiently, safely, and cheaply. It is built for scaling and optimized for durability, availability, and concurrency across the entire data set. Swift is ideal for storing unstructured data that can grow without bound.
- **HEAT**: this module orchestrates the infrastructure resources for a cloud application based on templates in the form of text files that can be treated like code. HEAT provides both an OpenStack-native REST API and a Cloud Formation-compatible Query API. It also provides an auto-scaling service that integrates with the OpenStack Telemetry services, so it is possible to include a scaling group as a resource in a template. HEAT will be further examined in Section 2.4.

OpenStack distributes DevStack, a development framework requiring less computational resources, but providing, with some limitations, the developers with the same APIs and functionalities of the full environment (OpenStack, 2019c). DevStack can be installed and deployed in a single VM. At this stage of the implementation, the Emulation Tool will rely on a DevStack environment.

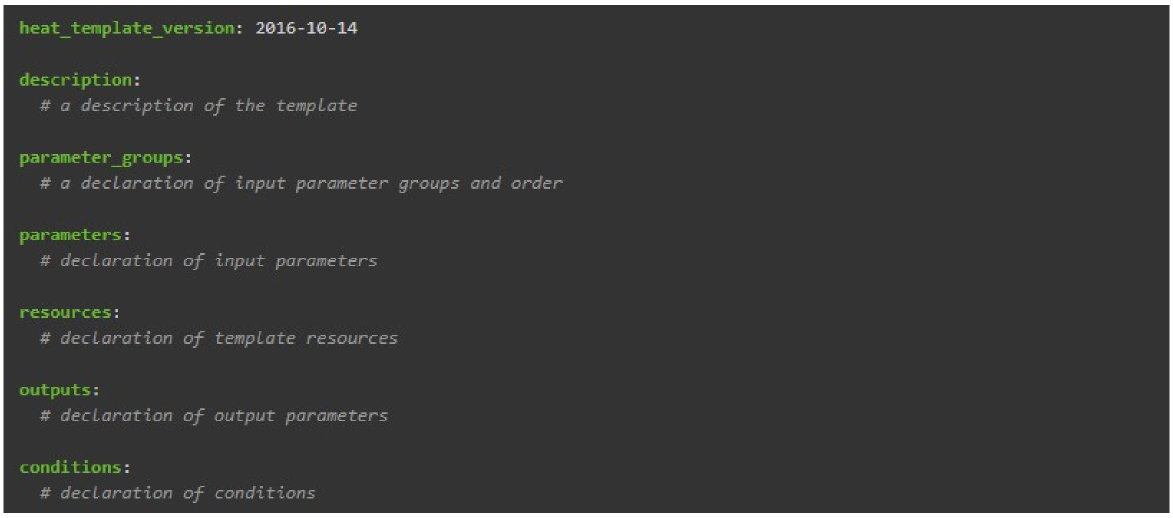
2.4 HEAT

HEAT is an OpenStack project (OpenStack, 2019b) that implements an orchestration engine allowing users to launch multiple composite cloud applications (called stacks) based on templates (HEAT Orchestration Templates - HOTs) in the form of text files readable and writable by humans that can be treated like code by the system. Furthermore, it provides OpenStack API calls to generate running cloud applications. A HOT template allows the creation of most OpenStack resource types (such as instances, networks and subnets, floating IPs, volumes, security groups, users, etc.), as well as some more advanced functionalities such as the possibility to configure software installed on the launched VMs. HOT templates are defined following the YAML notation, readable and writable by humans, and follow the structure outlined in Figure 3. More in detail, each HOT template has to include:

- the HEAT template version key with a valid version of HOT, indicating which version of configuration the system has to consider. The version used in THREAT-ARREST is the latest (2018-08-31).
- The parameters section, that includes all the configurations details that are already been defined in OpenStack and will be used in the actual scenario. The value of each parameter will be referred in the next sections of the file with the command *get_param*.
- The resources section, that contains the declaration of the resources to be deployed, selected from the OpenStack Resource Types (OpenStack, 2019d) list specifying the complete list of possible resources. Each resource type is defined indicating mandatory and optional parameters. The subset of types used in this version of the prototype is included in the subsection 2.4.1. The value of each resource subfield is referred in the file using the command *get_resource*.

The templates allow the creation of most OpenStack resource types (such as instances, floating IP addresses, volumes, security groups, users, etc.), as well as some advanced functionalities such as instance high availability, instance autoscaling, and nested stacks.

HEAT primarily manages infrastructure, but the templates integrate well with software configuration management tools such as Puppet (OpenStack, 2019f) and Ansible (OpenStack, 2019g), to manage scalable and reliable IT automation to OpenStack cloud deployments. Furthermore, developers can customize the capabilities of HEAT by installing plugins.



```
heat_template_version: 2016-10-14

description:
  # a description of the template

parameter_groups:
  # a declaration of input parameter groups and order

parameters:
  # declaration of input parameters

resources:
  # declaration of template resources

outputs:
  # declaration of output parameters

conditions:
  # declaration of conditions
```

Figure 3: HOT file structure (OpenStack, 2019d)

The HOT ends with the output section, which specifies which values the developer asks to be returned by the HEAT APIs when the deployment is completed. In general, returned values are strings generated composing the values of parameters and resources using *get_param* and *get_resource* commands.

2.4.1 HEAT Resource Types

Each resource has a name, a type, and a set of properties. Resource types are organized in packages that clearly indicate the OpenStack project that will manage the instanced object. For instance, VM nodes are defined using the resource type *OS::Nova::Server*, specifying the type of component (*Server*) and the OpenStack project that will instantiate and manage the resource (*Nova*).

In Table 1, we list the subset of resource types that are used by the Emulation Tool, indicating for each one the main properties.

Table 1: OpenStack Resource Types used in the Emulation Tool (properties in bold are required)

Type	Description	Property	
OS::Nova::Server	A Server resource managing the running VM instance	***flavor***	The ID or name of the flavor to boot onto. Flavor can be defined in the file or directly in OpenStack
		image	The ID or name of the image to boot with.
		key_name	Name of keypair to inject into the server
		name	Server name.
		port	ID of an existing port to associate with this server
		network	Name or ID of network to create a port on
		floating_ip	ID of the floating IP to associate
		fixed_ip	Fixed IP address to specify for the port created on the requested network
		mac_address	MAC address to give to this port

Type	Description	Property	
		user_data	User data script to be executed by cloud-init
		user_data_format	How the user_data should be formatted for the server
OS::Neutron::Port	A virtual switch port on a logical network switch. VM attach their interfaces into ports	***network_id***	Network this port belongs to
		mac_address	MAC address to allow through this port
		device_id	Device ID of this port
		name	A symbolic name for this port
		security_groups	Security group IDs to associate with this port
		subnet	Subnet in which to allocate the IP address for this port
OS::Neutron::FloatingIP	Provide public IP addresses to a private cloud, or used to have a “static” IP address reassigned when VMs are upgraded or moved	*floating_network*	Network to allocate floating IP from
		port_id	ID of an existing port with at least one IP address to associate with this floating IP
OS::Neutron::Net	A virtual isolated layer-2 broadcast domain, which can be explicitly configured to be shared.	*dns_domain*	DNS domain associated with this network
		name	Symbolic name for the network, which is not required to be unique
		shared	Define whether this network should be shared across all tenants
		mtu	Maximum transmission unit size(in bytes) for the network
		segments	Segments of this network
OS::Neutron::Subnet	A subnet represents an IP address block that can be used for assigning IP addresses to virtual instances	***network***	The ID of the attached network
		allocation_pools	The start and end addresses for the allocation pools
		cidr	The CIDR assigned to the subnet (e.g. 10.10.10.0/24)
		enable_dhcp	Set to true if DHCP is enabled and false if DHCP is disabled
		gateway_ip	The gateway IP address. If omitted when creation, Neutron will assign the first free IP address within the subnet to the gateway automatically
		host_routes	A list of host route dictionaries for the subnet
		ip_version	The IP version, which is 4 or 6
		name	The name of the subnet
OS::Neutron::Router	A physical or virtual network device that passes network traffic between different networks	***network***	ID or name of the external network for the gateway
		enable_snat	Enables Source NAT on the router gateway
		name	The name of the router
		ha	Indicates whether or not to create a highly available router
OS::Neutron::RouterInterface	Router interfaces associate routers with existing subnets or ports	***router***	The router
		port	Either subnet or port should be specified
		subnet	

