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

D3.1: CTTP Models and Programmes Specification Language †

Abstract: This deliverable is the main output of task "T3.1 – CTTP Language definition and Tool Support" and details the design and definition of the Cyber Threat and Training Preparation (CTTP) model and programme specification language. As such, the deliverable presents the Core Assurance and CTTP training models, and their sub-models, such as the physical and software sub-models, as well as the simulation, emulation sub-models. Moreover, the language constructs derived based on said models are defined, while an example is provided on the use of these models to define training programmes with different levels of complexity.

Contractual Date of Delivery	31/08/2019
Actual Date of Delivery	31/08/2019
Deliverable Security Class	Public
Editor	*Konstantinos Fysarakis, Michail Smyrlis (STS)*
Contributors	*Fulvio Frati (UMIL),* *Jiri Prusa (CZ.NIC)*
Quality Assurance	*George Hatzivasilis (FORTH),* *George Bravos (ITML)*

† *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. *George Hatzivasilis (FORTH)*
2. *George Bravos (ITML)*

Revisions

Version	Date	By	Overview
2.0	28/08/2019	Smyrlis Michail (STS)	Final Deliverable
1.2	19/08/2018	Smyrlis Michail (STS)	Addressed comments from reviewers
1.0	01/08/2019	Konstantinos Fysarakis (STS)	Version ready for internal review
0.7	31/07/2019	Konstantinos Fysarakis (STS)	Revised examples, models' refinement, text edits
0.45	29/07/2019	Konstantinos Fysarakis (STS)	Updated core model descriptions; CTTP model update
0.4	25/07/2019	Konstantinos Fysarakis (STS)	Revised core and CTTP model, example update
0.3	08/04/2019	Konstantinos Fysarakis (STS)	First draft of THREAT-ARREST model, incl. examples
0.2	29/03/2019	Konstantinos Fysarakis (STS)	Requirements update
0.1	15/12/2018	Konstantinos Fysarakis (STS)	First Draft

Executive Summary

This deliverable presents the CTTP language definition that will be followed for the CTTP models and programmes specification language and is developed under task “T3.1 – CTTP Language definition and Tool Support”. The proposed language relies on multiple UML figures that provide representational guidelines for the definition of the core assurance model and the CTTP sub-model. Lastly, some examples of simple and realistic simple programmes are proposed.

Table of Contents

List of Abbreviations

CPS Cyber Physical System

CTTP Cyber Threat and Training Preparation

CVE Common Vulnerabilities and Exposures

EHR Electronic Health Records

GDPR General Data Protection Regulation

IoT Internet of Things

NIST National Institute of Standards and Technology

NVD National Vulnerability Database

OCL Object Constraint Language

PAL Platform Level software

SAL Software Architecture Layer

UML Unified Modelling Language

VM Virtual Machine

List of Tables

List of Figures

1 Introduction

This deliverable is the main output of task "T3.1 – CTTP Language definition and Tool Support". As such, it details the design and definition of the specification language which will be used to define the Cyber Threat and Training Preparation (CTTP) training programmes and their associated CTTP models/sub-models. The language definition is driven by the requirements for said language, i.e. the key functions that the language will need to be able to specify in order to comprehensively define the Core Assurance models, the Platform Level software (PAL), Software Architecture Layer (SAL), and Physical sub-models, as well as the simulation, emulation and training one. This process is documented within this deliverable, along with the language defined including key constructs of the language and associated examples of their usage.

The associated language editor will be provided in the deliverable "D3.2 – CTTP Models and Programmes Specification Tool", while the development of the CTTP models via the language defined herein will take place in the context of task "T3.2 – CTTP Models and Programmes Development" and will be documented in the deliverables "D3.3 – Reference CTTP Models and Programmes Specifications v1" and "D3.5 – Reference CTTP Models and Programmes Specifications v2".

The responsible person utilizes the CTTP editor to create the Security Assurance Model for the target organisation. This model includes the assets of the organisation, security properties for these assets, threats that may violate these properties and the security controls that protect the assets.

Subsequently, the trainer uses the CTTP tool to choose specific assets, threats, properties and controls; based on their choice, tailored training programmes with varied difficulties will be generated. For example, the trainer can choose Low-privilege-end-users (assets) and phishing (threat), to generate phishing scenarios for low-privilege end users.

The document is organised as follows: Section 2 lists that requirements that must be considered in the language definition process; Section 3 details the core assurance model and the CTTP programme models defined that form the core of the THREAT-ARREST model-driven process; Section 4 presents the language constructs derived based on said models; Section 5 includes examples of the use of the primitives defined within the deliverable to deploy CTTP programmes, and; Section 6 concludes the deliverable.

Lastly, Appendix I presents the full view of the CTTP sub-model, Appendix II presents a simple phishing training scenario model instance, and Appendix III presents a realistic phishing training scenario model instance.

2 Language Requirements

The THREAT-ARREST platform will deliver security training, based on a model-driven approach with core assurance and training preparation (CTTP) models. To achieve this, said models must specify the system assets, the potential attacks, the security controls of cyber systems against them, and the tools that may be used to assess the effectiveness of these controls, while also driving the training process, and aligning it with operational cyber system security assurance mechanisms to ensure the relevance of training.

Some key considerations for the different sub-models that will comprise the THREAT-ARREST core assurance and CTTP training models are detailed in the subsections below.

2.1 Core Assurance Model Requirements

For the security assurance and certification, the core assurance model must define, via the language, the system's **assets** and a variety of properties related to them. **Security threats, risks** and **vulnerabilities** that may affect the physical or software components of the system; **assumptions,** regarding the external environment of the cyber system and the behaviour of agents; **security controls,** that are used to mitigate the risks arising from the threats; **assessment measures**, that determine how to identify attacks coming from the threats and assess the effectiveness of the security controls; **assessment tools,** that should be used to enable these measures prior the deployment of the system (e.g., **static analysis** and **testing tools**) or during the operation of the system (**monitoring** and **dynamic testing tools**); **parameters,** determining how the attacks may manifest themselves, how the security controls may respond to them (e.g., the attack manifestation events captured and detection time, the undertaken response actions) and the outputs that the deployed assessment tools will generate for the situation.

2.2 Hardware, Software and Physical sub-model requirements

The language should be able to define Hardware components (e.g., smart home devices, user's equipment), Software components (e.g., Hub/MQTT Broker, ActiveMQ, web interface) and Physical components (e.g., power, heat, air, ventilation and the physical network assets).

Hardware should include the hardware features of a device (e.g., CPU, bios information). A software component type can either be a SAL or a PAL and can include software services and/or software components. SALs are application layer software modules that should include key information about the software such as, software type, vendor, name, version and short description of its functionality; specifics about its APIs (e.g., how to access, URI, etc.); what data does it process, how does it process it and what measures are in place to protect it (i.e., encryption) in its different states (e.g., in-transit, at-rest); finally, different categories the software can belong to (e.g., confidentiality, integrity).

PALs are the platform level software such as the JVM, a VM or software like OpenStack. It should include the same information as SALs. The Physical components are the parts of a building such as the power, heat, air, ventilation conditioning (HVAC) and the physical network assets (networking cables etc.).

All the above are part of the Asset superclass. An asset can be contained in 1 other asset and can be managed by 0 or more Assets. A containment can illustrate a deployment relation if an asset is contained in another asset and it operates within the containment. For instance, a software asset can be contained in a hardware asset, a Software (SAL) operates in a virtual machine (PAL) while the virtual machine operates within the Operating System. Also, the OS controls the virtual machine, and the virtual machine is being controlled by the SAL.

2.3 Simulation specification requirements

Considering the simulation in the scope of THREAT-ARREST, the language must be able to describe the simulated components of a training scenario, their parameters/attributes, as well as general parameters to influence the execution phase of simulation runs.

Regarding the components, the model has to be able to define which parts of the system will be represented by simulated components. Various parts of the cyber system can be represented as simulated components; this includes hardware components (like Internet of Things – IoT devices, network hardware), software components (like IDS, E-Mail-Server, etc.) as well as humans involved in a training scenario (such as a

simulated attacker trying to perform some pre-defined actions in the cyber system). The simulation sub-model has to be able to link these simulation components to their counterparts in other parts of the overall model such as the SAL/PAL sub-models.

The model has to be able to express a component hierarchy, i.e., certain parts of the system will consist of other components. Components and their actions can be parameterized using attributes. Each simulated component has a pre-defined set of attributes to influence its behaviour. The model has to be able to define such attributes when required. As a special kind of attribute, it has to be possible to specify links to other components both within the simulation and emulation sub-models.

Components' behaviour and the initialization actions to be performed when the simulation starts executing will be primarily specified by mechanisms outside of the CTTP model (such as Java source code). The model might contain this source code to document a component's inner workings and the precise effects of certain attribute settings.

The language must describe the instantiation process of the simulated parts and be able to parameterize attributes such as initial simulation time required for successful simulation execution. Finally, the execution of the simulation has been linked to certain phases of a training program.

2.4 Emulation specification requirements

Regarding emulation, the language should be able to describe specific aspects needed for the realization of the emulation capabilities of the training scenarios in THREAT-ARREST.

During the emulation phase, the CTTP model of a training scenario is translated and deployed into a running environment where the training can be performed. The training scenario details/features needed for the emulation are the standard key elements of a typical network configuration, i.e. nodes, networking devices and connections among them, with the addition of some further information on the software running on the VMs and the possible vulnerabilities. Therefore, the emulation sub-model is a fragment of the overall CTTP model with respect of SAL and PAL components.

Although the OpenStack platform will be used to emulate a CTTP scenario, the language of the emulation sub-model is kept platform-independent by making a suitable abstraction of the OpenStack specific concepts. Platform-dependent features will be added during the compilation phase of a CTTP model in XML format into a Heat template (see further details in D2.4).

In particular, also the emulation model can be seen as an asset composition. For the emulator, an asset acts as a generalized superclass of the elements of a network configuration: *nodes* and *networking devices*.

A node can be a *virtual machine* (VM) and any *external device*, e.g., a web cam or a sensor which needs to be modeled separately as it presents features that cannot be captured by a VM. Virtual machines can either be *customized* VMs or *special types* of VMs that will be lunched from a specific OS image with no further configuration details (e.g., VMs used for simulation, firewalls, etc.).

For each customized VM it is possible to specify both *hardware and software properties*. Software properties can be of three different types according to the aspects of software considered: *operating system* (system software coordinating and controlling all the processing activities of the computer), *programming languages* and *applications*. The latter two types of software categories are needed to specify all possible software installed on hosts (not only OS), such as:

- Programming/scripting languages (PHP, JavaScript, etc.)
- Applications performing tasks for specific purposes (e.g., browser, email client, etc.)
- Server applications (Web servers -e.g., Apache, Tomcat-, DBMS -e.g., MySQL-, DNS, etc.)
- Packets and libraries
- Tools for the training teams depending on the exercise/objective: Wireshark, Web proxies, other tools from Kali Linux, etc.

The hardware properties represent the main physical characteristics of a computer, i.e. the system unit containing CPU, hard disk, and RAM.

The basic types of networking devices required for communication and interaction among nodes within the network are *switches* and *routers*, which must have a number of attributes such as *MAC addresses* and *NICs*, *ports* and *IP addresses* (either *static* or *dynamic* via DHCP). Then, the CTTP model of the scenario must be able to express the network configuration as standard in terms of *connections between nodes and network devices*, by means of the specification of *net and subnets* to which the different hosts belong to.

For each asset, it must be possible to specify the *vulnerability*, at hardware and software level, in terms of:

- Software bug
- Vulnerable application (e.g., with no sanitization of user's input or using a vulnerable protocol -http instead of https)
- Misconfigured application
- Weak authentication mechanism (guessable passwords or cryptographic keys)

2.5 CTTP Training Programme specification requirements

For all three pilots the language should be able to define the following aspects so that the CTTP training programme adequately fulfils the training requirements of each pilot: The types of trainees (e.g., Software engineers, security experts, system administrators, end users, security auditors, chief information-officers chief security officers, public and private system users.) The different types of actions (e.g., preparedness, detection and analysis, security incident response and post security incident response) and pilot specific actions (if applicable). Pilot specific threats (e.g., Botnet type of attacks), corresponding security controls (e.g., Network segmentation), attack vectors (e.g., IoT/Bluetooth enabled devices, physical interaction with IT assets and/or Hospital equipment), and simulated/emulated components (e.g., Smart Grid, EHR, Radar). Moreover, the language should be able to define the difficulty and its levels that relate to all type of users that the system aims to train. Specifically, the levels that we foresee for the training starting from the least complex are, basic training aiming at low-privilege end users; moderate training aiming at higher privilege end users and system administrator; advanced training aiming at security engineers. Lastly, for all three pilots the CTTP model should include Legal frameworks that the training programme is aligned with (e.g., General Data Protection Regulation – GDPR).

Besides the common requirements, specific training requirements (see D1.1) deriving from each pilot will also be considered and represented in the CTTP model:

- For the **energy domain**, the CTTP training programme should be able to define relevant specifics for Cyber Physical System organizations (e.g., types of devices they are using); goals for its trainees resolving around IoT-based solutions and the Cyber Physical System (CPS) infrastructure (e.g., IoT security lifecycle (Fysarakis et al., 2014)); training with regards to achieving IoT Privacy Impact Assessment and Safety Impact Assessment.
- For the **healthcare domain**, the CTTP training programme should be able to provide training about the established processes Identifying, Protecting, Detecting, Responding and Recovering (including users that need to be trained for each of these processes) to the corresponding trainees (e.g. (Hatzivasilis et al., 2019a; Hatzivasilis et al., 2019b)).
- For the **shipping domain**, the CTTP training programme should consider various risk factors and awareness aspects (e.g., risks related to installing and maintaining software on company hardware using infected hardware or software) that all its trainees should be prepared for. Moreover, the appropriate personnel should be trained to identify/follow specific indications that suggest a system has been compromised (e.g., an unresponsive or slow to respond system (Ferrera et al., 2018)).

CTTP Training Model Requirements	
Trainee Roles	The language should be able to support different roles of trainees, such as software engineers, security experts, system administrators, end users, security auditors, chief information-officers chief security officers, public and private system users

Actions	The language should be able to support various types of actions, such as preparedness, detection and analysis, security incident response and post security incident response and pilot specific actions (if applicable)
Difficulty Levels	The language should be able to support different levels of difficulty, such as basic training aiming at low-privilege end users; moderate training aiming at higher privilege end users and system administrator; advanced training aiming at security engineers
Pilot specific threats	The language should be able to support pilot specific threats, such as botnet type of attacks – Mirai
Security controls	The language should be able to support security controls, such as Intrusion detection systems, network segmentation
Attack Vectors	The language should be able to support various attack vectors, such as IoT/Bluetooth enabled devices, physical interaction with IT assets and/or Hospital equipment
Simulated/Emulated components	The language should be able to support different simulated and emulated components, such as Smart Grid, EHR, Radar
Legal Framework	The language should be able to support and align with a variety of legal frameworks, such as General Data Protection Regulation – GDPR
Pilot Specific Requirements	
Energy domain	The CTTP training programme should be able to define relevant specifics for Cyber Physical System organizations (e.g., types of devices they are using); goals for its trainees resolving around IoT-based solutions and the Cyber Physical System (CPS) infrastructure (e.g., IoT security lifecycle); training with regards to achieving IoT Privacy Impact Assessment and Safety Impact Assessment.
Healthcare domain	The CTTP training programme should be able to provide training about the established processes Identifying, Protecting, Detecting, Responding and Recovering (incl. users that need to be trained for each of these processes) to the corresponding trainees.
Shipping domain	The CTTP training programme should consider various risk factors and awareness aspects (e.g., risks related to installing and maintaining software on company hardware using infected hardware or software) that all its trainees should be prepared for. Moreover, the appropriate personnel should be trained to identify/follow specific indications that suggest a system has been compromised (e.g., an unresponsive or slow to respond system).

Table 1 CTTP training model requirements

3 The THREAT-ARREST system and training models

This chapter presents the models that will be leveraged in THREAT-ARREST to model both the actual cyber-system and its assurance properties as well as to specify the associated training programmes. Thus, this section presents both the Core Assurance Model, as well as the CTTP sub-model extension which is dedicated to the generation of training programmes.

A graphical view of the models is provided in the form of a Unified Modelling Language (UML) Class diagram, visualising the basic modelling constructs of the language and their relations. Different views of the model are presented to highlight specific aspects and avoid presenting the full view which would be hard for the reader to follow. Moreover, a description of the elements and attributes comprising each view is provided.

To assist the reader, it should be noted that throughout the document, we make use of the following multiplicities:

- 0...*: Zero(participation) or many(cardinality)
- 1...*: One or many
- 0..1: Zero or One
- 1: Only one (Obligatory attribute)

A multiplicity is the combination of the cardinality and the participation. The cardinality denotes the maximum number of possible relationship occurrences in which a certain entity can participate in while the participation denotes if all or only some entity occurrences participate in a relationship.

Lastly, the included figures that contain the UML of both the core assurance model and the CTTP sub-model were created through the MagicDraw modelling tool (MagicDraw, 2019).

3.1 The Core Assurance Model

The Core Assurance Model is defined to specify known threats that may affect the physical or software components of the system; assumptions regarding the external environment of the cyber system and the behaviour of agents (human- or system-agents) related to it that can affect it (i.e., prevent or enable threats); and security controls used to mitigate the risks arising from the threats.

The assurance model also specifies assessment measures, determining how to detect attacks arising from the threats and assess the effectiveness of the security controls. It also specifies the assessment tools that should be used to realize these measures prior to the deployment of the system (e.g., static analysis and testing tools) or during the operation of the system (monitoring and dynamic testing tools).

Moreover, it specifies parameters determining how the attacks may manifest themselves, how the security controls may respond to them (e.g., the attack manifestation events captured and detection time, the undertaken response actions) and the outputs that the deployed assessment tools will generate for the situation.