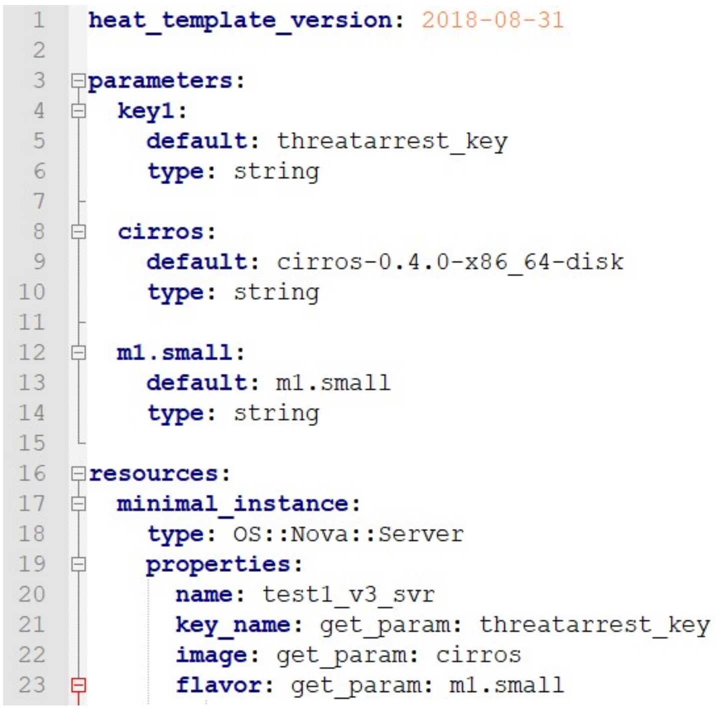
Type	Description	Property	
OS::Neutron::SecurityGroup	Sets of IP filter rules that are applied to an instance's networking	*description*	Description of the security group
		name	A string specifying a symbolic name for the security group, which is not required to be unique
		port_range_max	The minimum and maximum port number in the range that is matched by the security group rule
		port_range_min	
		protocol	The protocol that is matched by the security group rule. Valid values include tcp, udp, and icmp
OS::Nova::Flavor	A resource for creating OpenStack virtual hardware templates	***RAM***	Memory in MB for the flavor
		vcpus	Number of VCPUs for the flavor
		disk	Size of local disk in GB
		name	Name of the flavor
		rxtx_factor	RX/TX factor, defining the aggregate outbound bandwidth, in megabits per second, across all attached network interfaces
		swap	Swap space in MB
OS::Nova::KeyPair	A ssh key that can be injected into a server on launch	***name***	The name of the key pair
		public_key	Allows users to supply the public key from a pre-existing key pair. If not supplied, a new key pair will be generated

2.4.2 Example of HOT

HOT files are defined using the YAML syntax, a human friendly data serialization standard for all programming languages (YAML, 2019). YAML provides support for mappings (hashes/dictionaries), sequences (arrays/lists), and scalars (strings/numbers).

While it can be used with most programming languages, it works best with languages that are built around these data structure types, namely PHP, Python, Perl, JavaScript, and Ruby. One of most visible features of YAML is that it does not allow the use of tabs. Spaces are used instead, as tabs are not universally supported.

The most basic template that can be defined contains only a single compute instance. An example is depicted in Figure 4, where a single resource is specified.



```
heat_template_version: 2018-08-31

parameters:
  key1:
    default: threatarrest_key
    type: string

  cirros:
    default: cirros-0.4.0-x86_64-disk
    type: string

  m1.small:
    default: m1.small
    type: string

resources:
  minimal_instance:
    type: OS::Nova::Server
    properties:
      name: test1_v3_svr
      key_name: get_param: threatarrest_key
      image: get_param: cirros
      flavor: get_param: m1.small
```

Figure 4: HOT basic template

For the sake of simplicity, in this sample template a small instance has been deployed without any connection to the network. Please note that the specification of the virtual network is included in the deliverable “D2.3 – Interlinking of emulated Components module v1”.

In the figure above, the following fields has been used:

- *image*: specifying the image name of the virtual machine the system will deploy. This image is a qemu image QCOW2 of the VM the system have to deploy. The image is stored and indexed in the OpenStack repository.
- *flavor*: the flavor *m1.small* is one of the default flavors provided by default in OpenStack that can be used for quick deploys. In this flavor, vcpus, disk and RAM have been set at respectively 1 vcpu, 20GB and 2048MB. Other default flavors provided by OpenStack are *m1.tiny*, *m1.medium*, *m1.large* and *m1.xlarge*.
- *key*: the public SSH key to be injected into the instance on launch, in order to access it by connecting via SSH using the private key. The key has already been included in OpenStack and referred using its ID.
- *network*: the VM must be connected at least to a network. In general, it is possible to configure richer network topologies, by creating and configuring multiple networks and subnets, and attaching virtual devices to ports on these networks.

To set up the minimal system, the YAML file will be specified as input to the Heat API, which will take care of creating the VM instance and deploying it in the overall environment.

However, the scope of Heat is to deploy complex systems with multiple virtual machines and networks connected with virtual routers. The resulting template objects cannot be directly included in the CTTP models due to its intrinsic complexity and verbosity.

For this reason, the Emulation Tool has been designed to provide the user with mechanism and interface to deploy complex systems starting from the CTTP Emulation sub-model, written in XML, and give access to trainees and trainers to the deployed VMs via a simple web interface.

In the following sections, an overview of the Emulation Tool prototype will be provided, describing its structure and the compiling algorithm that has been used to produce the YAML template starting from the (CTTP) XML file in input.

3 Emulation Tool Modules for Components Generation

In this section, the software packages that compose the Emulation Tool are described in detail and depicted in Figure 5. The work is the result of the first iteration of the tasks T2.1 and T2.3, whose activities are strictly correlated.

For each package, a description of the technology and the algorithms are provided. Please note that all the details about the generation of virtual networks are demanded to the deliverable "D2.3 – Interlinking of emulated Components module v1".

In the second iteration of task T2.1, the tool will be finalized using the CTTP Models developed in the work package (WP) 3 for the definition of the training environment, and the second version of the deliverable, due at M24, will contain a detailed description on how setting up the images and the software contained in the use case virtual machines.

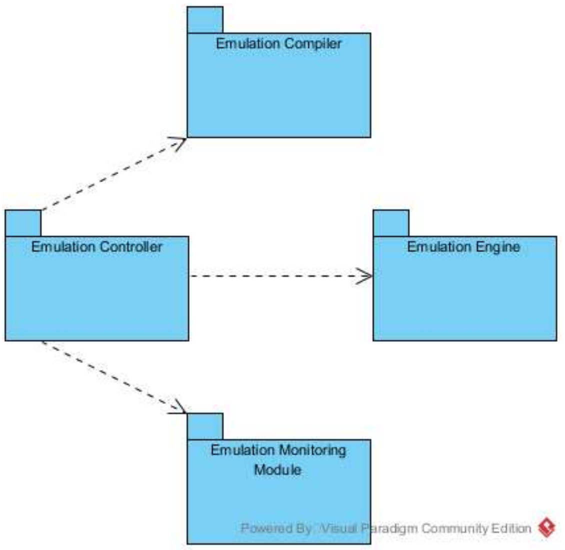

Emulation Compiler
Emulation Controller
Emulation Engine
Emulation Monitoring Module
Powered By: Visual Paradigm Community Edition

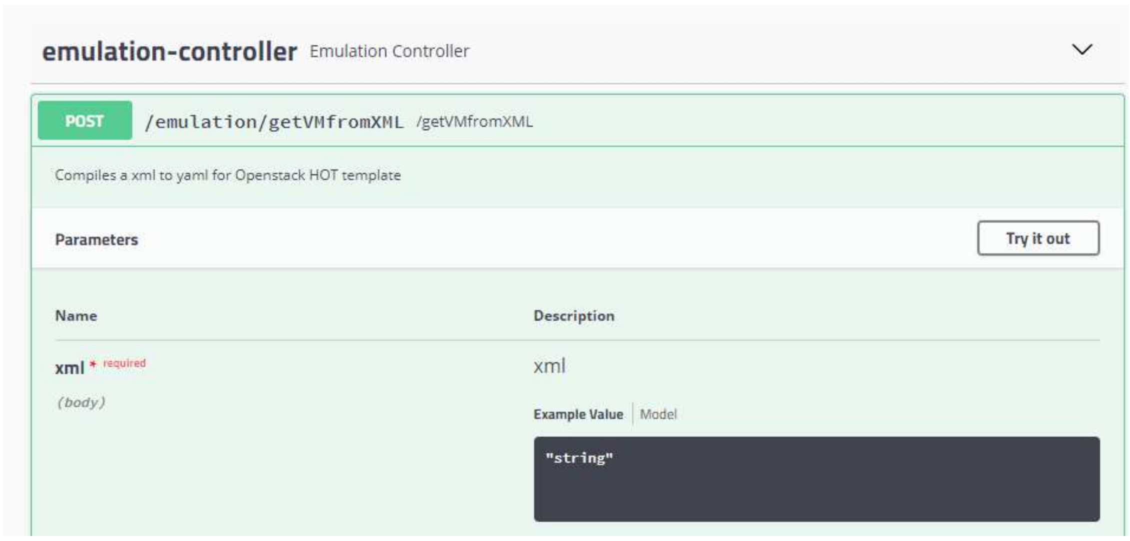
Figure 5: Emulation Tool package structure

3.1 Emulation Controller

The Emulation Controller is the module that manages the communication interface among the Emulation Tool and the other tools that compose the THREAT-ARREST platform. Major details on the communication among the tools are provided in deliverable "D2.4 – Emulation Tool Interoperability Module v1", due at M12.