3.1.1 Security Assurance Model Element

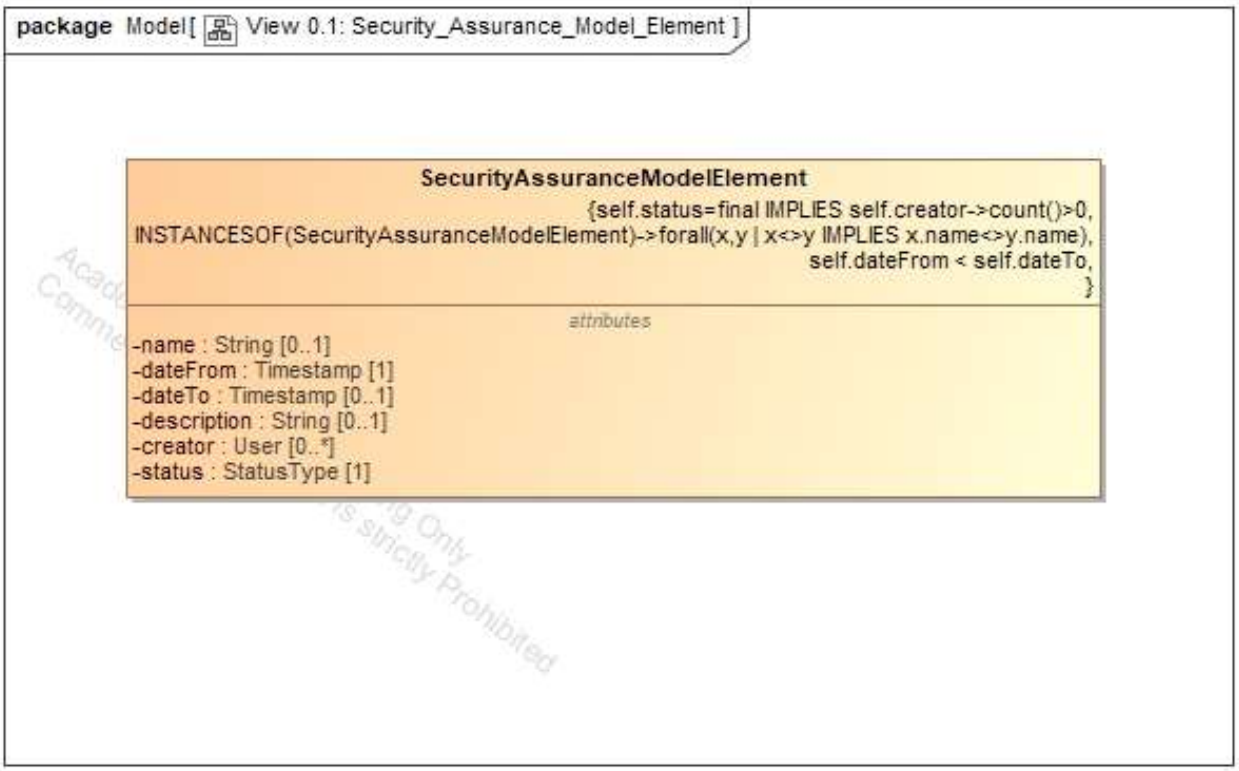

Figure 1 Security Assurance Model Element view

SecurityAssuranceModelElement: The core element of the security assurance language. It is inherited by every other class within the language (see Figure 1).

Attributes:

1. **Name:** The name of the element
2. **dateFrom:** The timestamp of the insertion of the element in the assurance platform
3. **dateTo:** The timestamp of the deletion of the element from the assurance platform.
4. **Description:** A brief description of the element that states its purpose.
5. **Creator:** The **Person** that inserted this element in the platform.
6. **Status:** The current status of the element. The status can either be **final** or **draft.** An element is considered as **Final,** if and only if the Object Constraint Language (OCL) constraints are met.

OCL Constraint:

1. If the status field of the Security assurance model element is set to final, then at least one creator must be defined.
2. Each instance of a security assurance model element must have different name.
3. The dateFrom attribute must always be less than the dateTo attribute.

3.1.2 Threats, Assets & Vulnerabilities sub-model

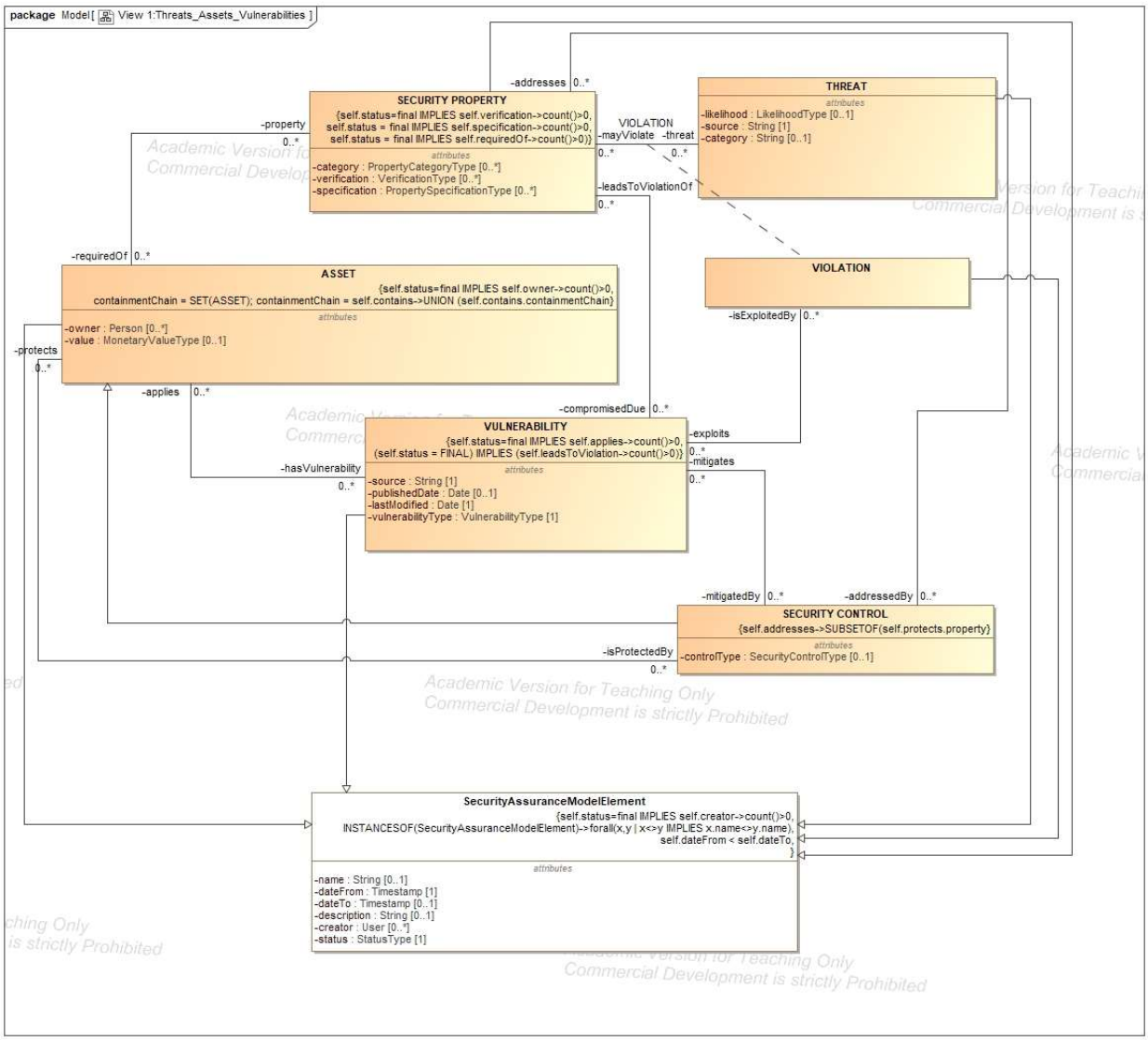

Figure 2 Threat, Asset & Vulnerability relationships view

Figure 2 includes the following key elements:

Vulnerability: A security vulnerability is a weakness an adversary could take advantage of to compromise the confidentiality, availability, or integrity of a resource (e.g., a buffer overflow vulnerability to a specific version of an SQL server software). Based on the computational asset's attributes we intend to extract the related vulnerabilities from the NVD (Nvd.nist.gov, 2019) or other repositories, and perform a risk analysis to identify the most critical threats for the examined organization.

Attributes:

- **Source:** The location a specific vulnerability was fetched from (e.g. National Vulnerability Database (NVD) (Nvd.nist.gov, 2019)
- **publishedDate:** The exploitation date of the vulnerability
- **lastModified (optional):** The date the vulnerability was modified from the source organization.
- **vulnerabilityType:** The type of the vulnerability. This attribute can either be **computational (**i.e. the vulnerability can be exploited from a computational asset, like an SQL Injection) or **physical** (i.e. the vulnerability can be exploited from a physical asset, such as a misplaced physical asset).

Associations:

- **Vulnerability** mitigatedBy 0…* **Security Control(s)**
- **Vulnerability** isExploitedBy 0…* **Violation(s)**
- **Vulnerability** leadsToViolationOf 0…* **Security Property(ies)**
- **Vulnerability** appliesTo 0…* **Asset(s)**

Inheritance:

- **Vulnerability** inherits the **SecurityAssuranceModelElement**

THREAT: Any circumstance or event with the potential to adversely impact an asset through unauthorized access, destruction, disclosure, modification of data, and/or denial of service (e.g., Information leakage/sharing due to human error).

Attributes:

- **Likelihood:** The likelihood of a threat to violate an Asset.
- **Source:** The location a specific threat was fetched from (e.g. ENISA's Threat Taxonomy (Enisa.europa.eu, 2019))
- **Category:** The category of a threat (e.g. Physical, Resources etc.)

Associations:

- **Threat** mayViolate 0…* **Security Property(ies)**

Inheritance:

- **Threat** inherits the **SecurityAssuranceModelElement.**

3.1.3 Assets sub-model

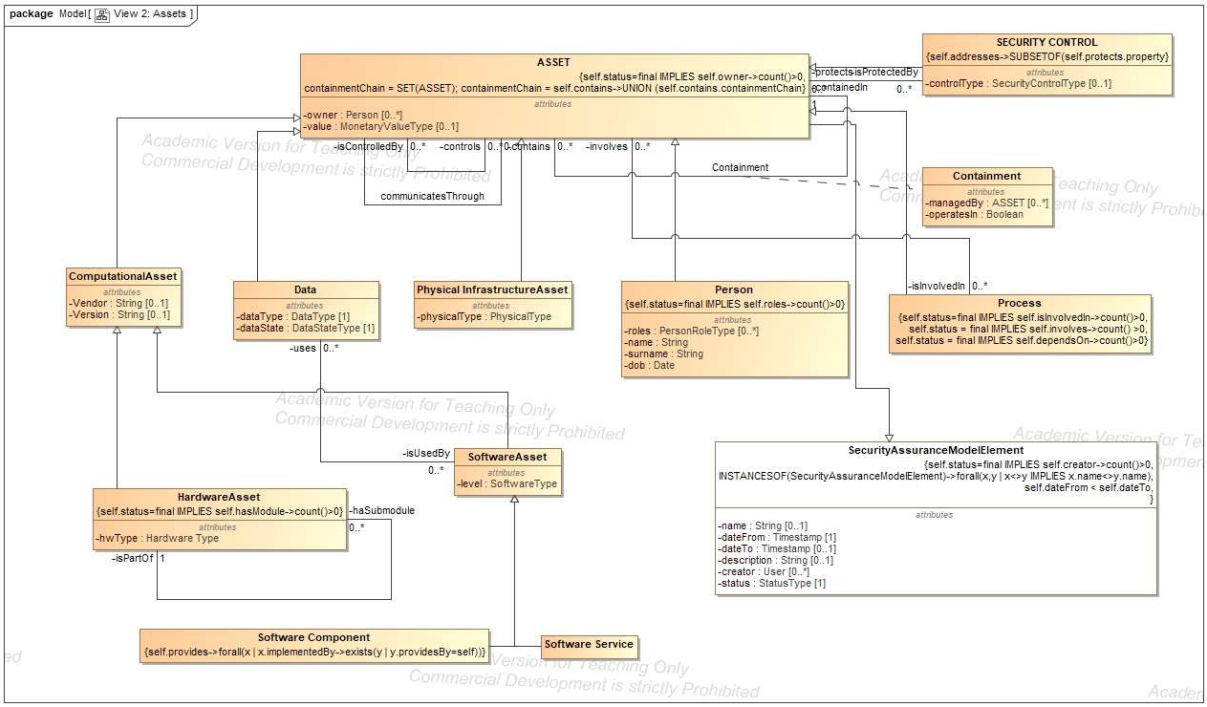

Figure 3 Asset sub-model view

ASSET: Describes the different assets of a cyber-system (see Figure 3).

Attributes:

- **Owner:** The **Person** that owns this asset.
- **Value:** The monetary value of the asset.

Associations:

- **Asset** protectedBy 0…* **Security Control(s)**
- **Asset** contained 0…* **Asset(s)**
- **Asset** isContainedIn 0…* **Asset(s)**
- **Asset** isInvolvedIn 0…* **Process(es)**
- **Asset** controls 0…* **Asset(s)**
- **Asset** isControledBy 0…* **Asset(s)**
- **Asset** communicatesThrough 0…* **Asset(s)**

Inheritance:

- **Asset** inherits the **SecurityAssuranceModelElement.**

Containment. A containment relation between 0 or more asset(s). For instance, a software asset can be contained in a hardware asset, a Software (SAL) operates in a virtual machine (PAL) while the virtual machine operates within the Operating System. Also, the OS controls the virtual machine, and the virtual machine is being controlled by the SAL. A containment can illustrate a deployment relation if an asset is contained in another asset and it operates within the containment (i.e., **operatesIn** equals true).

Attributes:

- **managedBy:** The number of Asset(s) that manages the contained Asset.
- **operatesIn:** Provides information regarding the operational behavior of a containment.

Computational Asset: A subclass of assets, used to defined vendor (e.g. MySQL 3.20) and identify the relevant vulnerabilities from vulnerabilities' databases (e.g. Common Vulnerabilities and Exposures – CVE)

Attributes:

- **vendor:** The vendor of the asset
- **version:** The version of the asset.

Inheritance:

- **ComputationalAsset** inherits the **Asset** class (and every other class inherited by the superclass).

Physical Infrastructure Asset: The physical components of a building belonging to the organisation (e.g., server room)

Attributes:

- **physicalType:** The type of a physical asset (e.g., the power, heat, air, ventilation conditioning (HVAC) units and the physical network assets (networking cables such as RJ-45 etc.)

Inheritance:

- **PhysicalAsset** inherits the **Asset** class (and every other class inherited by the superclass).

3.1.3.1 Software Asset sub-model

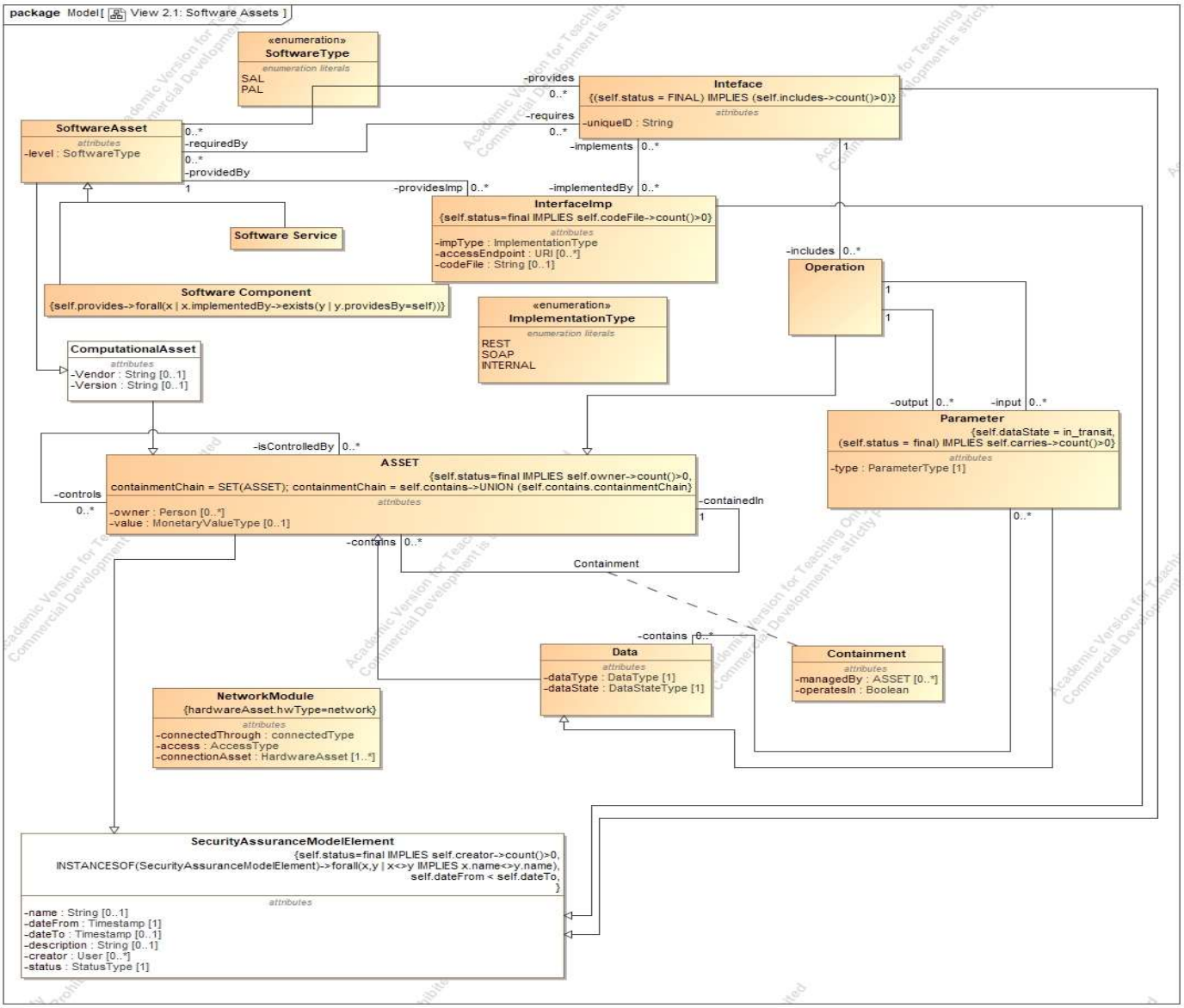

Figure 4 Software Asset sub-model view

Figure 4 contains the following key elements:

Software Asset: Software assets are a set of programs used to operate computers and execute specific tasks. It can either be a **Software Service** or a **Software Component.**

Attributes:

- **Level:** The level of the software. It can either be a SAL or a PAL. A software can either be a SAL or a PAL.
  - **SAL**: An application layer software module (the source code of a software implemented within the organization etc.)
  - **PAL** (Platform Level Software): An abstract software platform (i.e., a connector, the JVM, a web server, OpenStack (OpenStack, 2019) etc.).

Associations:

- **SoftwareAsset** provides 0…* **Interface(s)**
- **SoftwareAsset** requires 0…* **Interface(s)**
- **SoftwareAsset** providesImplTo 0…* **InterfaceImplementation(s)**

Inheritance:

- **SoftwareAsset** inherits the **ComputationalAsset** class (and every other class inherited by the superclass).

Interface: The various interfaces of a Software Asset (e.g., an interface that retrieves all the data from a local database).

Associations:

- **Interface** isProvidedBy 0…* **SoftwareAsset(s).**
- **Interface** isRequiredBy 0…* **SoftwareAsset(s).**
- **Interface** isImplementedBy 0…* **InterfaceImpl(s).**

Inheritance:

- **Interface** inherits the **SecurityAssuranceElementModel** class.

InterfaceImp: The implementation of an interface (e.g., the REST implementation of the retrieveAll interface mentioned above).

Attributes:

- **impType:** The implementation type. An interface can be implemented as: (a) REST, (b) SOAP or (c) internal (i.e., it is not exposed over the web).
- **accessEndPoint (optional):** If exposed over the web, the interface must provide a uri.
- **codeFile:** The source file of the interface.

Association:

- **InterfaceImp** implements 0…* **Interface(s)**

Inheritance:

- **InterfaceImp** inhetits the **SecurityAssuranceModelElement** class.

3.1.3.2 Hardware Asset sub-model

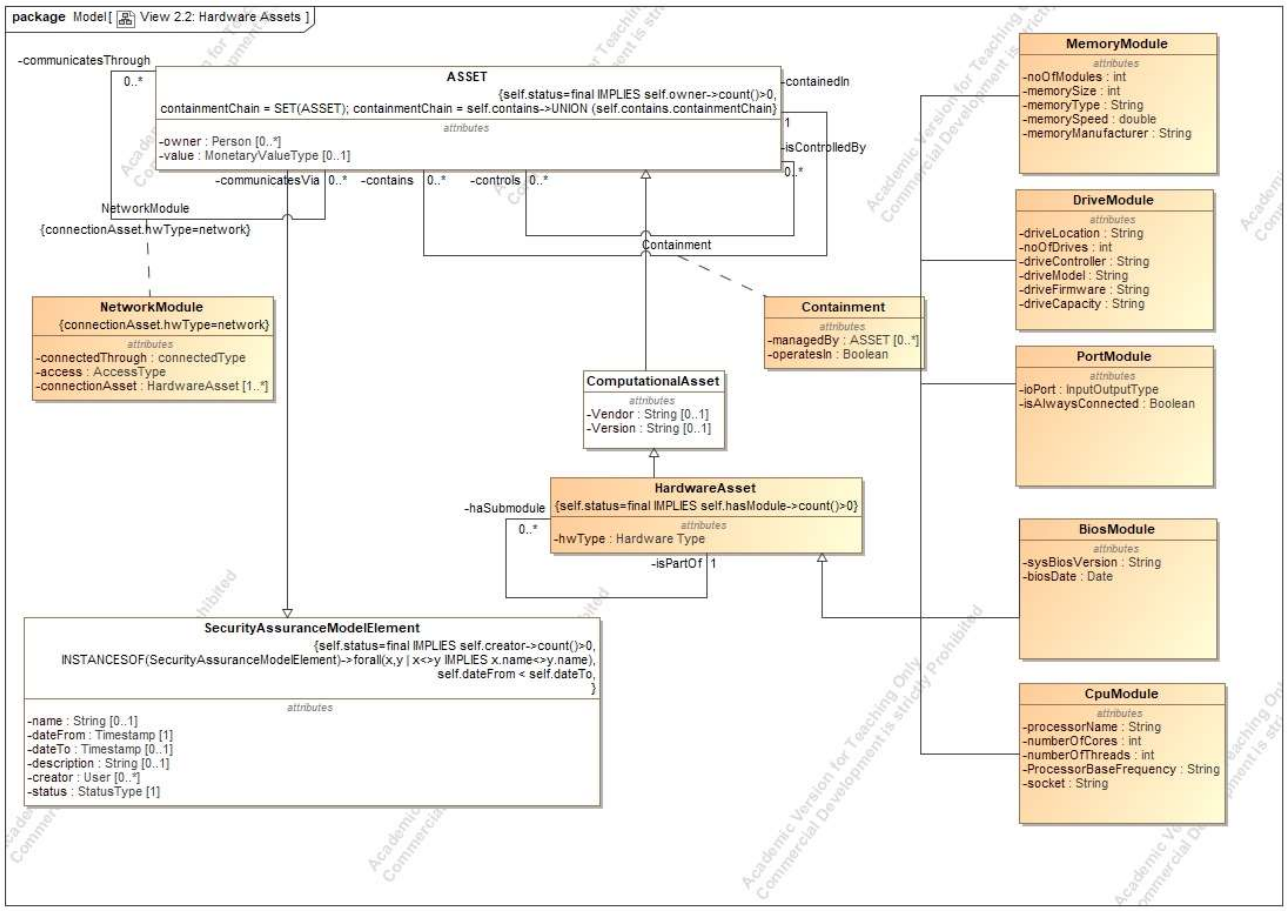

Figure 5 Hardware Asset sub-model view

Hardware Asset: The hardware components of a system (e.g., memory, CPU modules) (see Figure *5*).

Attributes:

- **hwType:** The type of the hardware asset. It can either be a **hardware computational asset** or a **network hardware asset (e.g. a router).** A hwType=network is different from the PhysicalAsset.physicalType=network as the latter describes networking cables etc.

Associations:

- **HardwareAsset** hasSubmodule 0…* **HardwareAsset(s)**
- **HardwareAsset** isPartOf 1 **HardwareAsset.**

Inheritance:

- **HardwareAsset inherits** the **Asset** class (and every other class inherited by the superclass).

Memory Module: The memory component of a system (e.g., 8GB RAM, DDR4, 3000 MHz, Vendor).

Attributes:

- **noOfModules:** The number of memory module(s).
- **memorySize:** The size of each memory.
- **memoryType:** The type of the memory (DIMM etc.)
- **memorySpeed:** The speed (in KHz) of the memory module
- **memoryManufacturer:** The vendor of the memory

Inheritance:

- **MemoryModule** inherits the **HardwareAsset** class (and every other class inherited by the superclass).

Drive Module: The drive(s) of a system (e.g., primary hard drive).

Attributes:

- **driveLocation:** The place where a drive is located within a hardware asset.
- **noOfDrives:** The number of the drives.
- **driveController:** The controller of the drive.
- **driveFirmware:** The existing firmware of a drive.
- **driveCapacity:** The capacity of the drive.

Inheritance:

- **DriveModule** inherits the **HardwareAsset** class (and every other class inherited by the superclass).

Port Module: The port(s) of a system (e.g., USB port).

Attributes:

- **ioPort:** The type of the port (e.g., usb,network,graphics,audio).
- **isAlwaysConnected:** A value that allows the end-user to know if the specific port is always connected to the asset.

Inheritance:

- **PortModule** inherits the **HardwareAsset** class (and every other class inherited by the superclass).

Bios Module: The bios of a system.

Attributes:

- **sysBiosVersion:** The installed version of the bios.
- **biosDate:** The date this bios was released.

Inheritance:

- **BiosModule** inherits the **HardwareAsset** class (and every other class inherited by the superclass).

CPU Module: The central processing unit of a system.

Attributes:

- **processorName:** The name of the processor.
- **numberOfCores:** The number of physical cores.
- **numberOfThreads:** The number of the threads.
- **ProcessorBaseFrequency:** The base frequency of the CPU.
- **Socket:** The socket name.

Inheritance:

- **CPUModule** inherits the **HardwareAsset** class (and every other class inherited by the superclass).

3.1.3.3 Data Assets sub-model

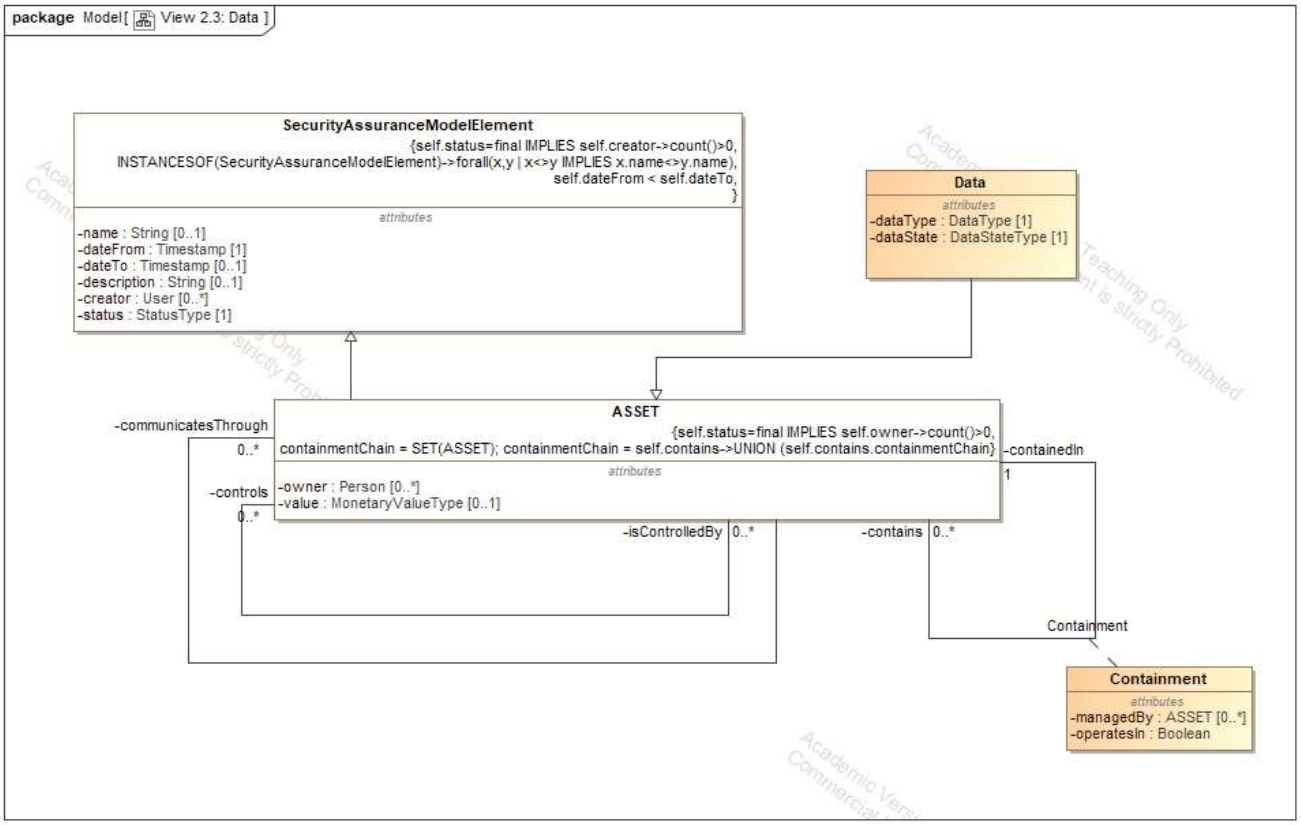

Figure 6 Data Asset sub-model view

Data: The operational and administrative data of the organization as well as the configuration settings of a system (e.g., security data stored in an offline hard drive that keep the employee's contracts) (see Figure 6)

Attributes:

- **dataType:** The type of the data. It can be one of the following: (a) operational, (b) admin, (c) config, (d) security.
- **dataState:** The state of the data. It can be one of the following: (a) in_transit, (b) in_processing, (c) at_rest and (d) offline (data that are at_rest and not used).

Inheritance:

- **Data** inherits the **Asset** class (and every other class inherited by the superclass).

3.1.3.4 Person Assets sub- model

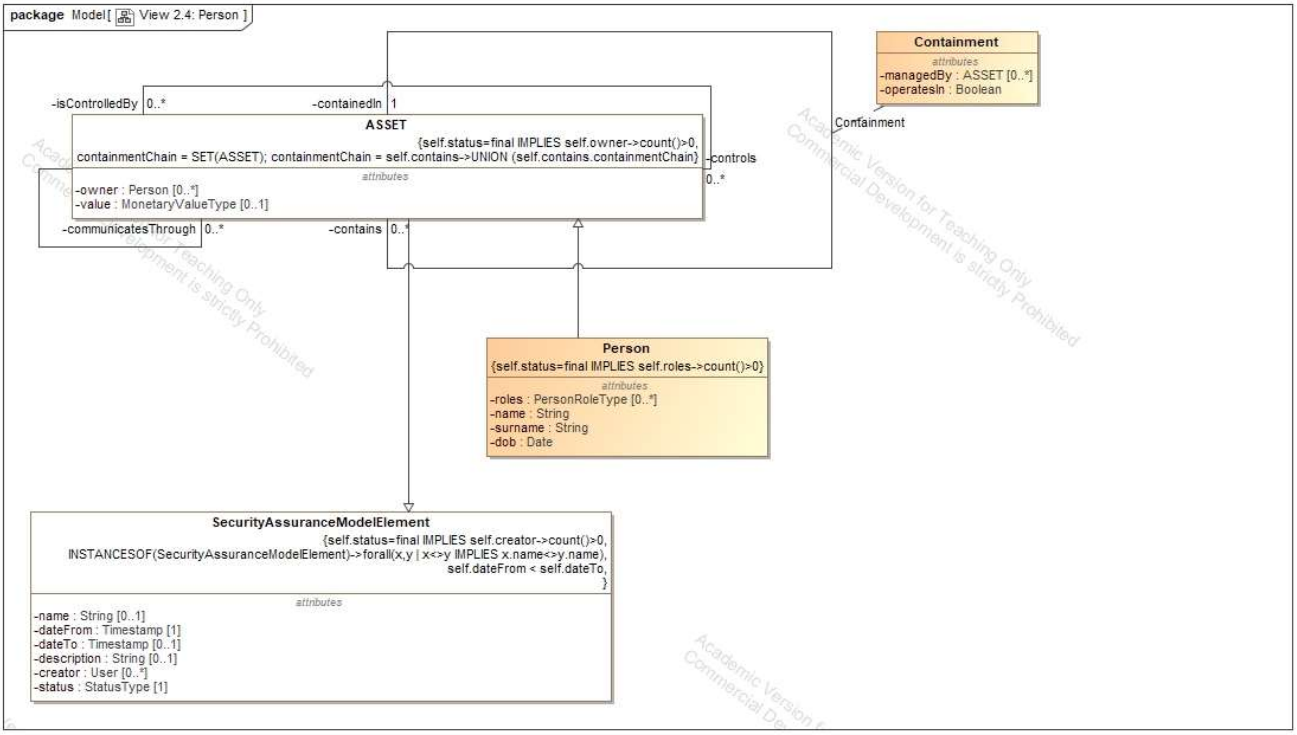

Figure 7 Person Asset sub-model view

Person: The people working within an organization (see Figure 7).

Attributes:

- **name:** The name of the person.
- **surname**: The last name of the person.
- **date of birth**: The date of birth of the person.
- **role**: The role of the person. It can be one of the following: (a) system administrator, (b) end-user, (c) asset owner, (d) asset developer, (e) asset controller.

Inheritance:

- Person inherits the **Asset** class (and every other class inherited by the superclass).

3.1.3.5 Network Asset Sub-model

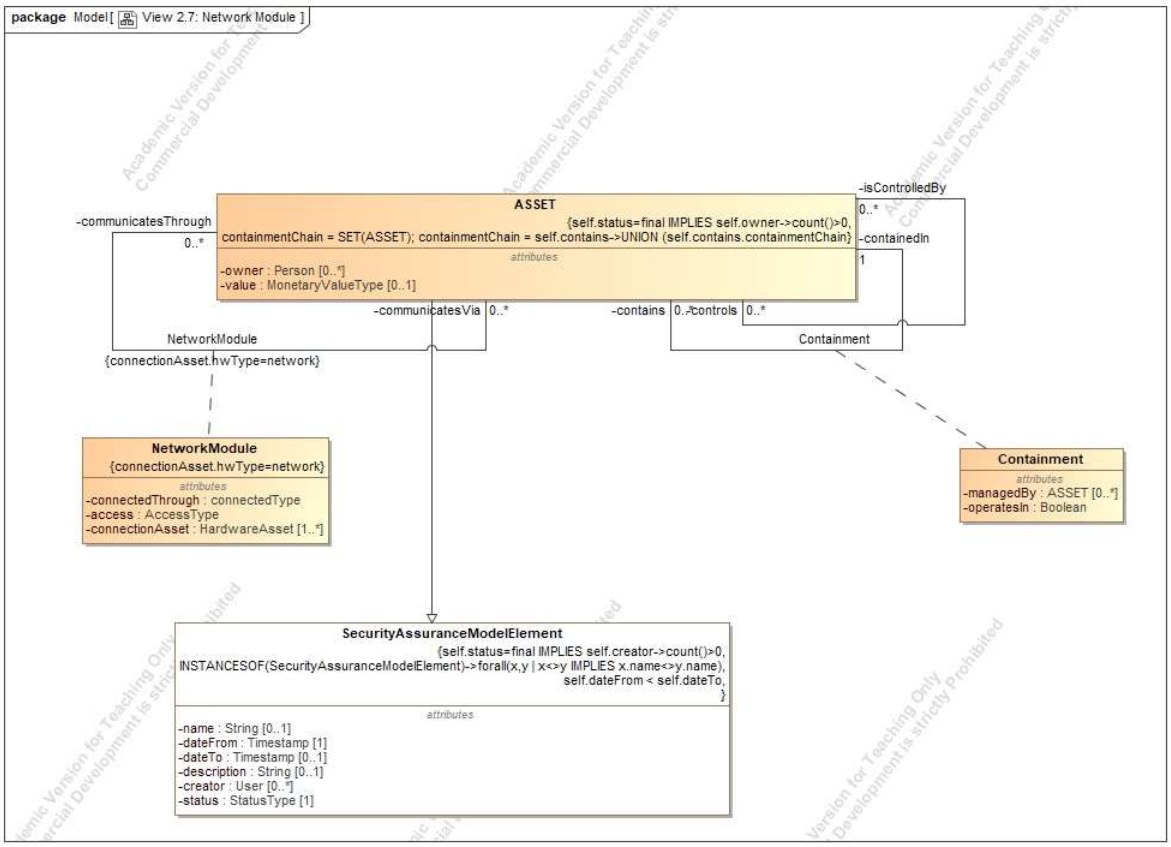

Figure 8 Network Asset sub-model view

Network Module: The network module of an asset. When the hardware type is equal to **network**, then the network module must be provided (e.g. a connection between two hardware assets through a secure Wi-Fi connection) (see Figure 8).