The controller acts as the front door to the Emulation Tool. It provides a REST interface with a set of methods that implement the creation and deployment of the virtual training scenario. The model-driven approach of the THREAT-ARREST platform represents its core aspect and each tool has been designed to execute their operation starting from their customized view of the CTTP Model defined for a specific training activity. These views are called *Sub-models* and are typical for each tool. A working example of the sub-model is provided in Section 3.2, but this example is subject to be changed since the CTTP model is also will be finalized.

In the Emulation Tool, the *Emulation Sub-model* is requested as unique input for the deploying of the training scenario, composed of virtual machines and networks (Figure 6). The interface can be easily extended to supply additional services to the THREAT-ARREST tools, integrating new methods in the Java code as shown in Figure 7.


emulation-controller Emulation Controller

POST /emulation/getVMfromXML /getVMfromXML

Compiles a xml to yaml for Openstack HOT template

Parameters

Name	Description
xml * required (body)	xml

Example Value | Model

```
"string"
```

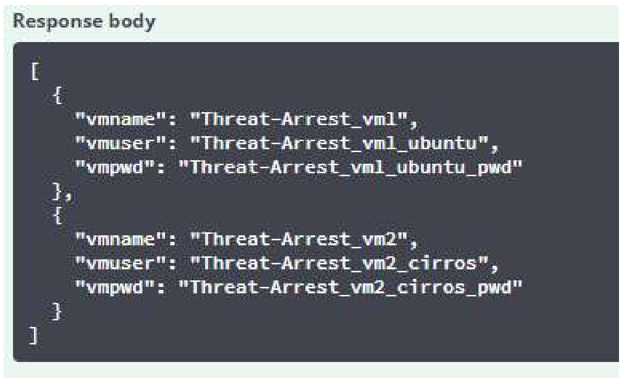
Figure 6: Emulation Controller REST interfaces.

```
package it.unimi.threatarrest.emulation.controller;

@RestController
@RequestMapping("/emulation")
public class EmulationController
{
    @Autowired
    private EngineService engineService;
    @Autowired
    private CompilerService compilerService;

    @RequestMapping(value = "/getVMfromXML",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON_VALUE)
    @ApiOperation(value = "/getVMfromXML",
        notes = "Compiles a xml to yaml for Openstack HOT template")
    public ResponseEntity getVMfromXML ( @RequestBody(required=true) String xml)
        throws IOException {
        String template = compilerService.generateYaml(xml);
        JAXBContext jaxbContext;
        Scenario scen = null;
        try {
            jaxbContext = JAXBContext.newInstance(Scenario.class);
            Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
            scen = (Scenario) jaxbUnmarshaller.unmarshal(new StringReader(xml));

        } catch (JAXBException e) {
            e.printStackTrace();
        }

        OSClientV3 os = engineService.getOsclientv3();
        TemplateResponse result = os.heat().templates().validateTemplate(template);
        MyStack stack = new MyStack();
        Map<String, Object> map = compilerService.createStack(scen.getName(), template);
        stack.setStackName(scen.getName());
        stack.setTemplate(template);
        stack.setValid(result.isValid());
        stack.setVms((List<VMServer>)map.get("vms"));
        return ResponseEntity.ok().body(map.get("vms"));
    }

}
```

Figure 7: emulation/getVMfromXML REST interface code.

The interface shown in Figure 7, implementing the method that accepts an XML as input and calls the Emulation Compiler to create and execute the YAML code. It exploits Swagger (SmartBear, 2019), an open-source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful Web services. Its toolset includes support for automated documentation, code generation, and test-case generation.

The method, if the creation of the training scenario is successful, returns the set of credentials needed to access the VM through the web based interface explained in the deliverable D2.3. If during the execution exceptions occur, the method will return an error message. Figure 8 provides an example of the values returned in case of successful execution, indicating the name, user and password needed to access the deployed VMs.



```
[
  {
    "vmname": "Threat-Arrest_vm1",
    "vmuser": "Threat-Arrest_vm1_ubuntu",
    "vmpwd": "Threat-Arrest_vm1_ubuntu_pwd"
  },
  {
    "vmname": "Threat-Arrest_vm2",
    "vmuser": "Threat-Arrest_vm2_cirros",
    "vmpwd": "Threat-Arrest_vm2_cirros_pwd"
  }
]
```

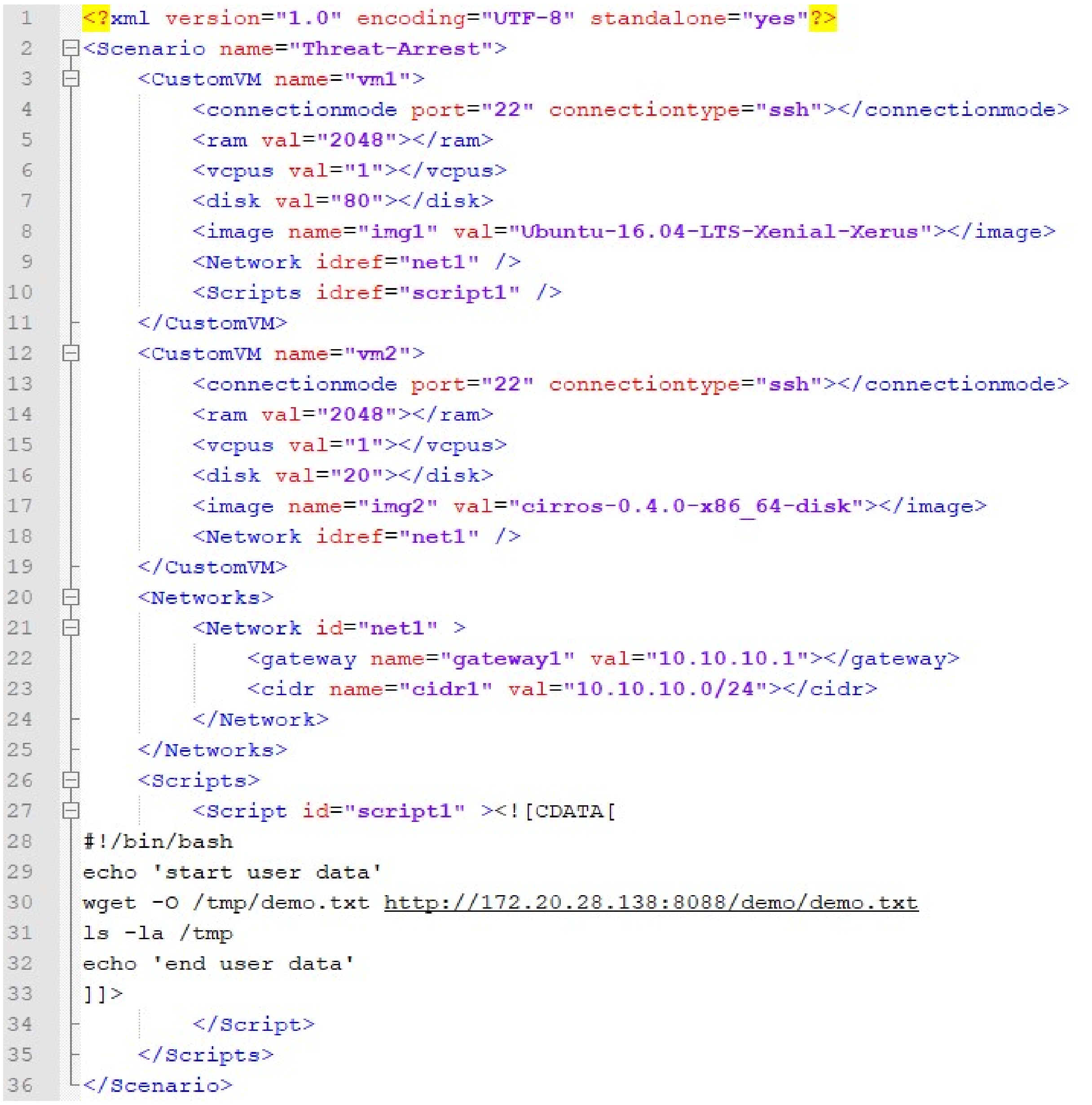
Figure 8: emulation/getVMfromXML response values in the YAML format.