Attributes:

- **connectedThrough**: The type of connection. It can be one of the following: (a) wifi, (b) ethernet, (c) radio and (d) bluetooth.

Inheritance:

- **NetworkModule** inherits the **Asset** class (and every other class inherited by the superclass).

3.1.3.6 Process sub-model

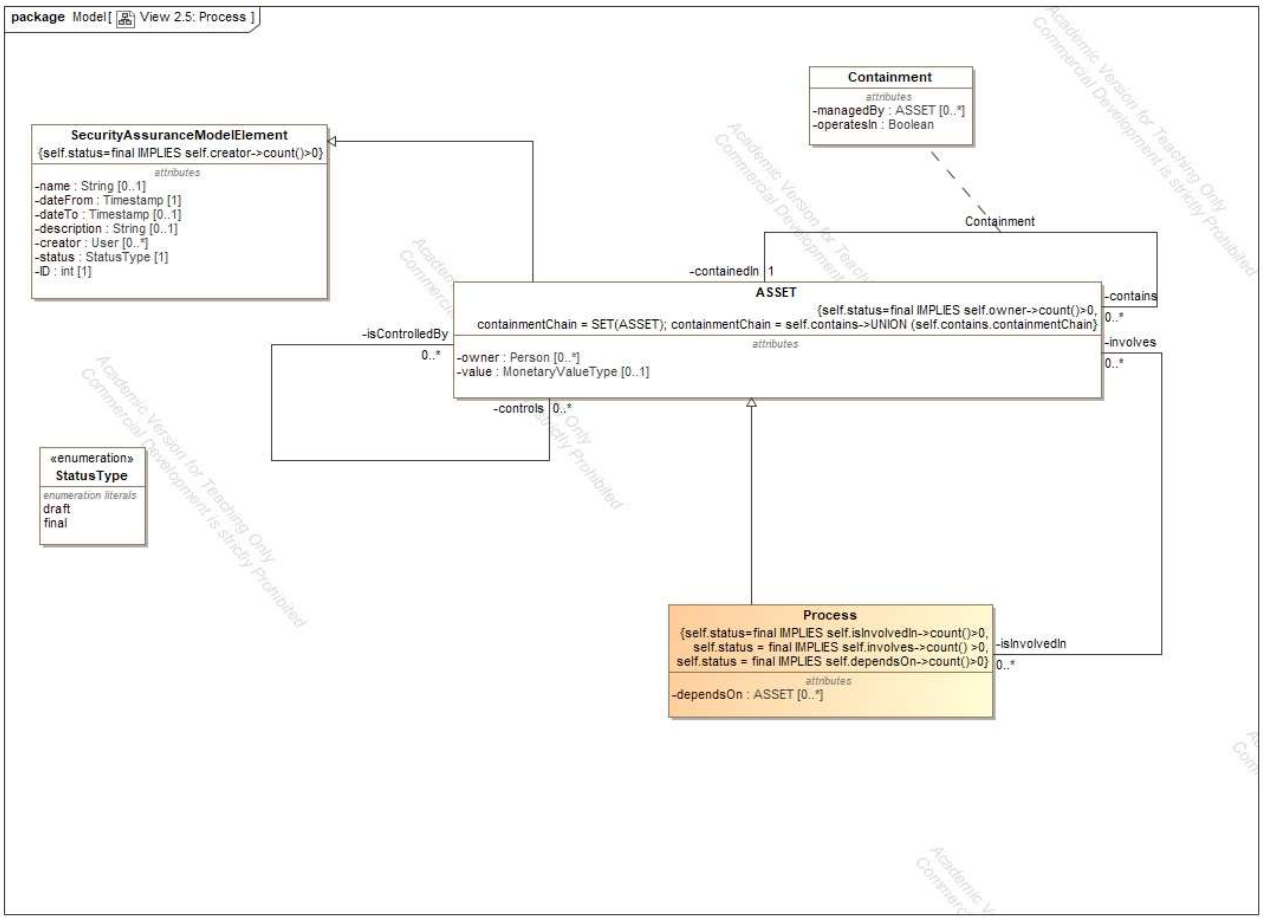

Figure 9 Process Assets sub-model view

Process: The structured activities or tasks by people or equipment which produce a service or product if a specific sequence of actions exists (e.g., patient admission, payroll process, product design) (see Figure 9).

Attributes:

- **dependsOn:** The number of assets a process is depending on.

Associations:

- **Process** involves 0…* **ASSET**

Inheritance

- **Process** inherits the **Asset** class (and every other class inherited by the superclass).

3.1.3.7 Security Controls Asset sub- model

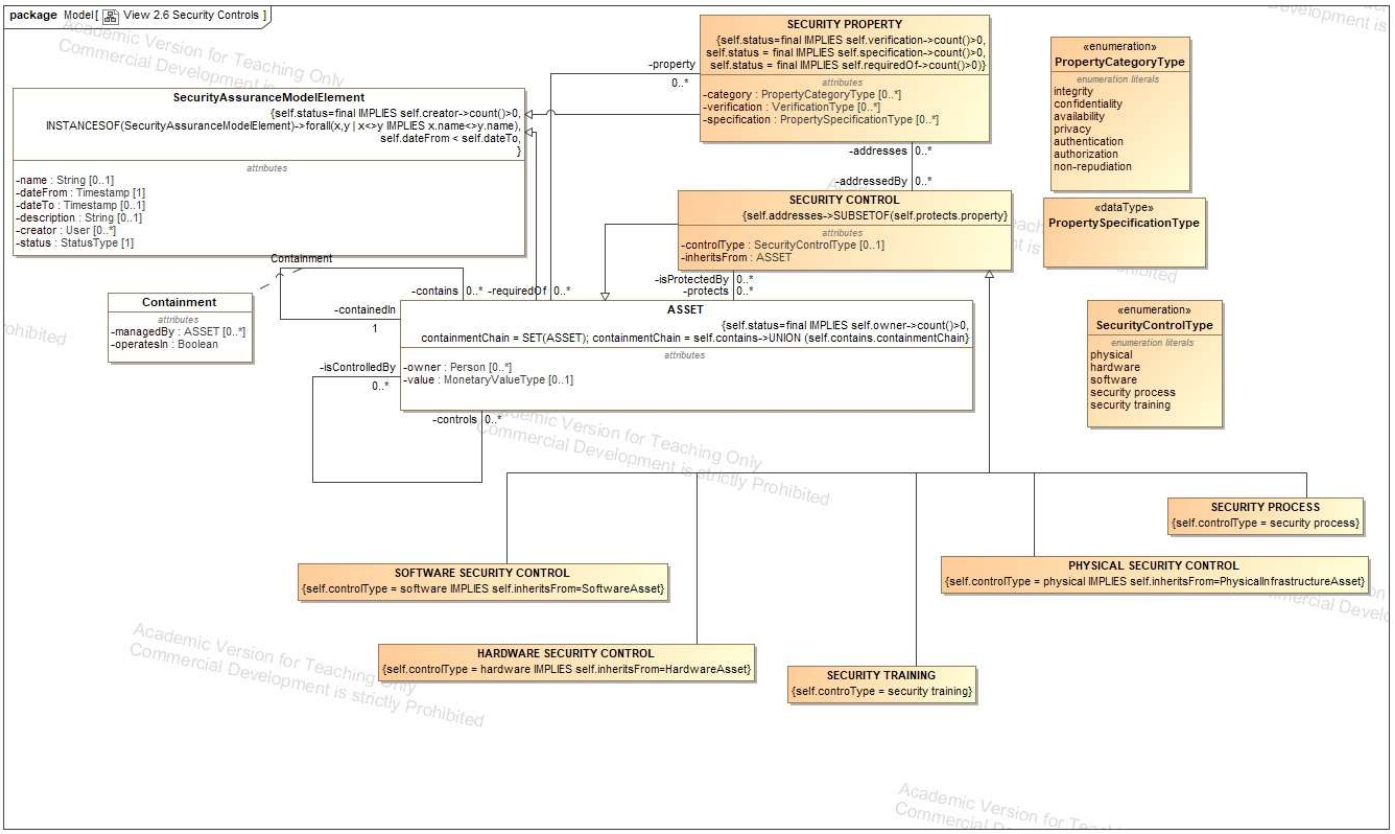

Figure 10 Security Controls Assets sub-model view

The Security Control asset's view contains the following key elements:

SECURITY CONTROL: Safeguards or countermeasures to avoid, detect, counteract, or minimize security risks to physical property, information, computer systems, or other assets (e.g., firewall) (see Figure 10).

Attributes:

- **controlType:** The type of the security control. It can be one of the following: (a) physical, (b) hardware, (c) software, (d) security process and (e) security training.

Association(s):

- **SecurityControl** addresses 0…* **SecurityProperty(ies).**

Inheritance:

- **SecurityControl** inherits the **Asset** class (and every other class inherited by the superclass).

3.1.3.8 Property sub-model

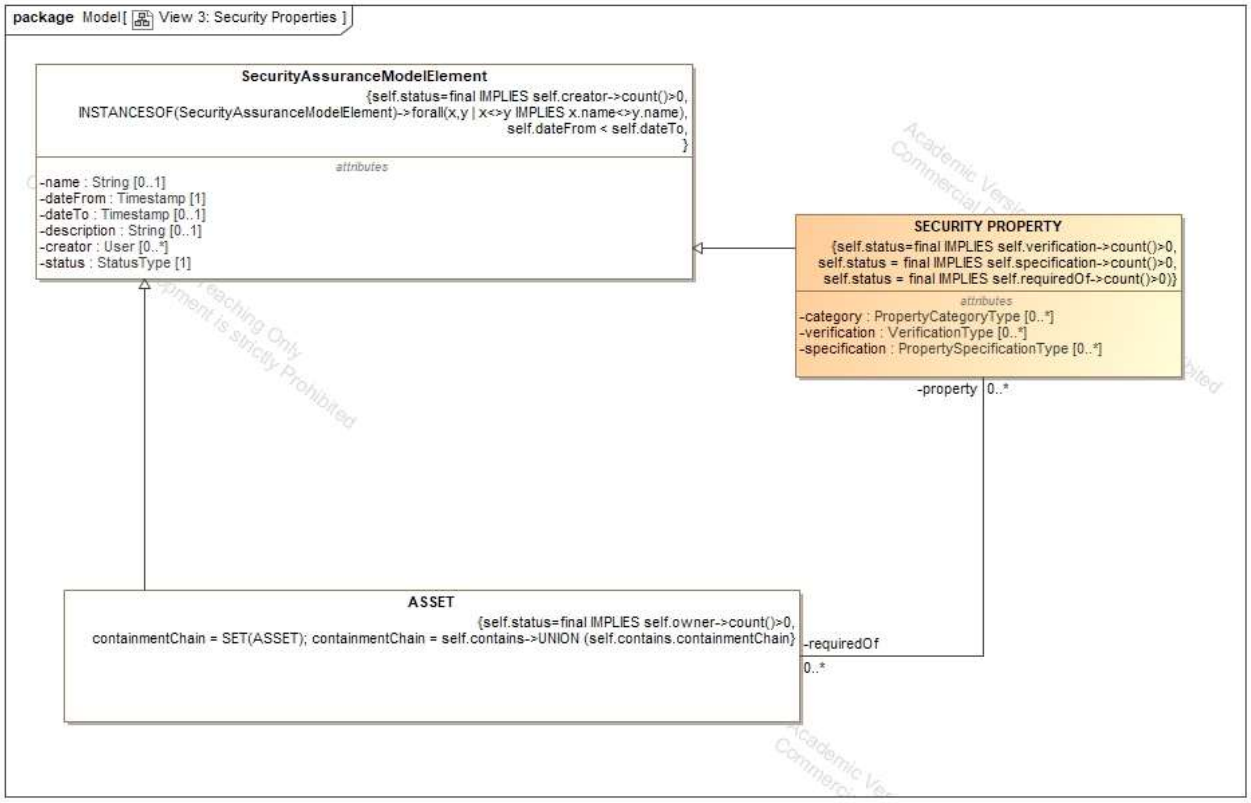

Figure 11 Security Property sub-model view

The security property asset's view contains the following key elements:

SECURITY PROPERTY: A security property is an implementation element used in an asset (see Figure 11).

Attributes:

- **Category:** The category of the security property. It can be 1...* of the following types: : (a) integrity, (b) confidentiality, (c) availability and (d) privacy.
- **Verification:** The type used to verify the property. It can be 1...* of the following types: (a) testing, (b) monitoring, (c) inspection, (d) static analysis and (e) what-if analysis.
- **Specification:** The specification of the security property. It includes the language and the specification.

Association:

- **SecurityProperty** requiredOf 0...* **Asset(s).**

Inheritance:

- **SecurityProperty** inherits the **SecurityAssuranceModelElement.**

3.1.4 Threats sub-model

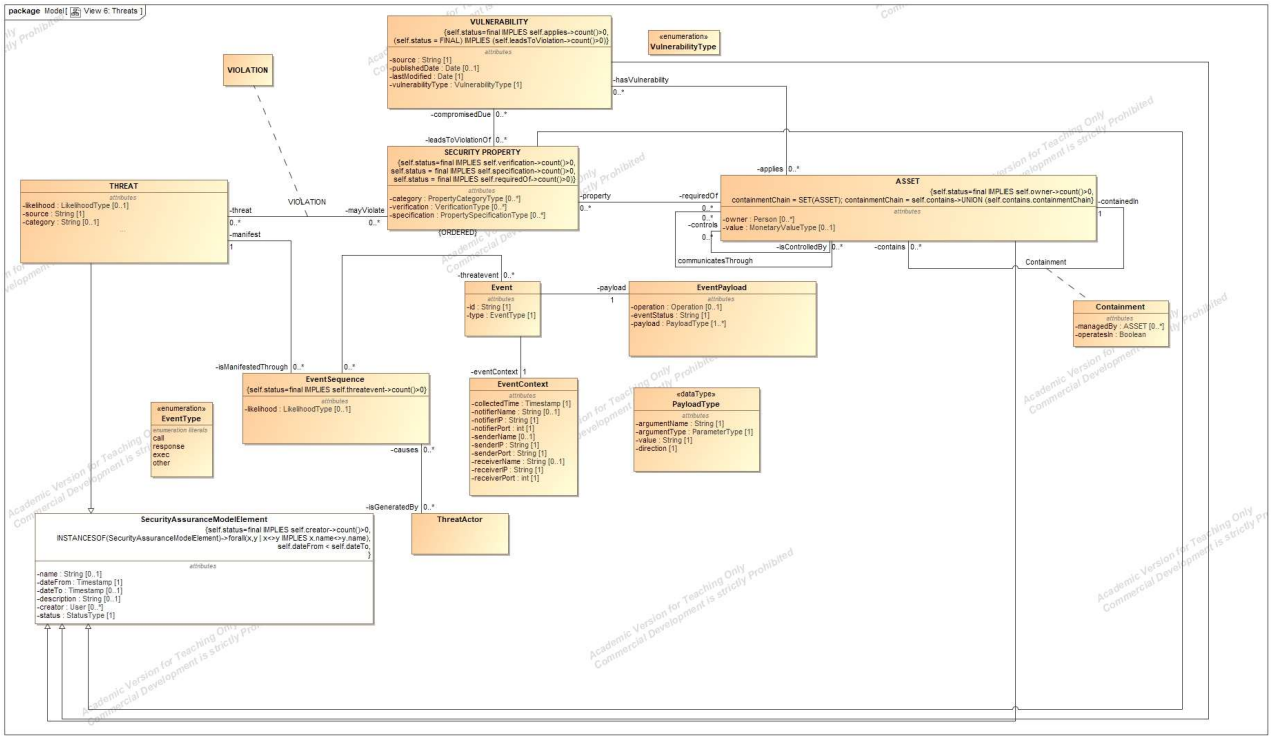

Figure 12 Threat sub-model view

Figure 12 contains elements described in subsection 3.1.2, as well as the following.

EventSequence: The sequence of events needed to manifest one threat.

Attributes:

- **Likelihood:** The likelihood for such a sequence to happen.

Association:

- **EventSequence** manifest 1 **Threat.**
- **EventSequence** isGeneratedBy 0..* **ThreatActor(s).**
- **EventSequence** threatEvent 0..* **Event(s).**

Event: The events used in the Event sequence class.

Attributes:

- **Type:** The type of the event. It can be one of the following: (a) call, (b) response, (c) execute or other.

Association:

- **Event** hasPayload 1 **EventPayload**

Payload: The main part of an event.

Attributes:

- **Operation:** The operation the event created from.
- **eventStatus**: An event can either be a request or a response.