3.2 Emulation Compiler

The Emulation Compiler is the module of the Emulation Tool responsible of translating the CTTP Emulation sub-model, given as input to the Controller, in a valid HOT YAML file. The final version of the CTTP model will be ready after the release of this deliverable, for this reason we based our prototype on an input XML file including all the properties that will be specified in the final CTTP model, even if the structure would be different. However, the chosen implementation technique will allow the developer team to adapt it easily to the final schema.

Figure 9 shows the XML that the prototype takes as input. It describes a simple scenario with two VMs, characterized by specific hardware properties (flavor), which share a common network.

In the following, a description of the algorithm followed to create the YAML file is provided, including code snippet to show the specific exploited technologies.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Scenario name="Threat-Arrest">
    <CustomVM name="vm1">
        <connectionmode port="22" connectiontype="ssh"></connectionmode>
        <ram val="2048"></ram>
        <vcpus val="1"></vcpus>
        <disk val="80"></disk>
        <image name="img1" val="Ubuntu-16.04-LTS-Xenial-Xerus"></image>
        <Network idref="net1" />
        <Scripts idref="script1" />
    </CustomVM>
    <CustomVM name="vm2">
        <connectionmode port="22" connectiontype="ssh"></connectionmode>
        <ram val="2048"></ram>
        <vcpus val="1"></vcpus>
        <disk val="20"></disk>
        <image name="img2" val="cirros-0.4.0-x86_64-disk"></image>
        <Network idref="net1" />
    </CustomVM>
    <Networks>
        <Network id="net1" >
            <gateway name="gateway1" val="10.10.10.1"></gateway>
            <cidr name="cidr1" val="10.10.10.0/24"></cidr>
        </Network>
    </Networks>
    <Scripts>
        <Script id="script1" ><![CDATA[
#!/bin/bash
echo 'start user data'
wget -O /tmp/demo.txt http://172.20.28.138:8088/demo/demo.txt
ls -la /tmp
echo 'end user data'
]]>
        </Script>
    </Scripts>
</Scenario>
```

Figure 9: Example of XML input file

3.2.1 Compiler Algorithm

The compiler algorithm is mainly composed of three steps:

1) Definition of the exploited parameters;
2) Definition of the OpenStack resources;
3) Definition of outputs.

Definition of the exploited parameters

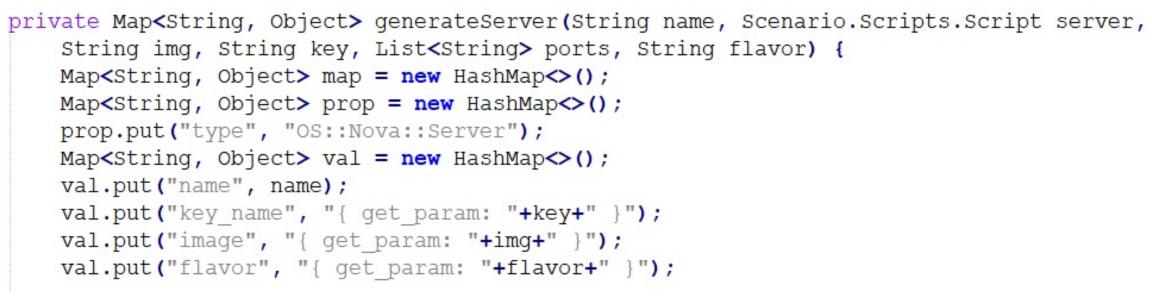
Some of the variables indicated in the XML files or already included in OpenStack are referred as variable.

1) For each CustomVM, collect the values needed to define the flavour tuple <*image, vcpu, RAM, disk*>;
2) Create the flavor based on the tuple using the command *generateFlavor()* and store it in OpenStack, giving as name the composition of tuple values;
3) For each CustomVM, verify that the specified image is present in the OpenStack repository, otherwise raise an error;
4) Get the ID of the public virtual network that provides access to internet;
5) Get the common key used to access the generated VMs;

Definition of the OpenStack resources

The following algorithm describes the specification of the single VM, without caring of the network configurations that are included in the deliverable D2.3.

1) For each CustomVM, define a node of type *OS::Nova::Server;*
2) Add a line *flavor:* with the specified flavor, getting the specific parameter with the command *get_param:<flavor tuple>;*
3) Add a line *image:* with the specified image, getting the specific parameter with the command *get_param:<image name>;*
4) Add a line *key:* getting the specific parameter with the command *get_param: <key name>;*
5) Add a line *name:* with the name of the VM;
6) If specified, add a line *user_data:* and include the script specified in the input; the script will be run by the daemon *cloud-init* when the deployment of the VM is complete;
7) Put all the data in a HashMap variable, as shown in Figure 10;
8) Serialize the HashMap to generate the YAML file.



```
private Map<String, Object> generateServer(String name, Scenario.Scripts.Script server,
    String img, String key, List<String> ports, String flavor) {
    Map<String, Object> map = new HashMap<>();
    Map<String, Object> prop = new HashMap<>();
    prop.put("type", "OS::Nova::Server");
    Map<String, Object> val = new HashMap<>();
    val.put("name", name);
    val.put("key_name", "{ get_param: "+key+" }");
    val.put("image", "{ get_param: "+img+" }");
    val.put("flavor", "{ get_param: "+flavor+" }");
```

Figure 10: VM generation code.

Definition of outputs

The following algorithm describes the specification of output variables, needed by the tool for the communication of username and password of VMs.

1) For each CustomVM
    a. Add a line *get_attr:* applied to the list <*name of VM resource*>, *first_address*, returning the name and private address of the VM;

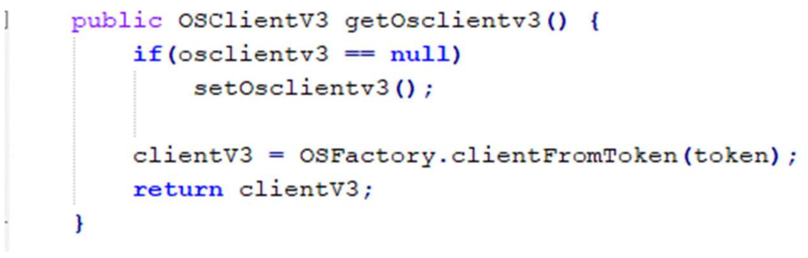
b. Add a line *get_attr:* applied to the list *<name of VM floating_ip resource>, floating_ip_address*, returning the name and floating ip address of the VM;

3.3 Emulation engine

The module works as interface of the OpenStack Java API (`openstack4j`) in order to allow the deployment of the actual training scenario, starting from the template produced by the Emulation Compiler (OpenStack4j, 2019).

The core of the OpenStack API for the creation of a new stack is the class `OSClientV3`, included in the package `org.openstack4j.api.OSClient`. The class create and authenticate a new client object, authorized to operate on the actual OpenStack installation.

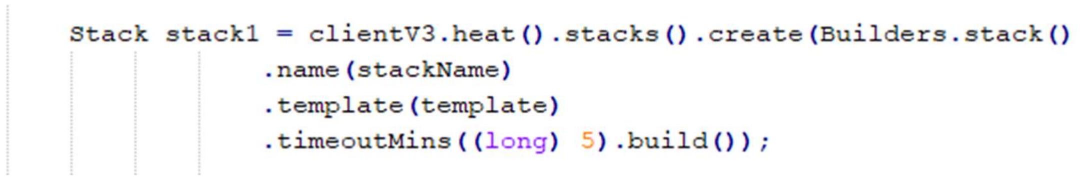
Through the method `getOsclientv3()`, shown in Figure 11, the Engine call the OpenStack Factory OSFactory requiring the creation of a new client.



```
public OSClientV3 getOsclientv3() {
    if(osclientv3 == null)
        setOsclientv3();

    clientV3 = OSFactory.clientFromToken(token);
    return clientV3;
}
```

Figure 11: Creation of the object clientV3

The object is then used to access the specific HEAT interface `HeatService` for the creation of a new stack starting from a predefined YAML template, a given name, and a predefined timeout in minutes (Figure 12).



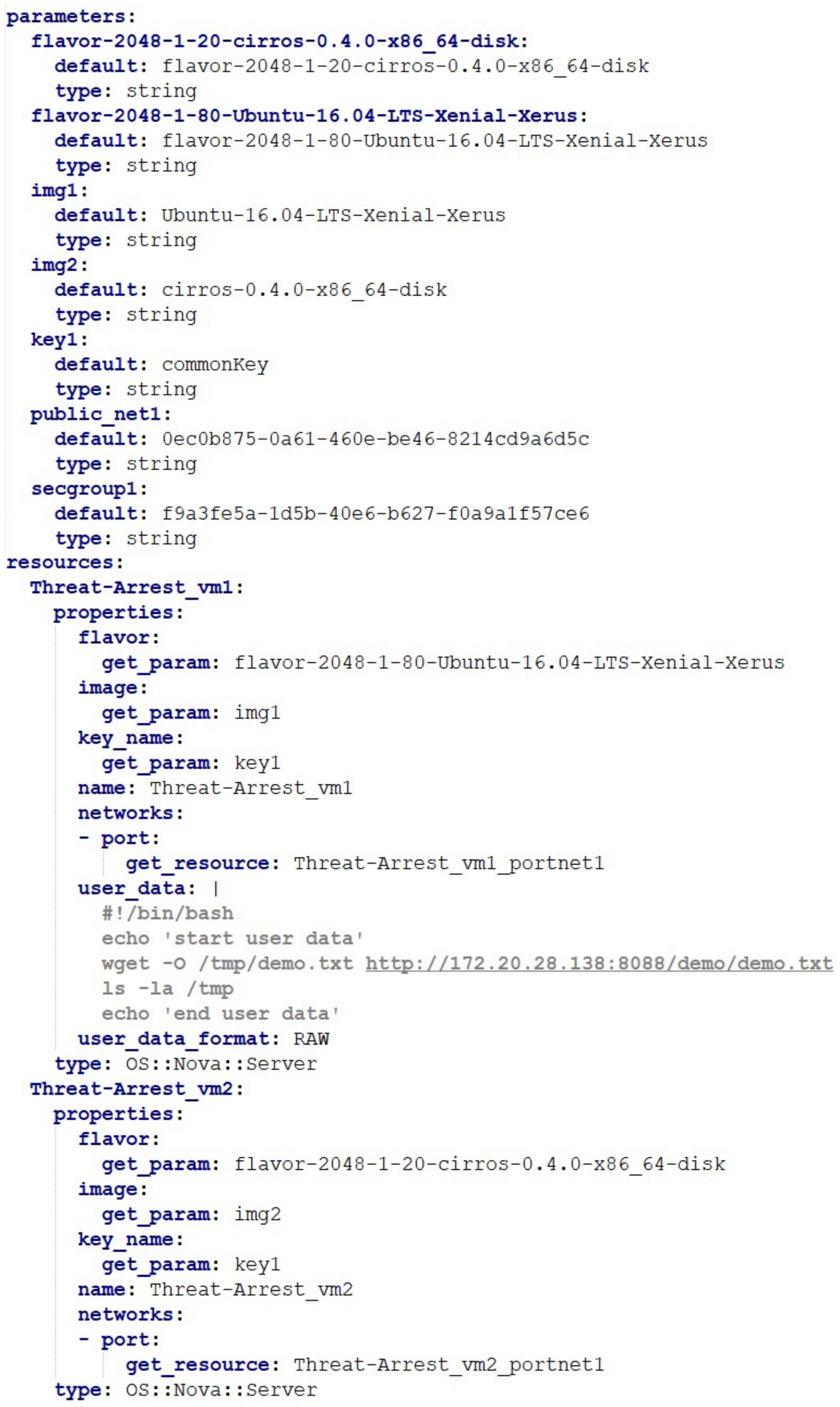
```
Stack stack1 = clientV3.heat().stacks().create(Builders.stack()
            .name(stackName)
            .template(template)
            .timeoutMins((long) 5).build());
```

Figure 12: Creation of the stack.

Furthermore, the Engine provides methods for the querying of the OpenStack instance, in terms of already created flavors, images, networks, subnets, etc., and eventually their creation if needed.

3.4 Result

The application of the Emulation Tool with the output specified in Figure 9 led to the definition of the YAML presented in Figure 13, which takes into consideration only the parameters and resources used for the definition of the VMs.