- **Payload**: The main body of the event. It contains: (a) the argument name, (b) the argument type, (c) the value and (e) the direction of the event. If the event status is a request then the direction is **in**, elsewhere is **out.**

3.1.5 Vulnerability sub-model

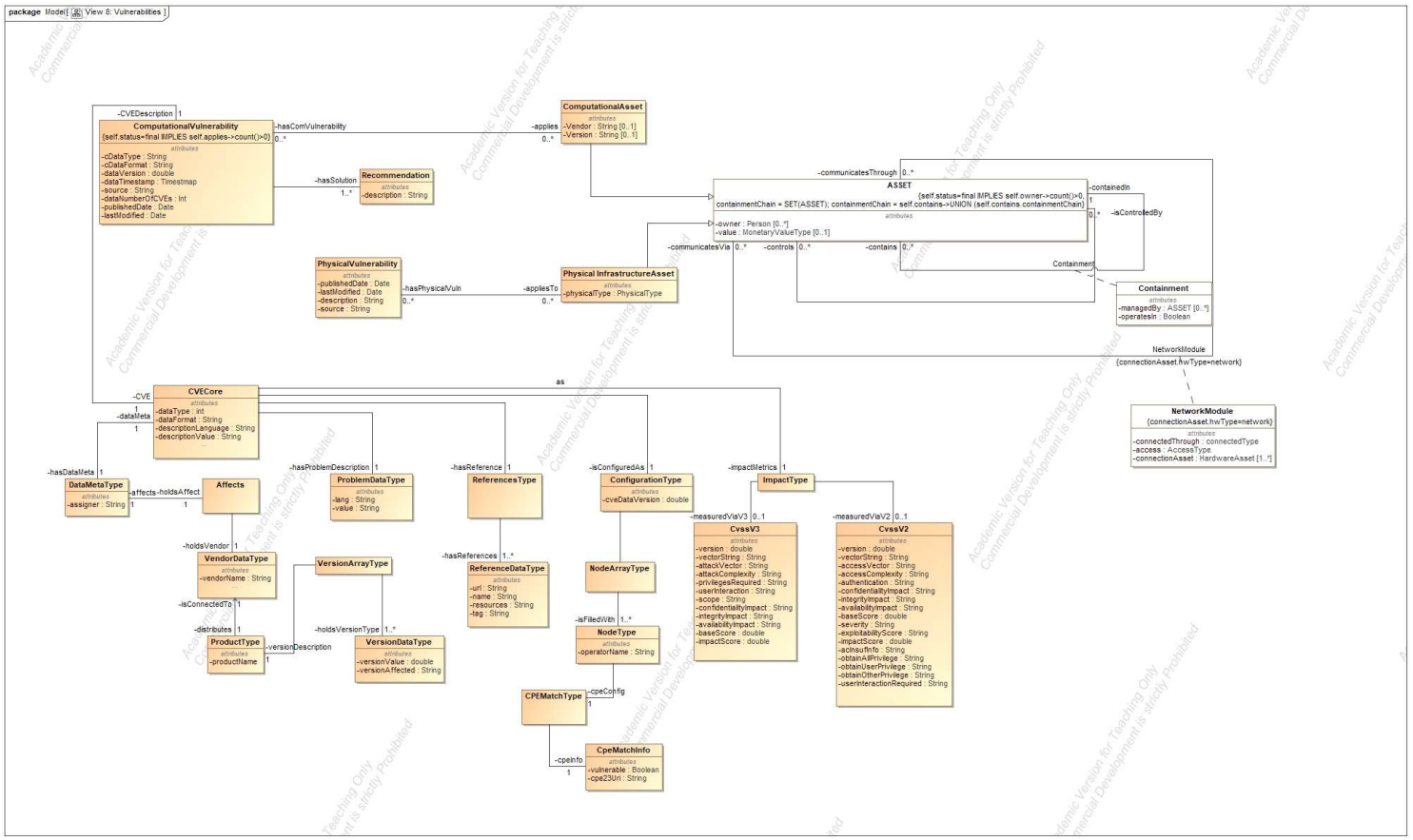

Figure 13 Vulnerability sub-model view

The vulnerability type is described in subsection 3.1.2. If the vulnerability type is equal to computational, then it can be fetched from existing vulnerability assessment tools or from vulnerability databases (e.g., the NVD (Nvd.nist.gov, 2019)). Such vulnerabilities follow the JSON format provided by the National Institute of Standards and Technology (NIST) in the NVD.

3.2 The Cyber Threat Training and Preparation (CTTP) sub-model

The **Cyber Threat Training and Preparation** (CTTP) models are leveraged to drive the THREAT-ARREST CTTP programmes. In order to drive the execution of simulation, emulation, and serious gaming processes that will realize a training programme, the CTTP models specify: (a) the cyber system components and cyber threats covered by a CTTP programme, (b) the ways of simulating components of a cyber-system and the cyber-attacks against it, (c) the components of the system that may be emulated and the ways of emulating them, (d) the real system operational events that should be monitored and analysed in order to assess the operational security status of a cyber-system in real time, (e) the actions that stakeholders are expected to take against cyber-attacks (e.g., preparedness, incident detection and analysis, real time incident response, and post incident response) and the tools that may be used for this purpose, and (f) the ways of measuring the preparedness and effectiveness of stakeholders in different stages of the CTTP programme.

The CTTP sub-model builds on top of the Core Assurance Model presented previously, with extensions to that will be leveraged by the THREAT-ARREST platform to define the training programmes. This CTTP extension is graphically depicted in Figure 14 below, while the subsections that follow present different views of this model, with descriptions of its components. A full-page view of the same model is provided in Appendix I – The CTTP sub-model (full view).

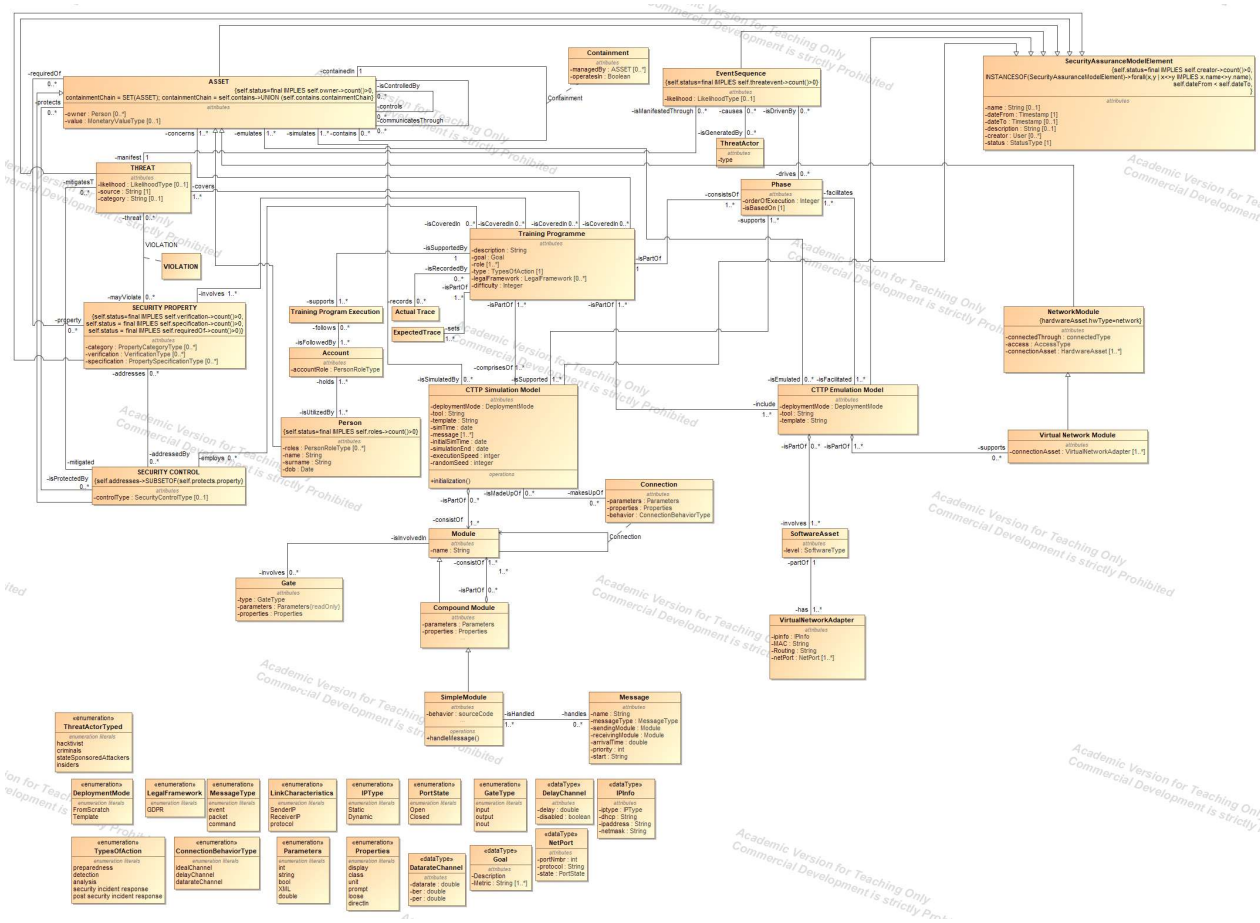

Figure 14 The CCTP sub-model

3.2.1 Training Programme sub-model

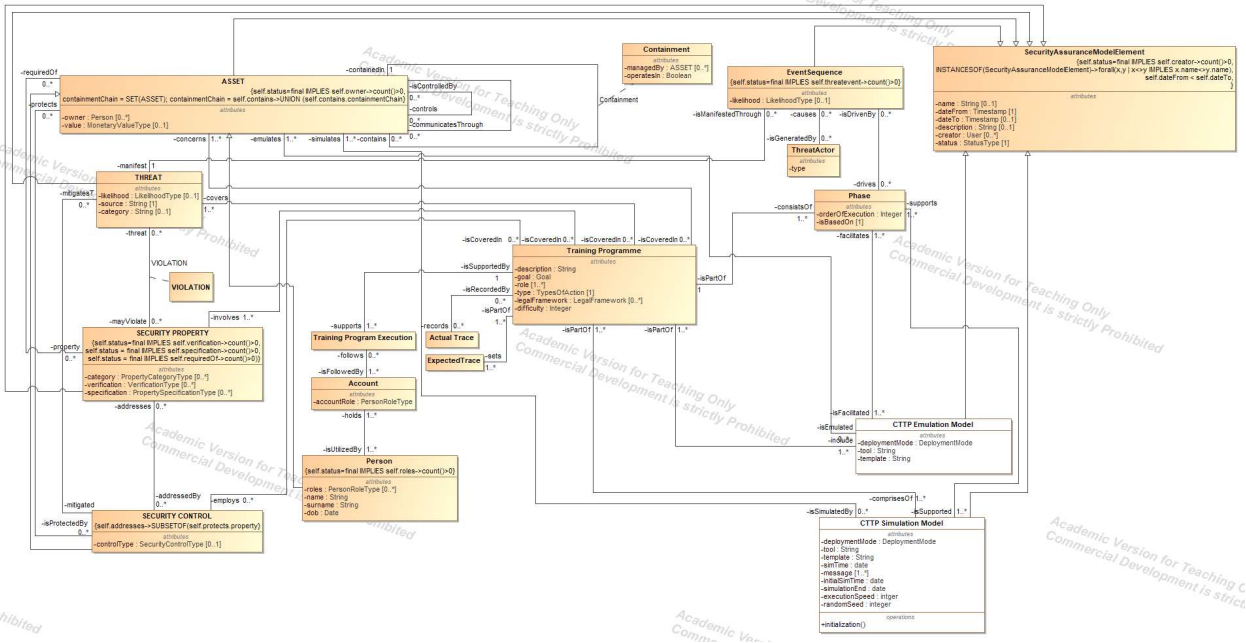

Figure 15 Training Programme sub-model view

Training Programme: A generated scenario based on a set of characteristics that is used to train users of a system (see Figure 15).

Attributes:

- **Description:** a brief text that describes the training program;

- **Goal:** The measurable target for the program. For example, in a phishing scenario, identify 50% of the malicious emails;
- **Role**: indicates the specific user roles trained in this scenario;
- **Type:** indicates the type of the training program (e.g., analysis, detection, preparedness etc.);
- **LegalFramework:** list of standards or security regulations that a training program can be aligned with;
- **Difficulty:** is a numeric value indicating the difficulty of a specific Training Program. For example, a simple phishing scenario could have a difficulty of 1/10 while a more complex 5/10.

Associations:

- The Training Programme covers:
  - 1…* ASSET;
  - 1…* THREAT;
  - 1...* SEC PROPERTIY;
  - 0…* SEC CONTRO;
- The Training Programme records 0...* Actual Trace;
- The Training Programme sets 1...* Expected Trace;
- The Training Programme supports 1...* Training program execution;
- The Training Programme consists of 1...* Phases;
- The Training Programme contains 1...* CTTP Simulation Model;
- The Training Programme includes 1...* CTTP Emulation Model;
- The Training Programme inherits from Security Assurance Model Element.

Actual Trace: The evidences gathered from the whole training program. For example, the logs of an emulation component, or statistics of a simulation environment.

Expected Trace: Defines evidence that are essential for tracking the progress of a user as part of the training program. For example, a user examines a malicious email, the email contains a link; the expected trace is if the user presses the link or not.

Training Program Execution: Describes what actions are enabled on the training program considering the role of the account that is using the training.

Associations:

- A training program execution is followed by 1…* account.

Account is a role-specific means of entry to the training program.

Attributes:

- AccountRole (e.g., forensics, sec engineer, admin, red/blue team).

Associations:

- An account is utilized by 1...* Person.

For example, a red team/blue team might have a single account for training its team members. An account can be used by a security engineer that wants to train on different positions, like system admin, forensics, blue/red team etc.

Person: A user of the system; this class is described in the core model (see 3.1.3.4)

Phase: Describes the stages of the realization of the threat scenario.

Attributes:

- **OrderOfExecution:** A numeric value indicating the priority of the steps for the realization of the stage.

Associations:

- A Phase is based on 1 eventSequence.

EventSequence involves certain steps required for the manifestation of a threat; EventSequence is described in the Threats sub-model (section 3.1.4).

ThreatActor is a malicious entity responsible for the manifestation of a threat.

Attribute:

- **Type:** The available types of threat actors include hactivist, criminal, insider, stateSponsored.

Associations:

- Threat Actor causes 0...* EventSequence

3.2.2 CTTP Simulation sub-model

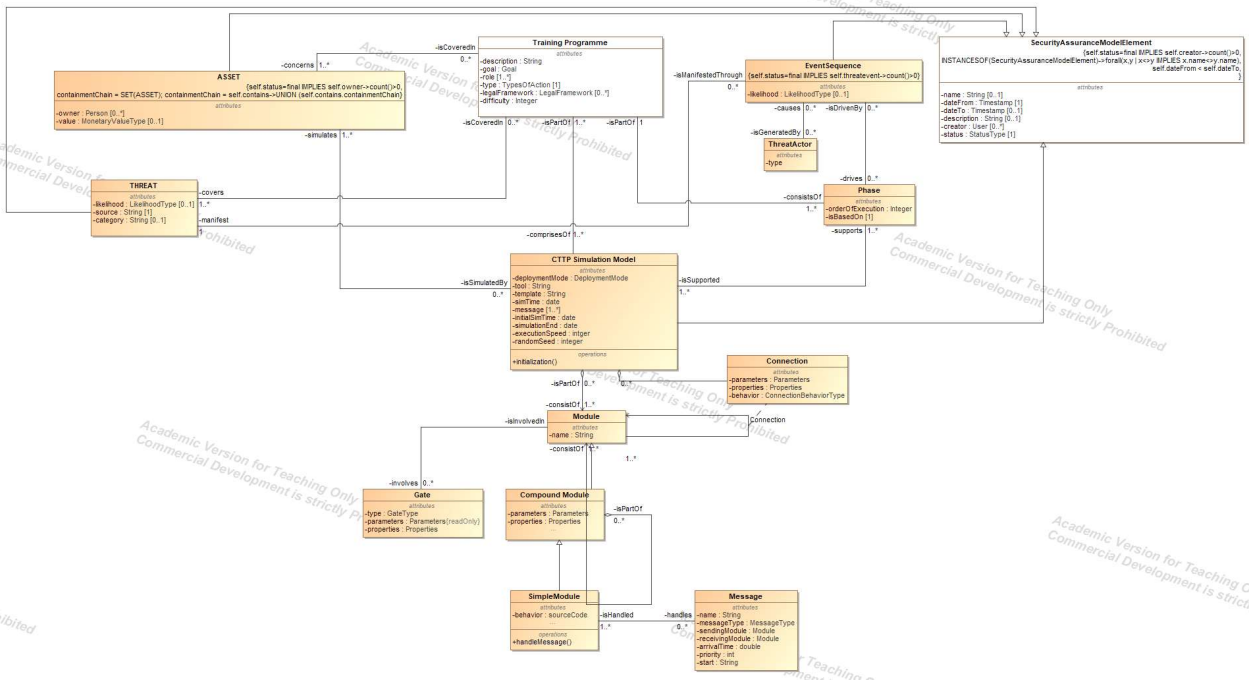

Figure 16 CTTP Simulation sub-model view

The **CTTP Simulation Sub-Model** is responsible for indicating if an asset can be simulated and describing the simulation and its individual component (see Figure 16).
An asset can be simulated if we can implement it using existing modules (one or more simple and/or compound modules). For example, to simulate an email server, we need a simple module that can imitate the functionality (inputs and/or outputs) of an email server.

Attributes:

- **DeploymentMode:** Specifies which modes of deployment are supported (e.g., build from the scratch, pre-set).
- **Tool:** Specifies the tool that will realize the simulation (e.g., Jasima (Jasima, 2019) Omnet++ (Simulator, 2019)).
- **Template (optional):** Provides the path for the file to be used by pre-set mode.

- **SimTime:** Stores the current simulation time and date;
- **InitialSimTime:** Indicates the time and date that the simulation started;
- **SimEnd:** Shows the time and date that the simulation will end;
- **ExecutionSpeed:** Indicates how fast is the simulation time passing; If this would be set to, e.g., 2, it would mean the simulation tool would progress simulation time synced with real time but progressing at twice the speed. Setting it to 0 would mean "as fast as possible".
- **RandomSeed:** Value used for random influences in a simulation model in order to have reproducible randomness. The seed during such random number generation is actually the starting point in the sequence. If we use same seed every time, it will yield same sequence of random numbers.
- **Message:** Represent events, packets, commands, frames, anything that is exchanges between modules. Messages can contain arbitrarily complex data structures. Simple modules can send messages either directly to their destination or along a predefined path, through gates and connections;

Functions:

- **Initialization**: Enables the initialization of the model. It contains the necessary code. In the current version, we include the instantiation scripts of the Jasima simulator (see D5.2 for more details).

Associations:

- CTTP Simulation Model is part of 1...* *Training Program(s);*
- CTTP Simulation Model has 1...* *Phase(s);*
- CTTP Simulation Model Simulates 1...* *ASSET(s);*
- CTTP Simulation Model consist of 1...* *Module(s);*
- CTTP Simulation Model Support 0...* *Connection(s).*

Module: A node on the simulation network (i.e., the simulation environment). A module can either be Simple or Compound.