```yaml
parameters:
  flavor-2048-1-20-cirros-0.4.0-x86_64-disk:
    default: flavor-2048-1-20-cirros-0.4.0-x86_64-disk
    type: string
  flavor-2048-1-80-Ubuntu-16.04-LTS-Xenial-Xerus:
    default: flavor-2048-1-80-Ubuntu-16.04-LTS-Xenial-Xerus
    type: string
  img1:
    default: Ubuntu-16.04-LTS-Xenial-Xerus
    type: string
  img2:
    default: cirros-0.4.0-x86_64-disk
    type: string
  key1:
    default: commonKey
    type: string
  public_net1:
    default: 0ec0b875-0a61-460e-be46-8214cd9a6d5c
    type: string
  secgroup1:
    default: f9a3fe5a-1d5b-40e6-b627-f0a9a1f57ce6
    type: string
resources:
  Threat-Arrest_vm1:
    properties:
      flavor:
        get_param: flavor-2048-1-80-Ubuntu-16.04-LTS-Xenial-Xerus
      image:
        get_param: img1
      key_name:
        get_param: key1
      name: Threat-Arrest_vm1
      networks:
      - port:
          get_resource: Threat-Arrest_vm1_portnet1
      user_data: |
        #!/bin/bash
        echo 'start user data'
        wget -O /tmp/demo.txt http://172.20.28.138:8088/demo/demo.txt
        ls -la /tmp
        echo 'end user data'
      user_data_format: RAW
    type: OS::Nova::Server
  Threat-Arrest_vm2:
    properties:
      flavor:
        get_param: flavor-2048-1-20-cirros-0.4.0-x86_64-disk
      image:
        get_param: img2
      key_name:
        get_param: key1
      name: Threat-Arrest_vm2
      networks:
      - port:
          get_resource: Threat-Arrest_vm2_portnet1
    type: OS::Nova::Server
```

Figure 13: HOT YAML file

The HOT has been then executed by the Emulation engine, as described in the subsection 3.3, to deploy the scenario.

The successful conclusion of the process has been confirmed by the response of the REST method, and can also be validated accessing the OpenStack framework and verifying if the new stack is actually included in the list of the deployed stacks.

Figure 14 and Figure 15 give an evidence of the actual creation of the stack. In particular, Figure 15 reports the list of all the resources that have been deployed, where in particular we can find the two VMs indicated in the XML: *Threat-Arrest_vm1* and *Threat-Arrest_vm2*.

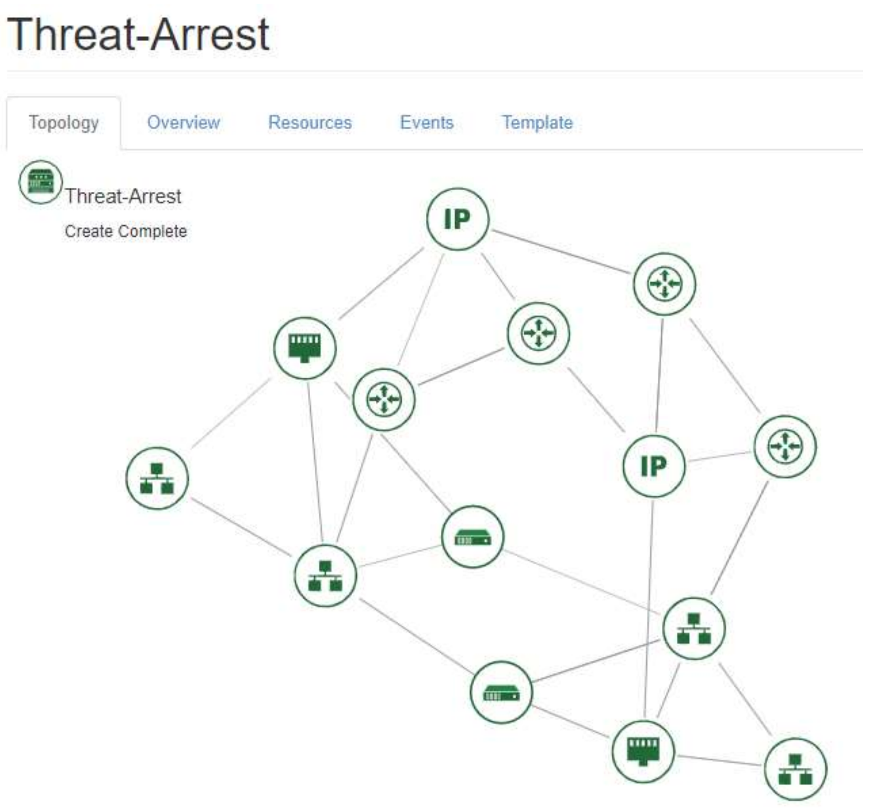
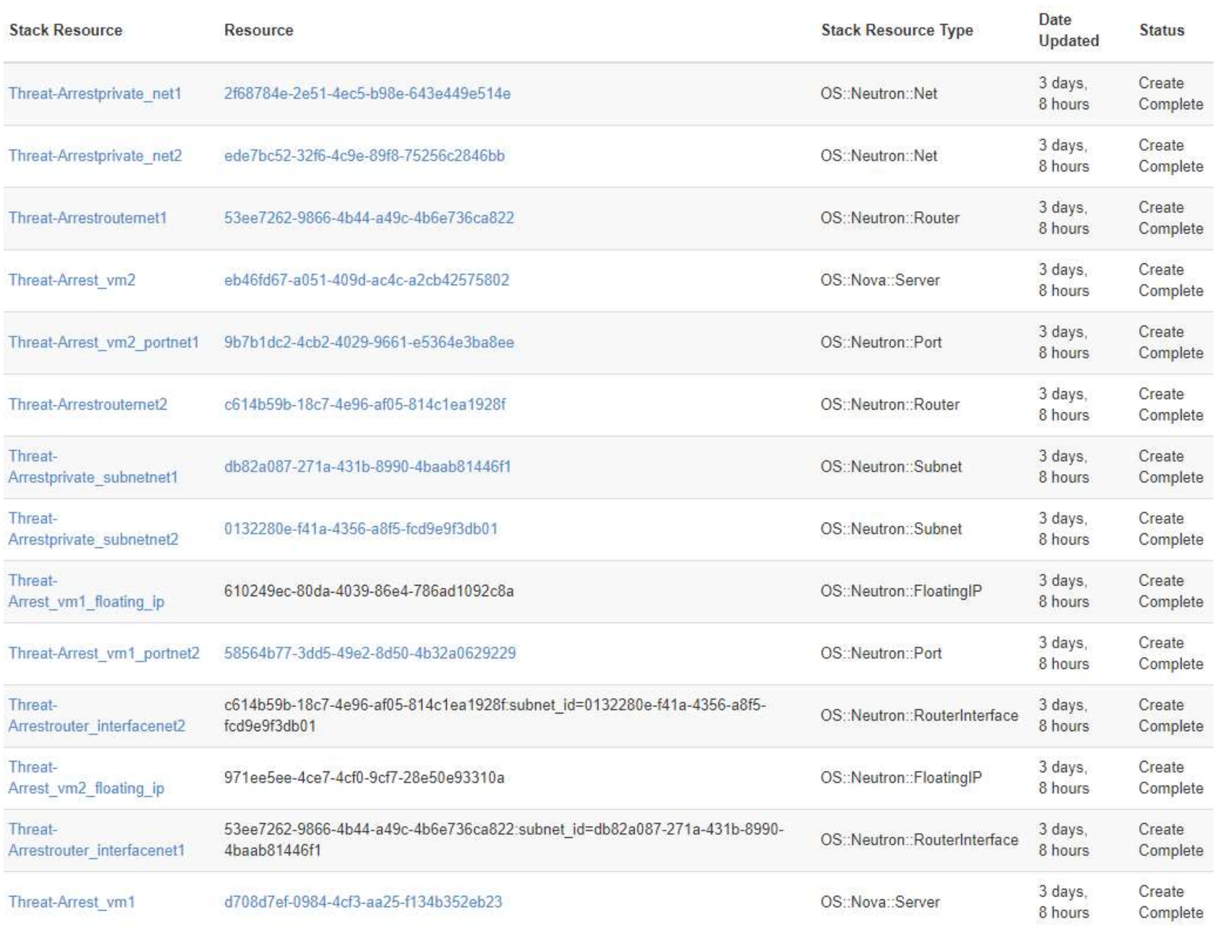


Figure 14: Stack topology.



Stack Resource	Resource	Stack Resource Type	Date Updated	Status
Threat-Arrestprivate_net1	2f68784e-2e51-4ec5-b98e-643e449e514e	OS::Neutron::Net	3 days, 8 hours	Create Complete
Threat-Arrestprivate_net2	ede7bc52-32f6-4c9e-89f8-75256c2846bb	OS::Neutron::Net	3 days, 8 hours	Create Complete
Threat-Arrestrouternet1	53ee7262-9866-4b44-a49c-4b6e736ca822	OS::Neutron::Router	3 days, 8 hours	Create Complete
Threat-Arrest_vm2	eb46fd67-a051-409d-ac4c-a2cb42575802	OS::Nova::Server	3 days, 8 hours	Create Complete
Threat-Arrest_vm2_portnet1	9b7b1dc2-4cb2-4029-9661-e5364e3ba8ee	OS::Neutron::Port	3 days, 8 hours	Create Complete
Threat-Arrestroutemet2	c614b59b-18c7-4e96-af05-814c1ea1928f	OS::Neutron::Router	3 days, 8 hours	Create Complete
Threat-Arrestprivate_subnetnet1	db82a087-271a-431b-8990-4baab81446f1	OS::Neutron::Subnet	3 days, 8 hours	Create Complete
Threat-Arrestprivate_subnetnet2	0132280e-f41a-4356-a8f5-fcd9e9f3db01	OS::Neutron::Subnet	3 days, 8 hours	Create Complete
Threat-Arrest_vm1_floating_ip	610249ec-80da-4039-86e4-786ad1092c8a	OS::Neutron::FloatingIP	3 days, 8 hours	Create Complete
Threat-Arrest_vm1_portnet2	58564b77-3dd5-49e2-8d50-4b32a0629229	OS::Neutron::Port	3 days, 8 hours	Create Complete
Threat-Arrestrouter_interfacenet2	c614b59b-18c7-4e96-af05-814c1ea1928f:subnet_id=0132280e-f41a-4356-a8f5-fcd9e9f3db01	OS::Neutron::RouterInterface	3 days, 8 hours	Create Complete
Threat-Arrest_vm2_floating_ip	971ee5ee-4ce7-4cf0-9cf7-28e50e93310a	OS::Neutron::FloatingIP	3 days, 8 hours	Create Complete
Threat-Arrestrouter_interfacenet1	53ee7262-9866-4b44-a49c-4b6e736ca822:subnet_id=db82a087-271a-431b-8990-4baab81446f1	OS::Neutron::RouterInterface	3 days, 8 hours	Create Complete
Threat-Arrest_vm1	d708d7ef-0984-4cf3-aa25-f134b352eb23	OS::Nova::Server	3 days, 8 hours	Create Complete

Figure 15: OpenStack resources created.

4 CTTP Model-Driven Emulation

4.1 Development sub-model

The THREAT-ARREST platform operation will be driven by the Cyber Threat and Training Preparation (CTTP) models. Among others, a CTTP model contains information regarding how to instantiate the various platform modules and facilitate the training procedures. In this section we provide an overview on how the CTTP Model will be incorporated and supported by the Emulation Tool and how it can be used to implement a realistic training scenario. The CTTP model will be fully supported in the final version of the Emulation Tool.

The development subset of a valid CTTP model determines the operational aspects of the pilot system, how they can be deployed, and their connections. The following figure illustrates the main elements of the development sub-model and Appendix I presents an initial version of its schema. A relevant Scalable Vector Graphics (SVG) file has been uploaded in the project SVN.

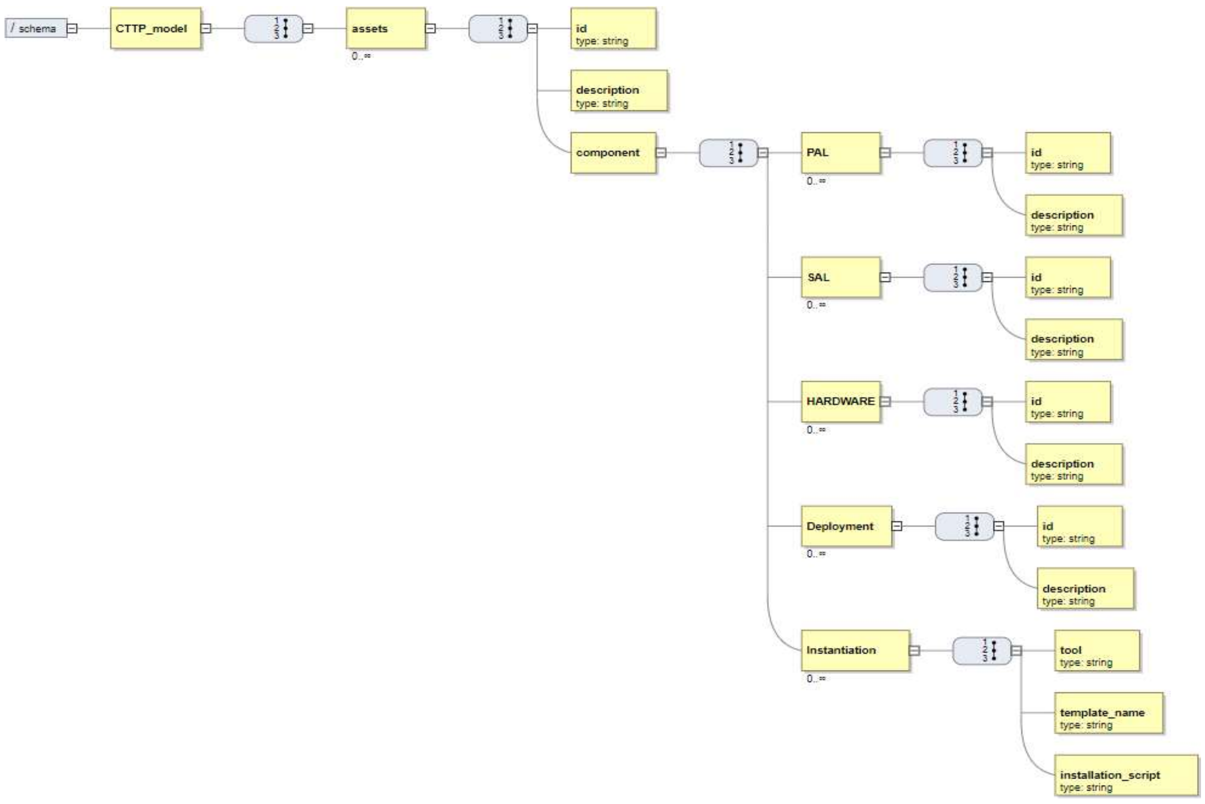


Figure 16: The development sub-model schema

Every asset is disassembled into several components that describe its actual structure. These components can model the Platform Level software (PAL) (i.e. operating system), the Software Architecture Layer (SAL) (e.g. software applications), the HARDWARE modules, and how these PAL/SAL/HARDWARE elements are connected and deployed. Each of these three component types has a *unique ID* and a *brief description*, presenting to the user the main details for the component.

The Deployment sub-model also determines the procedure of developing/instantiating the component in the related platform tools. This may include the interoperation of real equipment (administrated via the Assurance Tool) or emulation/simulation of it. The CTTP designer has to define the relevant tool and how the platform can instantiate the component.

In the Emulation Tool OpenStack infrastructure, at first, the designer has to implement the various resources that will be instantiated for the training process. This includes system images (software and OS settings), flavors (hardware characteristics), and networks (internal connection of OpenStack resources or external communications). These primitives can be built from the scratch or with the assistance of OpenStack tools. Once created, they can be utilized by the various OpenStack services by referring to their alias name. Then, the CTTP designer orchestrates the management of HEAT templates that deploy specific instances of these resources.

4.2 Emulation instantiation script