Attributes:

- **Name**: The name of the module

Associations:

- A module communicates with another module via one or more (1...*) connection(s). (adjacent node or via a path of connections);
- A module involves 0…* Gate(s).

Compound Module: A group of modules. It can be used to simulate a closed system that has 1 or more simple/complex functionalities.

Attributes:

- **Parameters** (see below for the description);
- **Properties** (see below for the description).

Associations:

- A compound module inherits from Module;
- A compound module consists of 1...* *Module(s);*
- A compound module inherits from *Module.*

Simple Module: Contains the source code to simulate a component (e.g., hardware, software) and handles the messages.

Attributes:

- **Behavior:** The source code.

Functions:

- **HandleMessage:** enables the processing of the message. This include sending the message to other modules, scheduling a message and cancelling a message.

Associations:

- A simple module inherits from *Compound Module*
- A simple module handles 0...* Message(s)

Connection facilitates the communication of two modules via their *Gates*.

Attributes:

- **Parameters** (see 3.1.3.1)
- **Properties** (see 3.1.3.1)
- **Behavior,** defines characteristics of the connection such as datarate, delay etc. Behavior

Gate serves as a connection point of a Module. *input*, *output* and *inout*,

Attributes:

- **Parameters (see below, examples to be added)**
- **Properties (see below, examples to be added)**
- **Type** defines the functionality of the gate. It can be input, output and *inout*, A gate, whether input or output, can only be connected to one other gate. (For compound module gates, this means one connection "outside" and one "inside".)

Associations:

- Gate is part of 1 *Simple Module*.
- Gate is part of 1 *Compound Module*.

Parameters: Variables of an object. Variables can be int, bool, double, string, XML or any other know data type. For example, for an App – Simple Module, the parameters may be the communication protocol (i.e. UDP/TCP/ICMP, destination address, packet length of a message etc.)

Properties: Various metadata that can be attached to the different objects of the CTTP Simulation Model. For instance, properties can be statistics that are needed for the Expected Trace, such as end-to-end delay, jitter etc. Properties can also be rendering information of the specific object for the GUI.

3.2.3 CTTP Emulation sub-model

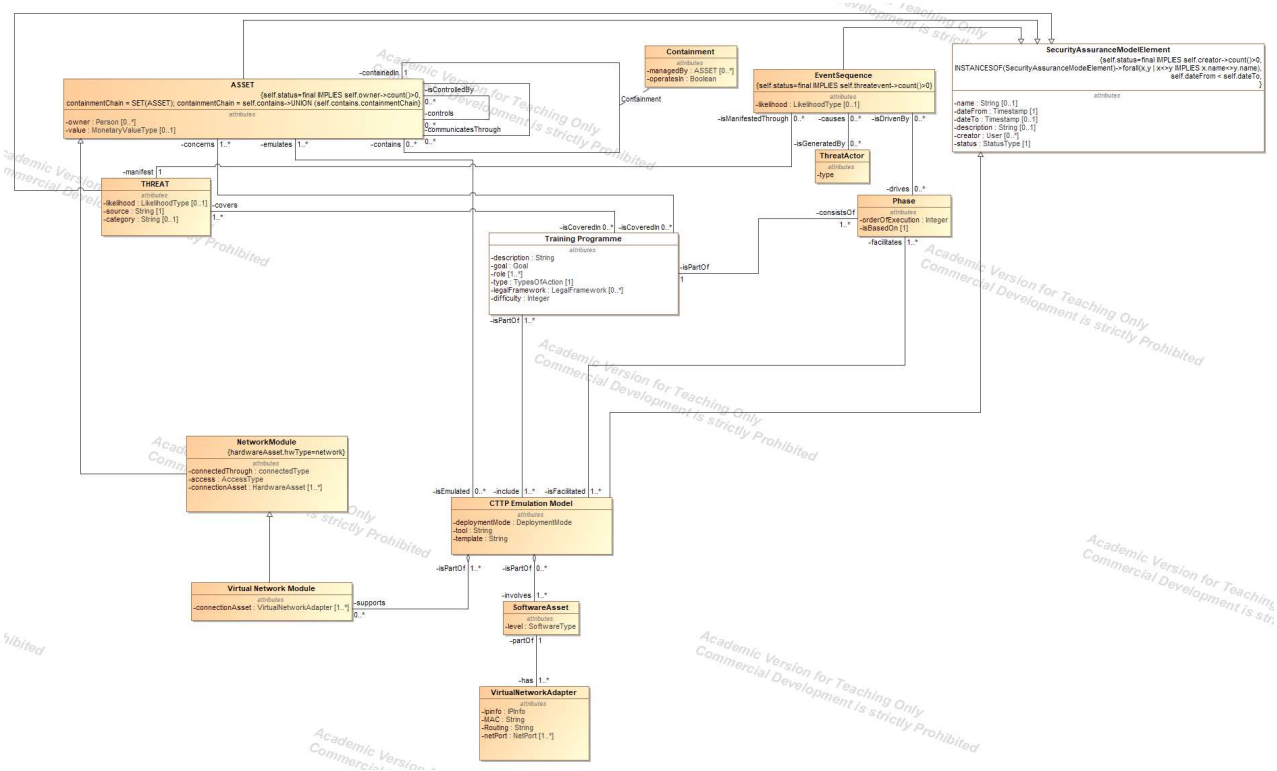

Figure 17 CCTP Emulation sub-model view

The **CTTP emulation sub-model** (see Figure 17) indicates if an Asset can be emulated and is responsible for defining the information required by the Training Programme to emulate Assets and facilitate connections between them, with simulated Assets and possibly external real assets (e.g., external email server via the Internet).
An asset can be emulated if we have a template (e.g., Openstack template) or the necessary resources (e.g., ISO) for it.

Attributes:

- **DeploymentMode:** Specifies which modes of deployment are supported (e.g., build from the scratch, pre-set);
- **Tool:** Indicates the tool that will realize the emulation (e.g. Openstack (OpenStack, 2019));
- **Template (optional):** Provides the path for the file to be used by pre-set mode.

Functions:

- **Initialization**: Enables the initialization of the model. It contains the necessary code. In the current version, we include the instantiation OpenStack scripts (see D5.3 for more details).

Associations:

- CTTP Emulation Model is part of 1...* *Training Program(s)*
- CTTP Emulation Model has 1...* *Phase(s)*
- CTTP Emulation Model emulates 1...* *ASSET(s)*
- CTTP Emulation Model consists of 1...* *SoftwareAsset(s)*
- CTTP Emulation Model supports 0...* *VirtualNetworkModule(s)*

VirtualNetworkModule: Supports the connection between two virtual components.

Attributes:

- **connectionAsset**: Includes 1…* VirtualNetworkAdapter(s) that participate in the connection.

Associations:

- **VirtualNetworkModule** inherits from the ***NetworkModule*** (see 3.1.3.5).

SoftwareAsset: Involves applications of the system. The SoftwareAsset is described in the core model (see 3.1.3.1)

Associations:

- SoftwareAsset has 1...* *VirtualNetworkAdapter(s).*

VirtualNetworkAdapter: Describes the network information required for a virtual connection. This is the hardware equivalent of the hardware network card.

Attributes:

- **IPinfo:** Includes IP information (i.e., Static or Dynamic IP, if dynamic the DHCP, IP address, Netmask);
- **MAC:** Describes the virtual MAC address;
- **Routing**: Includes routing information;
- **NetPort**: Indicates information of the ports used by the virtual components. Specifically, the port number (i.e., 0 - 65535), the protocol (e.g., TCP, UDP) and its state (i.e., OPEN, CLOSE).

3.2.4 Gamification

The CTTP Model also supports the needs of the gamification tool through the training programme sub-model. In this context, a subset of the attributes (see section 3.2.1) of the Training Programme will be used to instantiate a gaming tool (e.g. the online card game PROTECT) within the gamification tool. To this, the attribute **description** determines the learning content for the game (e.g., phishing attacks, external attacks) to be used and the attribute **difficulty** sets the difficulty of the game. At the example of PROTECT, these attributes will define which card deck of PROTECT is played at which difficulty level.

The **goal** specifies the training target for the gamification scenario (e.g. answer 50 % of the questions correctly) within the CTTP model. This information may be later related to the result of the corresponding game, which is returned by the gamification tool, within the assessment of a trainees. Regarding to PROTECT, the returned result for a game contains the number of attacks that have been solved correctly and incorrectly.

During the actual project phase of THREAT-ARREST the learning content of the gaming tools will address solely the theme of social engineering. Accordingly, the attribute **LegalFramework** in the context of gamification may reserved for a possible future use (e.g. GDPR related learning content).

4 Language constructs specification – The THREAT-ARREST Grammar

Based on the Core Assurance and CTTP models defined in the previous section, a grammar is derived that will form the basis for the definition of CTTP programmes. To achieve this, a Java Maven Project was created using ANTLR (a parser generator, i.e. a tool for processing structured text (ANTLR, 2019). ANTLR is used to generate the source code for several tools based on the language specification, including, language recognizers, analyzers and translators. To that end, ANTLR receives as an input the grammar below and subsequently generates the source code files. The grammar is a precise description of a language with additional semantic actions and consists of several parser and Lexer rules (i.e., constructs). Using this grammar, the main tools generated by ANTLR are Lexers (a.k.a tokenizer, scanners) and Parsers. A Lexer reads an input character or byte stream, splits it into tokens using specific rules (defined in the language) and generates a token stream output. Lexers can also be used to flag specific tokens that for example we want to ignore (e.g., whitespace, comments). A Parser reads a token stream generated by the Lexer, matches phrases in the specified language via the parser rules (defined in the language) and then usually performs a predefined action.

The aforementioned grammar is listed in Table 2, while the associated language editor will be developed in the context of the same task (T3.1 – CTTP Language Definition and Tool Support) and will be provided in the deliverable "D3.2 – CTTP Models and Programmes Specification Tool". Detailed examples for the language usage will be documented in D3.2.

Table 2. THREAT-ARREST Language Constructs

Training Programme Constructs – Parser Rules	
Element Name	**Attributes & Associations**
securityAssuranceModelElement	saTitle OPEN_PAREN saID (COMMA name)? (COMMA datefrom)? (COMMA dateto) (COMMA description)? COMMA saStatus CLOSE_PAREN
asset Programme	trainingProgrammeTitle OPEN_PAREN trainingProgrammeID COMMA trainingProgrammeDescription COMMA trainingProgrammeGoal COMMA (role)+ COMMA (type)? COMMA (legalFramework)* COMMA difficulty (COMMA consistsOf)+ (COMMA involves)+ (COMMA employs)* (COMMA concerns)* (COMMA covers)+ (COMMA includesEmModel)+ (COMMA comprisesOfSimModel)+ (COMMA records)* (COMMA sets)+ (COMMA supports)+ CLOSE_PAREN
asset	assetTitle OPEN_PAREN assetID COMMA saID COMMA owner COMMA value (COMMA controledBy)*(COMMA contains)* (COMMA communicatesThrough)* (COMMA property)* (COMMA isInvolvedIn)* CLOSE_PAREN
controledByType	controledByTitle OPEN_PAREN assetID CLOSE_PAREN
containsType	containsTitle OPEN_PAREN assetID COMMA (containment)* CLOSE_PAREN
containment	containmentTitle OPEN_PAREN containmentID COMMA (managedBy)* COMMA operatesIn CLOSE_PAREN

communicatesThroughType	communicatesThroughTitle OPEN_PAREN assetID (COMMA networkModuleID)* CLOSE_PAREN
networkModule	networkModuleTitle OPEN_PAREN networkModuleID COMMA connectedThrough COMMA access CLOSE_PAREN
managedByType	managedByTitle OPEN_PAREN asset CLOSE_PAREN
threat	threatTitle OPEN_PAREN threatID COMMA saID COMMA likelihood COMMA source COMMA category (COMMA mayViolate)*CLOSE_PAREN
securityProperty	securityPropertyTitle OPEN_PAREN secPropertyID COMMA saID (COMMA category) * (COMMA verification) * (COMMA specification) * (COMMA requiredOf)* (COMMA compromisedDue)* CLOSE_PAREN;
securityControl	secControlTitle OPEN_PAREN secControlID COMMA assetID COMMA controlType (COMMA protects) * CLOSE_PAREN
account	accountTitle OPEN_PAREN accountID COMMA accountRole COMMA (follows)* CLOSE_PAREN
trainingProgrammeExecution	trainingProgrammeExecutionTitle OPEN_PAREN trainingProgrammeExecutionID COMMA (isFollowedBy)+ COMMA (isSupportedBy)? CLOSE_PAREN
person	personTitle OPEN_PAREN personID COMMA assetID COMMA (roles)* COMMA (holds)+ CLOSE_PAREN
actualTrace	actualTraceTitle OPEN_PAREN actualTraceID CLOSE_PAREN
expectedTrace	expectedTraceTitle OPEN_PAREN expectedTraceID CLOSE_PAREN
phase	phaseTitle OPEN_PAREN phaseID COMMA orderOfExecution CLOSE_PAREN
eventSequence	eventSequenceTitle OPEN_PAREN eventSequenceID COMMA saID COMMA likelihood COMMA (isGeneratedBy)* COMMA (drives)* CLOSE_PAREN
threatActor	threatActorTitle OPEN_PAREN threatActorID COMMA threatActorType COMMA (causes)* CLOSE_PAREN
CTTP Emulation Model Constructs – Parser Rules	
cttpEmulationModel	cttpEmulationModelTitle OPEN_PAREN cttpEmulationModelID COMMA saID COMMA deploymentMode COMMA emTool COMMA emTemplate COMMA (involvesSoftwareAsset)+ COMMA (supportsNetModule)* COMMA (facilitatesPhase)+ COMMA (emulates)+ CLOSE_PAREN

virtualNetworkModule	virtualNetworkModuleTitle OPEN_PAREN virtualNetworkModuleID COMMA networkModuleID COMMA (connectionAsset)+ CLOSE_PAREN
softwareAsset	oftTitle OPEN_PAREN softwareAssetID COMMA compAssetID COMMA level (COMMA provides) *(COMMA requires)* (COMMA providesImpl)* CLOSE_PAREN
virtualNetworkAdapter	virtualNetworkAdapterTitle OPEN_PAREN virtualNetworkAdapterID COMMA ipInfo COMMA mac COMMA routing COMMA (netPort)+ CLOSE_PAREN
CTTP Simulation Model Constructs – Parser Rules	
cttpSimulationModel	cttpSimulationModelTitle OPEN_PAREN cttpSimulationModelID COMMA saID COMMA deploymentMode COMMA simTool COMMA simTemplate COMMA simTime COMMA (message)+ COMMA (supportsPhase)+ COMMA (comprisesOf)+ COMMA (makesUpOf)* COMMA (simulates)+ CLOSE_PAREN
module	moduleTitle OPEN_PAREN moduleID COMMA connectsVia COMMA (involvesGate)* CLOSE_PAREN
connectsViaType	connectsViaTitle OPEN_PAREN moduleID COMMA (connectionID)? CLOSE_PAREN
connection	connectionTitle OPEN_PAREN connectionID COMA connectionParameters COMA connectionProperties COMA connectionBehavior CLOSE_PAREN
compoundModule	compoundModuleTitle OPEN_PAREN compoundModuleID COMA moduleID COMA compoundModuleparameters COMA compoundModuleproperties CLOSE_PAREN
simpleModule	simpleModuleTitle OPEN_PAREN simpleModuleID COMA moduleID COMA simpleModulebehavior COMA (handles)* CLOSE_PAREN
gate	gatelTitle OPEN_PAREN gateID COMMA gateType COMMA gateParameters COMMA gateProperties CLOSE_PAREN
message	messageTitle OPEN_PAREN messageID COMMA messageName COMMA messageType COMMA messageRecipient COMMA messageSender
Enumeration Types	
DeploymentMode	'FromScratch' |'Template'
LegalFramework	'GDPR'
MessageType	'event' |'packet' |'command'

IPType	'Static' |'Dynamic'
Parameters	'int' |'string' |'bool' |'XML' |'double'
TypesOfAction	'preparedness' |'detection' |'analysis' |'security incident response' |'post security incident response'
PortState	'Open' |'Closed'
Properties	'dislplay' |'class' |'unit' |'prompt' |'loose' |'directIn'
GateType	'input' |'output' |'inout'
PersonRoleType	'end user' |'administrator' |'asset owner' |'asset developer' |'asset controller'
ConnectionBehaviorType	'idealChannel' |'delayChannel' |'datarateChannel'
ThreatActorType	'hacktivists' |'criminals' |'stateSponsored' |'insiders'
Data Types	
delayChannel	delayChannelTitle OPEN_PAREN delayChannelID COMMA delay COMMA disabled CLOSE_PAREN;
ipInfo	ipInfoTitle OPEN_PAREN ipInfoID COMMA ipType COMMA ipAddr COMMA netmask CLOSE_PAREN;
netPort	netPortTitle OPEN_PAREN netPortID COMMA netPortNmbr COMMA netPortprotocol COMMA netPortstate CLOSE_PAREN;
goal	goalTitle OPEN_PAREN goalID COMMA goalDescription COMMA metric CLOSE_PAREN;
datarateChannel	datarateChannelTitle OPEN_PAREN datarateChannelID COMMA datarate COMMA ber COMMA per CLOSE_PAREN;

likelihoodType	likelihoodTypeTitle OPEN_PAREN likelihoodTypeID COMMA min COMMA max COMMA calculatedBy CLOSE_PAREN;
likelihoodModel	likelihoodModelTitle OPEN_PAREN likelihoodModelID CLOSE_PAREN;
Lexer Rules	
fragment ESC	'\\' (["\\/bfnrt] | UNICODE)
fragment UNICODE	'u' HEX HEX HEX HEX
fragment HEX	[0-9a-fA-F]
fragment Digit	[0-9]
Fragment TWODIGIT	Digit Digit
Fragment LETTER	[A-Za-z]
Fragment EXPONENT	('e'|'E') ('+'|'-')? ('0'..'9')+
ETimestamp	Date ' ' Time
Identifier	[a-zA-Z0-9]+
INT	[0-9]+
DOUBLE	Digit+ '.' Digit+
FLOAT	('0'..'9')+ '.' ('0'..'9')* EXPONENT? | '.' ('0'..'9')+ EXPONENT? | ('0'..'9')+ EXPONENT
NEWLINE	\r'? '\n' ; // return newlines to parser (is end-statement signal)
WS	\t]+ -> skip ; // toss out whitespace
STRING	'"' (ESC | ~["\\])* '"'
COMMA	','
OPEN_PAREN	'('
CLOSE_PAREN	')'
Date	TWODIGIT TWODIGIT '-' LETTER LETTER LETTER '-' TWODIGIT
TIME	TWODIGIT ':' TWODIGIT ':' TWODIGIT
BOOLEAN	'true' | 'false'
Year	Digit Digit Digit Digit
Month	'0' Digit | '1' '0'..'2'
Day	'0'..'2' Digit | '3' '0'..'1'
Time	Hour ':' Minute
Hour	'0'..'1' Digit | '2' '0'..'3'
Minute	'0'..'5' Digit

5 CTTP Programme Examples

This section provides an example of the use of the THREAT-ARREST CTTP models to define a training programme.

The example selected focuses on a Phishing attack, which is a common threat in all organisations, targeting employees with different levels of expertise, based on the sophistication of the attack used (e.g., varying from a generic phishing e-mail message for employees with no technical background, to targeted phishing emails, following elaborate profiling of the target, for more advanced users).

Based on the above scenario, this section aims to show the use of the CTTP models to define the different training scenarios, but also to show the different levels of detail and elaboration that the THREAT-ARREST platform will be able to provide on the same scenario: from a very simple training programme, with low trainee interactions, where most of the components can be simulated and abstracted, to more sophisticated scenarios, with realistic setups deployed, where the trainee can interact with the actual services running (particularly useful for training employees with technical background, allowing them to gain hands-on experience with systems and issues that are identical to the actual ones).

To achieve this, three variants of a Phishing-focused training programme are provided, with increasing difficulty and complexity; namely a Simple Phishing Programme, a Realistic Phishing Programme and a more difficulty version of the latter; the first two are elaborated in more detail, with a representation of the defined CTTP model for all aspects of the scenario whilst the third is briefly displayed.

5.1.1 Simple Phishing Programme

This is a social engineering scenario that targets trainees with low security expertise. In this simple scenario the user is trained on identifying phishing email attempts. The user logs into the THREAT-ARREST application and is presented with a sequence of emails; their target is to select which of them are legitimate or malicious. The tools involved in this Programme are the Training Tool that facilitates the training aspects and the Simulation tool that accommodates the simulated components. By using the CTTP sub-model the above scenario is specified below and depicted in Figure 18.

The **Training Programme** class includes a description (i.e., Simple Phishing Scenario), a goal (i.e., identify 50 percent of the malicious emails), a type (i.e., preparedness), role (i.e., low privilege end-user), difficulty (i.e. 1).

The **actual trace** that the training programme records includes, system logs, traffic logs and the fake emails that were generated during this scenario.

The **expected trace** that the training programme sets includes the correct and wrong answers of the user.

The **Training Programme Execution** specifies that the only actions allowed to the user is to indicate, via the THREAT-ARREST software and specifically the Training Tool, if an email is legitimate or malicious.

This scenario has one **phase** that involves the presentation of the emails to the user and the identification of the legitimate/malicious emails by the user; when that phase concludes the scenario is completed.

The **CTTP Simulation Model** defines the following, the deployment mode for this scenario is "from scratch"; the tool that is used to implement the simulation is "Jasima"; the simulation timer starts at the time point 0; the simulations ends after 60 mins; the execution speed is 1; the random seed is 1; the messages send during the simulation; specifically the event that starts the simulation, 10 emails with varying states (i.e., legitimate or malicious), and the event that concludes the simulation. The simulation topology consists of a simple module, an email generator application.

The **CTTP Emulation Model** defines that the deployment mode for this scenario is "preset"; the tool that is used to implement the emulation is "OpenStack"; and also provides a path to the preset template *"/path/simplePhishingVM"*. The emulation sub-model consists of one Software-PAL asset (virtual machine) that will contain two Software-SAL assets: 1) PAL - the operating system "Linux" and 2) the simulation software Jasima.

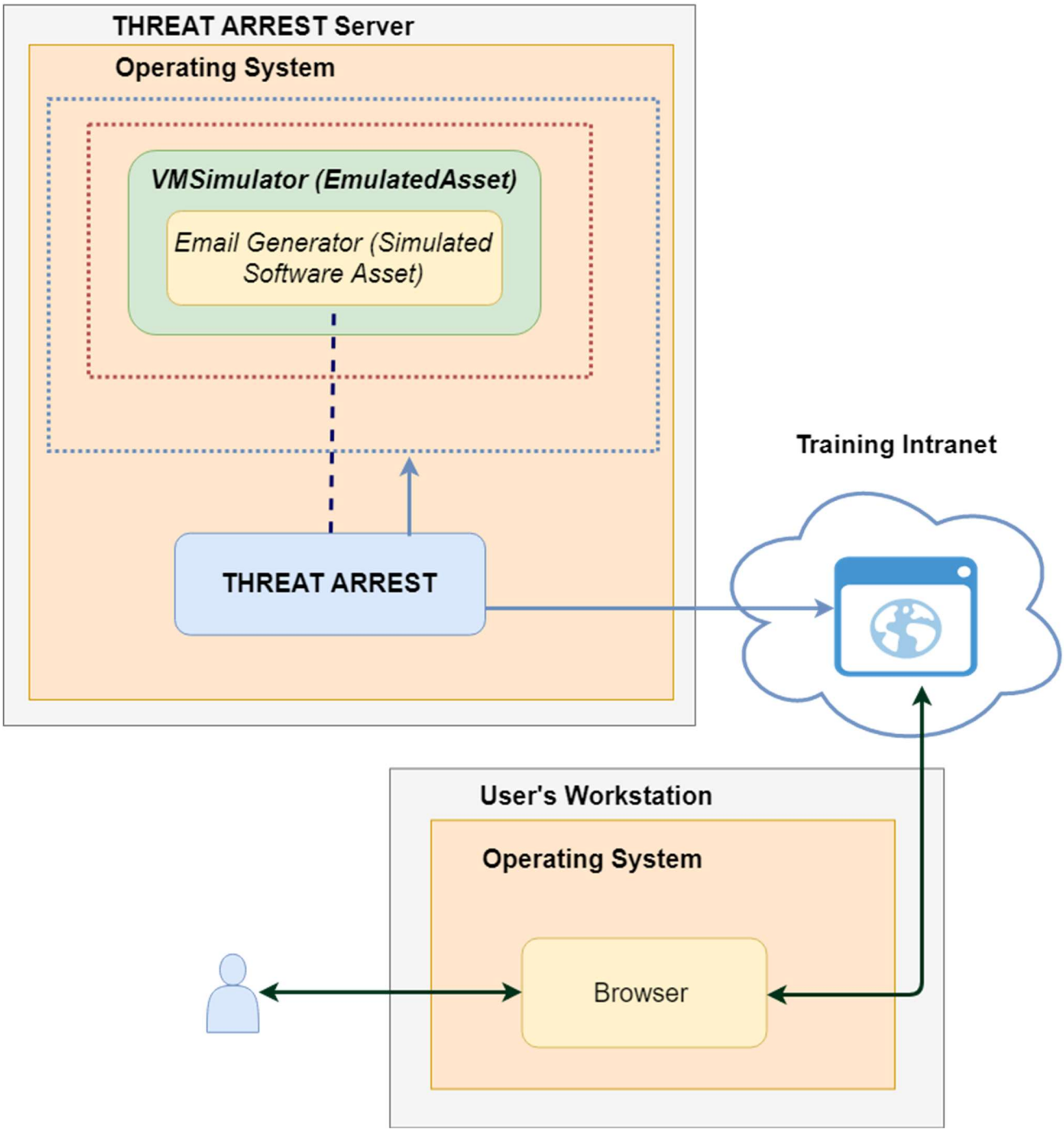

Figure 18 Phishing Training – High level view of the simple variant

5.1.1.1 Representation in the CTTP Model

Figure 19 shows the representation of the above training scenario in the form of a CTTP model, i.e. the model of what the defined scenario would look like within the THREAT-ARREST platform. A full-page view of the same model is provided in Appendix II – Simple Phishing Training Scenario Model Instance.

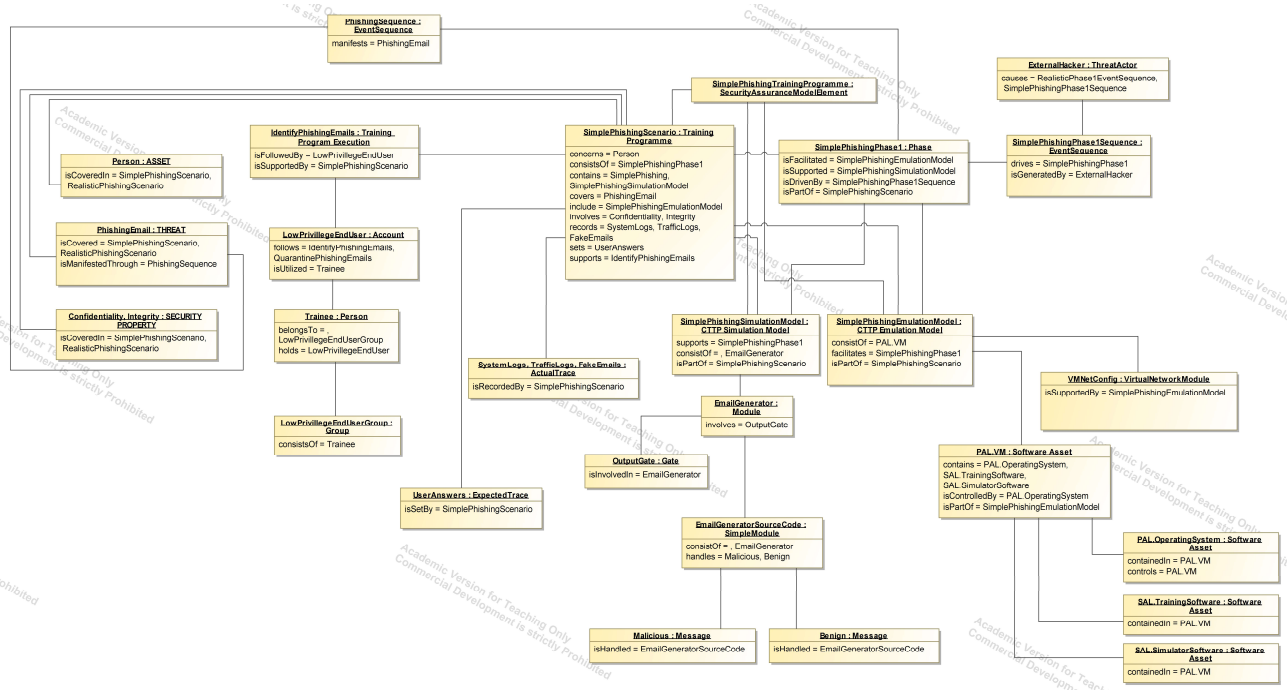

Figure 19 Simple Phishing Training Programme CTTP model instance

5.1.2 Realistic Phishing Programme

This is a social engineering scenario that targets trainees with low-moderate security expertise. In this realistic scenario, the user is trained in identifying phishing emails and quarantining them. The user logs into the THREAT-ARREST application and is presented with a desktop environment. The environment has preinstalled software, an email client, a web browser and security tools (only to be used if the role of the user is 'security engineer'). The email client contains a mailbox file that consists of malicious and legitimate emails. The user will have to go through all these emails, then, identify and quarantine them accordingly. If the user is a security engineer, then an additional phase can be used; in this phase the user will gather information about the malicious emails and their origin.

By using the CTTP training sub-model depicted in Figure 14, the aforementioned scenario is described below and illustrated in Figure 20.

The **Training Programme** class includes a description (i.e., Realistic Phishing Scenario), goals depending on the user role (e.g., quarantine 50 percent of the malicious emails, track the origin of the malicious emails), a type (i.e., preparedness), role (i.e., low privilege end-user/security engineer), difficulty (i.e., 3).

The **actual trace** that the training programme records includes, system logs, traffic logs and the emails that were generated for this scenario.

The **expected trace** that the training programme sets to track are the quarantined emails. The clicks of URLs' contained in the malicious emails are also recorded.

The **Training Programme Execution** specifies the available actions that different users may do. For this scenario the training model can accommodate two different types of users.

a) Low privilege end-user role will be able to use the email client and a web browser with Internet connection.

b) Security Engineer role will be able to use the email client and a web browser with Internet connection. Moreover, the user in this role will be able to use the essential security tool to gather necessary information about the malicious emails.

With regard to **phases**, the training programme will support two phases based on the role of the user.

a) For the low-privileged end-user, this scenario supports 1 phase. In this phase the user must identify the malicious emails and quarantine them.

b) For the security engineer, this scenario supports 2 phases. In the first phase the user must identify the malicious emails and quarantine them; in the second phase the user must gather information of the origin of malicious attempt.

CTTP Simulation Model: N/A (no simulated assets).

The **CTTP Emulation Model** defines that the deployment mode for this scenario is "preset"; the tool that is used to implement the emulation is "Open Stack"; and provides a path to the preset template *"/path/simplePhishingVM"*. The emulation sub-model consists of one Software-PAL asset (virtual machine) that will contain three Software assets: 1) PAL with the operating system "Linux", 2) SAL with the phishing software, and 3) PAL with the email server. The CTTP emulation model also supports a virtual network module that supports the connection between the VM-Trainee and the VM-Attacker. To enable this connection the VM-Trainee has a virtualNetworkAdapter with a unique network information (IP, netmask, etc.). Similarly, the VM-Attacker has a different virtualNetworkAdapter with its network information.

This scenario can be further expanded, in terms of complexity by adding another virtual machine to act as the email server used by the attacker, located in a remote location (see Figure 21). With the simple addition of a supplementary node, the difficulty of the scenario is increased, and the security engineer has to follow a more complex approach to find the origin of the malicious email.