In order to instantiate an emulated component, we have to provide a valid OpenStack script to the Emulation Controller (see Section 3.1). For THREAT-ARREST, these scripts are described in the HEAT YAML files (see Section 3.2). The Deployment sub-model contains the related YAML script for every emulated component (to be instantiated by existing pre-set primitives).

At the current version, the information in the asset's "description" and the related "instantiation script" are correlated, but they are set separately and manually by the CTTP designer in order to facilitate the implementation process.

As aforementioned, in OpenStack *flavors* define the hardware characteristics (compute, memory, and storage) of a deployed computing unit, while the software aspects are determined in a VM *image* (virtual disk with a bootable OS installed on it). Thus, the main Hardware features of an emulated component are mapped in a flavor and the PAL/SAL features are mapped in an image. Then, HEAT can instantiate VMs, which implement the functionality of the emulated component, and orchestrate them.

4.3 Example – Smart shipping scenario

This section presents a complete example for modelling a simple emulated component for the Smart Shipping scenario (Use Case 3) and how to instantiate it during a training session. The interconnection of different emulated components is detailed in the related deliverable D2.3. The use case described below will be fully supported and deployed in the final version of the Emulation Tool.

4.3.1 Emulated component – The Captain's PC

The emulated component is the Captain's PC on the vessel. It facilitates the communication of the Captain with the company's headquarters. In this scenario, the user accesses the company's email via a web browser that runs on the PC and communicates with the back-office personnel (e.g. receive weather broadcasts, report problems, etc.). The Internet connection is enabled via a satellite connection (modelled as an external connection for OpenStack).

4.3.2 Scenario – Phishing email

This is a social engineering scenario which targets valuable actuators with moderate security training (e.g. (Manifavas et al., 2014; Cesena et al. 2017; Ferrera et al. 2018)). Thus, an email phishing attack is performed to a ship's captain. The attacker sends bogus emails to the captain and tries to mislead him/her in performing requested actions. The main goal is to make the captain to update his/hers account credentials via a malicious (phishing) web site (thus, stealing the actual username/password) or disclose other organization-critical information.

The scenario is implemented in the Emulation Tool. A web browser, which models the email client service, is pre-installed in the VM that instantiates the captain's PC. As the scenario is in progress, the Captain/trainee receives emails originated from the organization's back office (i.e. weather broadcast, change route requests, update account credentials, etc.). The trainee has to discriminate if the received messages are either faulty/malicious and ignore them or legitimate and perform the designated actions.

4.3.2.1 Scenario propagation

All referred actions test the captain's decision making:

- Legitimate email with the weather broadcast, denoting that the weather is good.
- Legitimate faulty email requesting route diversions due to bad weather conditions. The email contains the details of another ship and was sent to the captain/trainee by mistake.
- Legitimate email requesting to ignore the previous (faulty) email and retain the initial course.
- Malicious (phishing) email requesting to update the account's password via a malicious site, which resembles the original one. The message justifies that this urgent update is ordered due to malicious activity that is monitored by the organization's security officer the last days. The email contains the actual name of the chief security officer, which could have been mined by the attacker either by the information that is publicly available for the organization or by other social engineering attempts.

4.3.2.2 Scenario modelling

For modelling the full scenario, we acquire the following VMs. The trainee operates the VM for the Captain's PC. The legitimate messages are sent by the VM that instantiates the backend office PCs via the email server. The wile actions are performed through another VM that deploys the malicious equipment (e.g. email client, phishing web site, etc.).

The emails will be sent either automatically based on triggered events (e.g. timestamps) or manually by the trainer. Furthermore, the message types (legitimate, faulty, or malicious), as they are defined for the aforementioned scenario, can be altered between different training sessions. In this version, the VMs are communicating through Internet. In the following subsections, we describe the CTTP model for the captain's PC. The rest VMs can be deployed in a similar fusion.

4.3.3 CTTP model

The code sample in Appendix II models the development sub-model for the aforementioned asset in an XML format. The characteristics of the PC are described in the SAL, PAL, and Deployment elements.

The physical characteristics (HARDWARE1) of the PC are:

- Hardware (part of the related OpenStack flavor definition)
  - CPU: 2.4 GHz
  - Core: 8
  - Hard disk: 20GB
  - RAM: 16GB

  - Motherboard
  - Graphics card
  - Sound card
- Connectivity (part of the related OpenStack networks' definitions)
  - Ethernet
  - WiFi/802.11ac
  - USB ports
- Peripherals (not modelled for the emulation Deployment sub-model)
  - Monitor
  - Keyboard
  - Mouse
  - CD/DVD ROM
  - Speaker
  - Mike

The software characteristics (PAL1/SAL1) of the PC are (part of the related OpenStack image definition):

- PAL1
  - Operating System (OS): Windows 10
  - Architecture: 64-bit
- SAL1
  - Web browser: Internet Explorer
  - Anti-virus: Windows Defender

The Deployment feature (DEP1) that is correlated with the PC is:

- *PAL1/SAL1* runs on *HARDWARE1* and connects to the Internet through the Satellite connection *Network*

4.3.4 Instantiation script 1 – Build template from the scratch

At first, the CTTP designer has to build the underlying primitives (i.e. images, flavors, and networks) or use the default ones that are provided by the OpenStack. Then, he/she can run the instantiating HEAT template. The following piece of code describes how to deploy a pre-existing VM image of the captain's PC, as the Windows 10 iso (windows_10_pro_iso_64_bit_iso_october_2018_update_v_1809) with the flavor m1.large (8 CPU cores, 20GB hard disk, and 16GB RAM) that has a satellite Internet connection (satellite-conn).

```
<Instantiation>
  <tool>Emulation</tool>
  <template_name>NO<\template_name>
  <installation_script>
    <template_filename>captain_pc.yaml<\template_filename>
    <template_script>
    heat_template_version: 2018-08-31
        description: Simple template to an instance of the captain's PC,
  which will be built from the scratch
        resources:
            my_instance:
                type: OS::Heat::SoftwareConfig
                properties:
                    key_name: captain_pc
                    image: windows_10_pro_iso_64_bit
                    flavor: m1.xlarge
                    networks:
                     -   Network: satellite-conn
    </template_script>
   </installation_script>
</Instantiation>
```

The Emulator Controller will extract the 'template_script' and create a file captain_pc.yaml' that contains this deployment script for OpenStack, that the Emulation engine will execute on the target platform.

4.3.5 Instantiation script 2 – Deploy a pre-set template

The next piece of code deploys a pre-set HEAT template of the captain's PC (captain_pc.yaml), which has been deployed in advance in our OpenStack installation.

```
<Instantiation>
  <tool>Emulation</tool>
    <template_name>captain_pc.yaml</template_name>
  <installation_script>NO</installation_script>
</Instantiation>
```

The Emulator Controller will extract the 'template_name' and the emulation engine will execute it on the target platform.

5 Conclusion

In this deliverable we provided an overview of the first round of activities of task T2.1 of the "WP2 – Emulation Tool".

The work has led to the development of a working prototype that manages the compiling and execution of XML input models, describing the training scenario, in working HEAT templates, and the execution of these templates in the OpenStack environment.

The technologies, and in particular the OpenStack framework, have been selected basing on the conclusion of the "D1.3 – THREAT-ARREST platform's initial reference architecture" and "D1.2 – The platform system requirements analysis report". The adoption of a modular and extensible framework as OpenStack has allowed the implementation of a system able to support all the emulated and simulated components.

The second round of the task activities, ending at M24, will be focused on the releasing of a complete system strictly connected with the other THREAT-ARREST tools and, in particular, with the full support of the project-specific CTTP Models.

References

[1] Cesena, M., et al. 2017. SHIELD Technology Demonstrators. CRC Press, Book for Measurable and Composable Security, Privacy, and Dependability for Cyberphysical Systems, pp. 381-434.

[2] Davis, J., & Magrath, S. (2013). *A Survey of Cyber Ranges and Testbeds.* Cyber and Electronic Warfare Division - Australian Government. Retrieved from https://apps.dtic.mil/docs/citations/ADA594524

[3] Ferrera, E., et al. 2018. IoT European Security and Privacy Projects: Integration, Architectures and Interoperability. CRIStin – SINTEF, Next Generation Internet of Things. Distributed Intelligence at the Edge and Human Machine-to-Machine Cooperation. Book Chapter 7, pp. 207-292.

[4] Manifavas, C., et al., 2014. DSAPE – Dynamic Security Awareness Program Evaluation. Human Aspects of Information Security, Privacy and Trust (HCI International 2014), 22-27 June, 2014, Creta Maris, Heraklion, Crete, Greece, Springer, LNCS, vol. 8533, pp. 258-269.

[5] OpenStack. (2019). *OpenStack Open Infrastructure*. Retrieved 2019, from https://www.openstack.org/

[6] OpenStack. (2019b). *Heat - OpenStack*. Retrieved 2019, from https://wiki.openstack.org/wiki/Heat

[7] OpenStack. (2019c). *OpenStack docs: DevStack*. Retrieved 2019, from https://docs.openstack.org/devstack/latest/

[8] OpenStack. (2019d). *OpenStack Resource Types*. Retrieved 2019, from https://docs.openstack.org/heat/stein/template_guide/openstack.html

[9] OpenStack. (2019e). *Openstack Software*. Retrieved 2019, from https://www.openstack.org/software/project-navigator/openstack-components

[10] OpenStack. (2019f). *Puppet OpenStack Guide*. Retrieved from https://docs.openstack.org/puppet-openstack-guide/latest/

[11] OpenStack. (2019g). *OpenStack-Ansible Deployment Guide*. Retrieved from https://docs.openstack.org/project-deploy-guide/openstack-ansible/latest/

[12] OpenStack4j. (2019). *Fluent OpenStack SDK for Java*. Retrieved from http://www.openstack4j.com/

[13] Pridmore, L., Lardieri, P., & Hollister, R. (2010). National Cyber Range (NCR) automated test tools: Implications and application to network-centric support tools. *Proc. of 2010 IEEE AUTOTESTCON*, (pp. 1-4). Orlando, FL, USA. doi:https://doi.org/10.1109/AUTEST.2010.5613581

[14] SmartBear. (2019). *Swagger tool for API development*. Retrieved 2019, from https://swagger.io/

[15] YAML. (2019). *The Official YAML website*. Retrieved 2019, from https://yaml.org/

Appendix I – CTTP Development sub-model Schema

This appendix presents an initial schema version of the CTTP development sub-model. The schema is an XML Schema Definition (XSD) file and is also included in the SVN.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:hfp="http://www.w3.org/2001/XMLSchema-hasFacetAndProperty">
 <xs:element minOccurs="0" maxOccurs="unbounded" abstract="true" name="CTTP_model">
 <xs:complexType>
  <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="assets">
         <xs:complexType>
          <xs:sequence>
             <xs:element name="id" type="xs:string" use="required"/>
             <xs:element name="description" type="xs:string" use="required"/>
             <xs:element minOccurs="1" maxOccurs="1" name="component">
                 <xs:complexType>
                  <xs:sequence>
                     <xs:element minOccurs="0" maxOccurs="unbounded " name="PAL">
                      <xs:complexType>
                         <xs:sequence>
                             <xs:element name="id" type="xs:string" use="required"/>
                             <xs:element name="description" type="xs:string" use="required"/>
                         </xs:sequence>
                      </xs:complexType>
                     </xs:element>
                     <xs:element minOccurs="0" maxOccurs="unbounded " name="SAL">
                      <xs:complexType>
                         <xs:sequence>
                             <xs:element name="id" type="xs:string" use="required"/>
                             <xs:element name="description" type="xs:string" use="required"/>
                         </xs:sequence>
                      </xs:complexType>
                     </xs:element>
                     <xs:element minOccurs="0" maxOccurs="unbounded " name="HARDWARE">
                      <xs:complexType>
                         <xs:sequence>
                             <xs:element name="id" type="xs:string" use="required"/>
                             <xs:element name="description" type="xs:string" use="required"/>
                         </xs:sequence>
                      </xs:complexType>
                     </xs:element>
                     <xs:element minOccurs="0" maxOccurs="unbounded " name="Deployment">
                      <xs:complexType>
                         <xs:sequence>
                             <xs:element name="id" type="xs:string" use="required"/>
                             <xs:element name="description" type="xs:string" use="required"/>
                         </xs:sequence>
                      </xs:complexType>
                     </xs:element>
                      <xs:element minOccurs="0" maxOccurs="unbounded " name="Instantiation">
                             <xs:complexType>
                                 <xs:sequence>
                                      <xs:element name="tool" type="xs:string" use="required"/>
                                      <xs:element name="template_name" type="xs:string" use="required"/>
                                      <xs:element name="installation_script" type="xs:string" use="required"/>
                                 </xs:sequence>
                             </xs:complexType>
```

```
                        </xs:element>
                     </xs:sequence>
                   </xs:complexType>
               </xs:element>
           </xs:sequence>
        </xs:complexType>
        </xs:element>
    </xs:sequence> </xs:complexType> </xs:element> </xs:schema>
```

Appendix II – CTTP Development sub-model for the Captain's PC

This appendix presents a first version of the CTTP development sub-model for instantiating the Captain's PC (see Section 5.3). The scripts for the two instantiation types (build form the scratch or deploy a pre-set image) are detailed in Sections 5.3.3 and 5.3.4 respectively. The complete file has be uploaded in the SVN.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--CTTP Model for the Captain's PC -->
<CTTP_model xmlns="http://www.w3schools.com"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3schools.com file:///C:/CTTP_Model.xsd">
   <name>Smart_Shipping_CTTP</name>
   <assets>
      <id>ASSET1</id>
      <description>PCs/devices that facilitate the communication with the shiiping company backend (i.e.
the captain's PC). The PC will emulate the typical software that is required for the captain (i.e. an email
service)</description>
      <component>
         <PAL>
            <id>PAL1</id>
            <description>The platform software for the captain's PC: Windows 10 OS</description>
         </PAL>
         <SAL>
            <id>SAL1</id>
            <description>The application software for the captain's PC: Mozilla Firefox web browser, Avira
anti-virus</description>
         </SAL>
         <HARDWARE>
            <id>HARDWARE1</id>
            <description>Captain's PC: Hardware (CPU 2.4 GHz, 50GB hard disk, 16GB RAM, motherboard,
graphics card, sound card), Connectivity (Ethernet, WiFi/802.11ac, USB ports), Peripherals (monitor,
keyboard, mouse, CD/DVD ROM, speaker, mike)  </description>
         </ HARDWARE >
         <Deployment>
            <id>DEP1</id>
            <description>The satellite connection which enables the connection of the on-ship system with the
Internet</description>
         </Deployment>
         <!--One of the two 'Instantiation' choices will be made by the CTTP designer-->
         <Instantiation>
            <tool>Emulation</tool>
            <template_name>NO</template_name>
            <installation_script>-- HEAT YAML Script / Subsection 5.3.3 --</installation_script>
         </Instantiation>
         <Instantiation>
            <tool>Emulation</tool>
            <template_name>-- HEAT YAML Script / Subsection 5.3.34 --</template_name>
            <installation_script>NO</installation_script>
         </Instantiation>
      </component>
   </assets>
</CTTP_model>
```