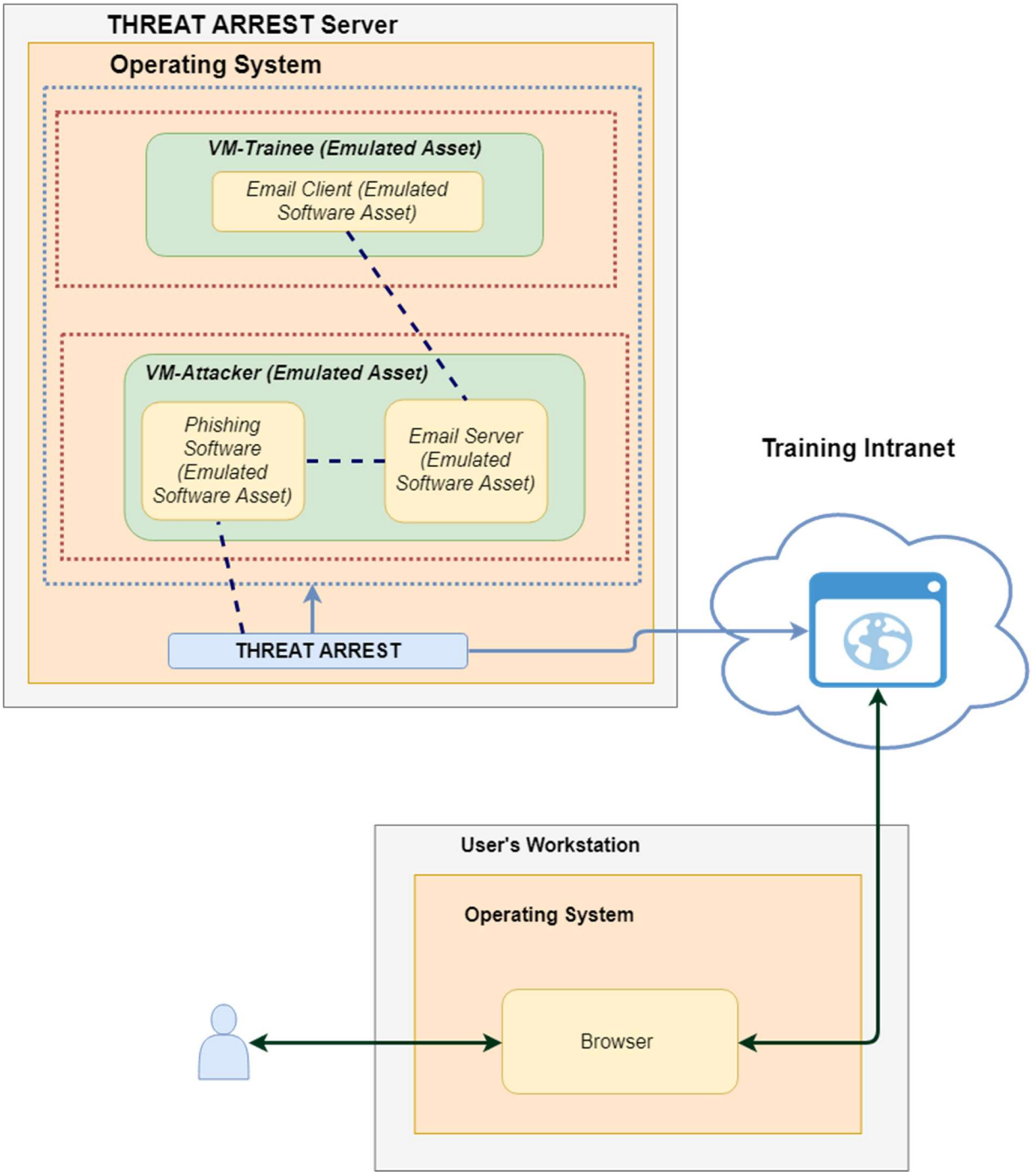

Figure 20 Realistic Phishing Scenario high level view – Easy difficulty

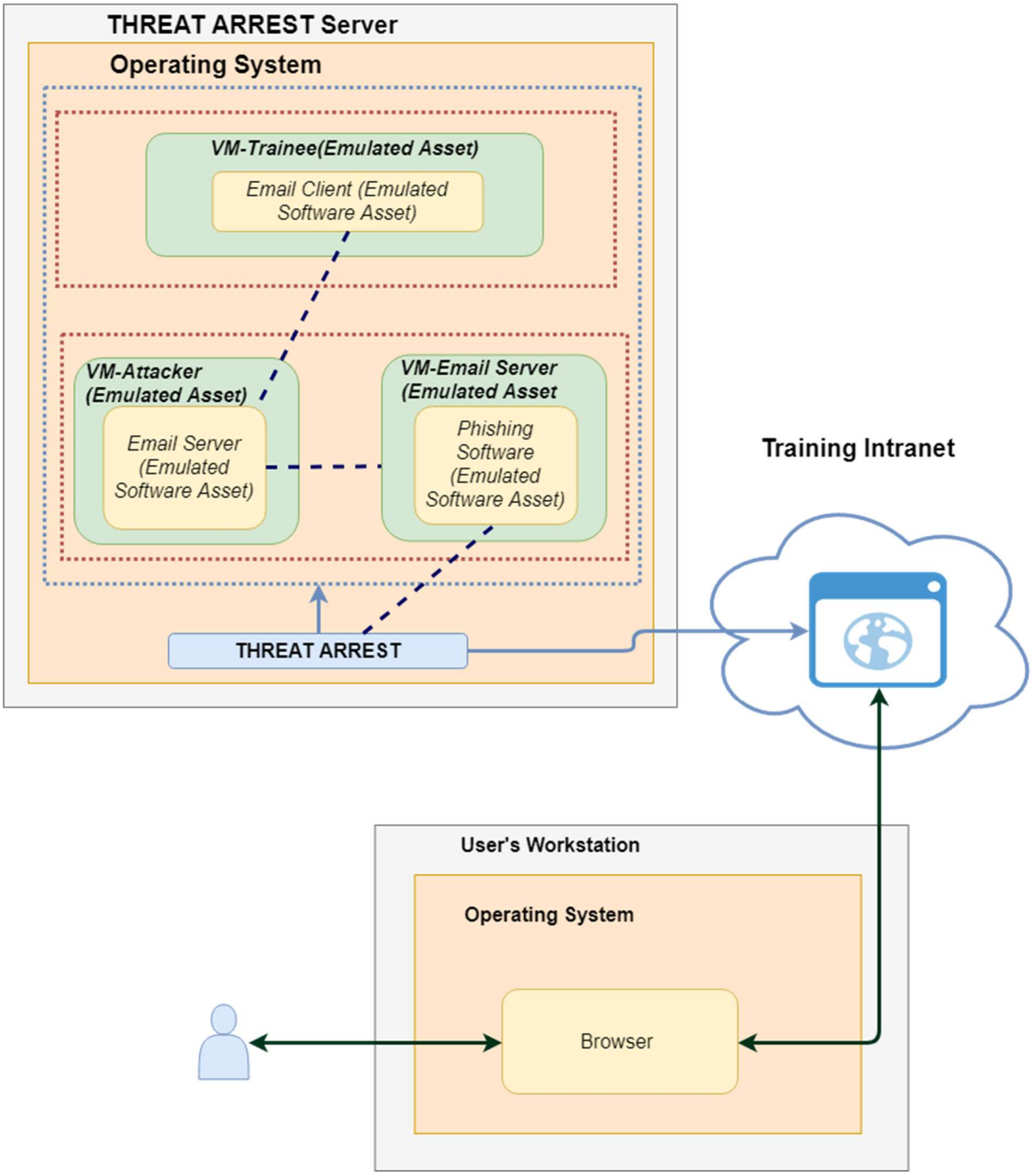

Figure 21 Realistic Phishing Scenario high level view – Medium difficulty

5.1.2.1 Representation in the CTTP Model

Figure 22 shows the representation of the above-mentioned training scenario in the form of a CTTP model, i.e. the model of what the defined scenario would look like within the THREAT-ARREST platform. A full-page view of the same model is provided in Appendix III – Realistic Phishing Training Scenario Model Instance.

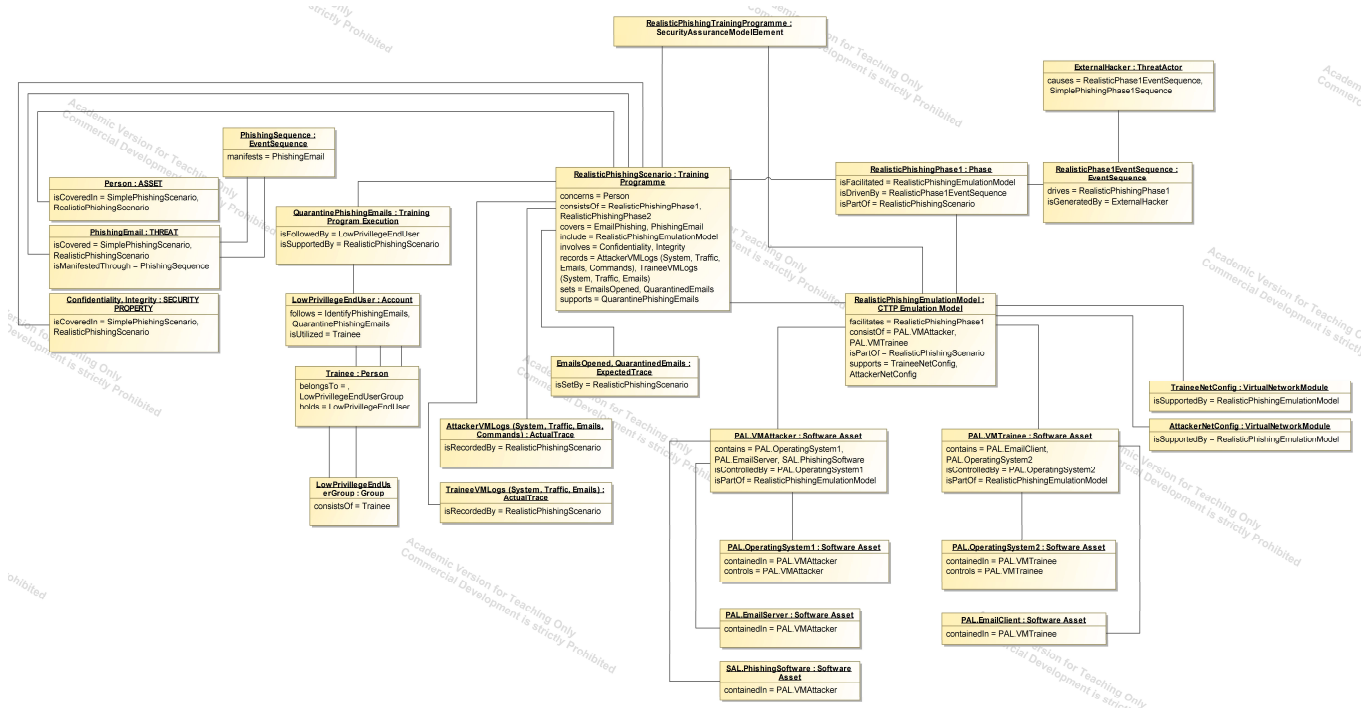

Figure 22 Realistic Phishing Training Programme CTTP model instance

6 Conclusion

This deliverable documented the design and definition of the THREAT-ARREST CTTP Models and Programmes Specification Language. As such, it presented the requirements, as well as the models defined that lead to the specification of the language.

In more detail, the Core Assurance model was presented, covering system assets in their various forms (software, hardware, physical, persons, data, etc.), and the relationship between said assets, threats, and vulnerabilities. Moreover, an extension to the Core Assurance model in the form of a training sub-model (CTTP sub-model) was provided, which was designed to cover all aspects needed to implement a CTTP Training Programme.

Based on these models, a language was derived, with specific language constructs that can be used to define all elements needed for the operation of the THREAT-ARREST training platform. The use of this language to define models will be facilitated by the provision of an associated language editor, which will be provided in the deliverable "D3.2 – CTTP Models and Programmes Specification Tool".

Leveraging the above, CTTP programmes will specify training scenarios, focusing on particular threats, cyber system components and assessment tools of a CTTP model, and will be used to drive (along with the CTTP models) the execution of simulation, emulation, and serious gaming processes that will realize a training programme. An example of this process focused on phishing attacks was also presented, providing three variants with different levels of complexity, allowing the platform to train employees with different levels of expertise.

The results of the use of the primitives defined herein will be documented in the deliverables pertaining to the training programme definition. In more detail, the development of the CTTP models via the language defined herein will take place in the context of task "T3.2 – CTTP Models and Programmes Development" and will be documented in the deliverables "D3.3 – Reference CTTP Models and Programmes Specifications v1" and "D3.5 – Reference CTTP Models and Programmes Specifications v2".

7 References

[1] ANTLR, 2019. *Antlr.org.* [Online]
Available at: https://www.antlr.org/

[2] Enisa.europa.eu, 2019. *Threat Taxonomy.* [Online]
Available at: https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/enisa-threat-landscape/threat-taxonomy/view .

[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] Fysarakis, K., et al., 2014. Embedded systems security challenges. Measurable security for Embedded Computing and Communication Systems (MeSeCCS 2014), within the 4th International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2014), 7-9 January 2014, Lisbon, Portugal, pp. 1-10.

[5] Hatzivasilis, G., et al., 2019a. Review of Security and Privacy for the Internet of Medical Things (IoMT). 1st International Workshop on Smart Circular Economy (SmaCE), Santorini Island, Greece, 30 May 2019, IEEE, pp. 1-8.

[6] Hatzivasilis, G., et al., 2019b. Towards the Insurance of Healthcare Systems. 1st Model-driven Simulation and Training Environments for Cybersecurity (MSTEC), ESORICS, Springer, LNCS, vol. 11981, Luxembourg, 27 September 2019, pp. 1-14.

[7] Jasima, 2019. [Online]
Available at: https://www.simplan.de/en/software-2/jasima/

[8] MagicDraw, 2019. *Nomagic.com.* [Online]
Available at: https://www.nomagic.com/products/magicdraw

[9] Nvd.nist.gov, 2019. *NVD - Data Feeds.* [Online]
Available at: https://nvd.nist.gov/vuln/data-feeds#JSON_FEED
[Accessed 2019].

[10] OpenStack, 2019. *Build the future of Open Infrastructure..* [Online]
Available at: https://www.openstack.org/

[11] Simulator, O. D. E., 2019. [Online]
Available at: http://www.omnetpp.org
[Accessed 2019].

Appendix I – The CTTP sub-model (full view)

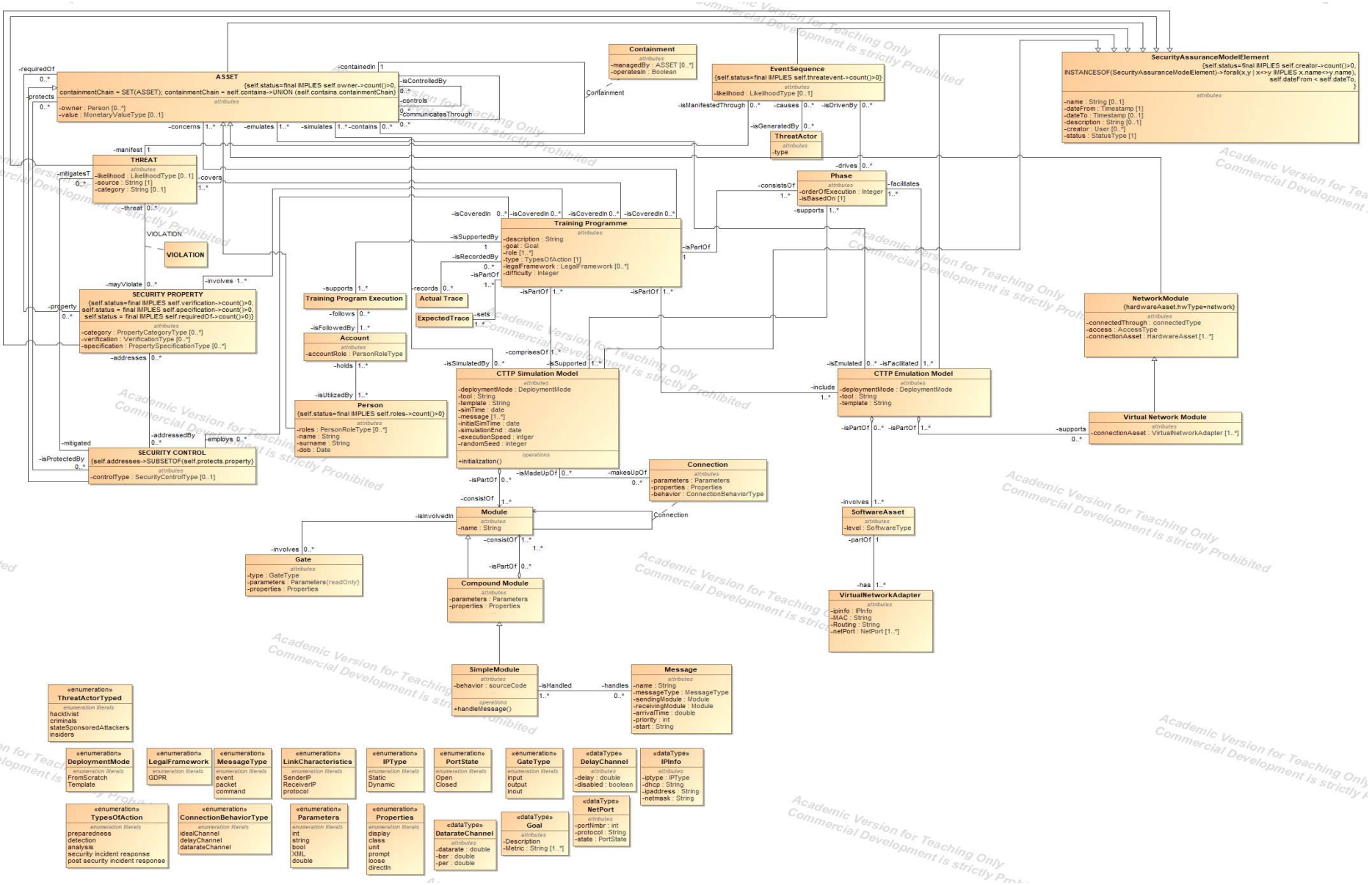

Figure 23 The CTTP sub-model

Appendix II – Simple Phishing Training Scenario Model Instance

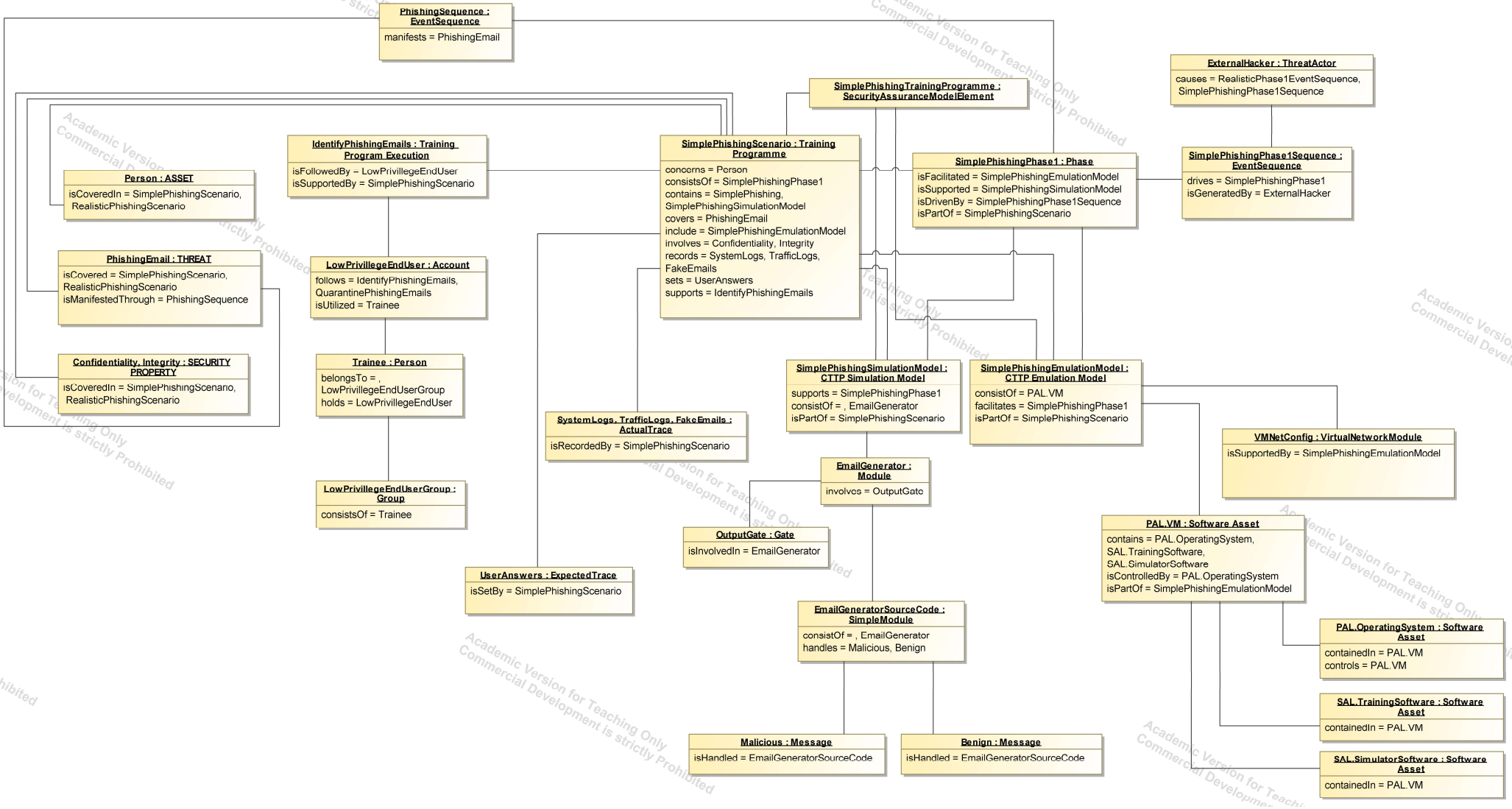

Figure 24 Simple Phishing Training scenario

Appendix III – Realistic Phishing Training Scenario Model Instance

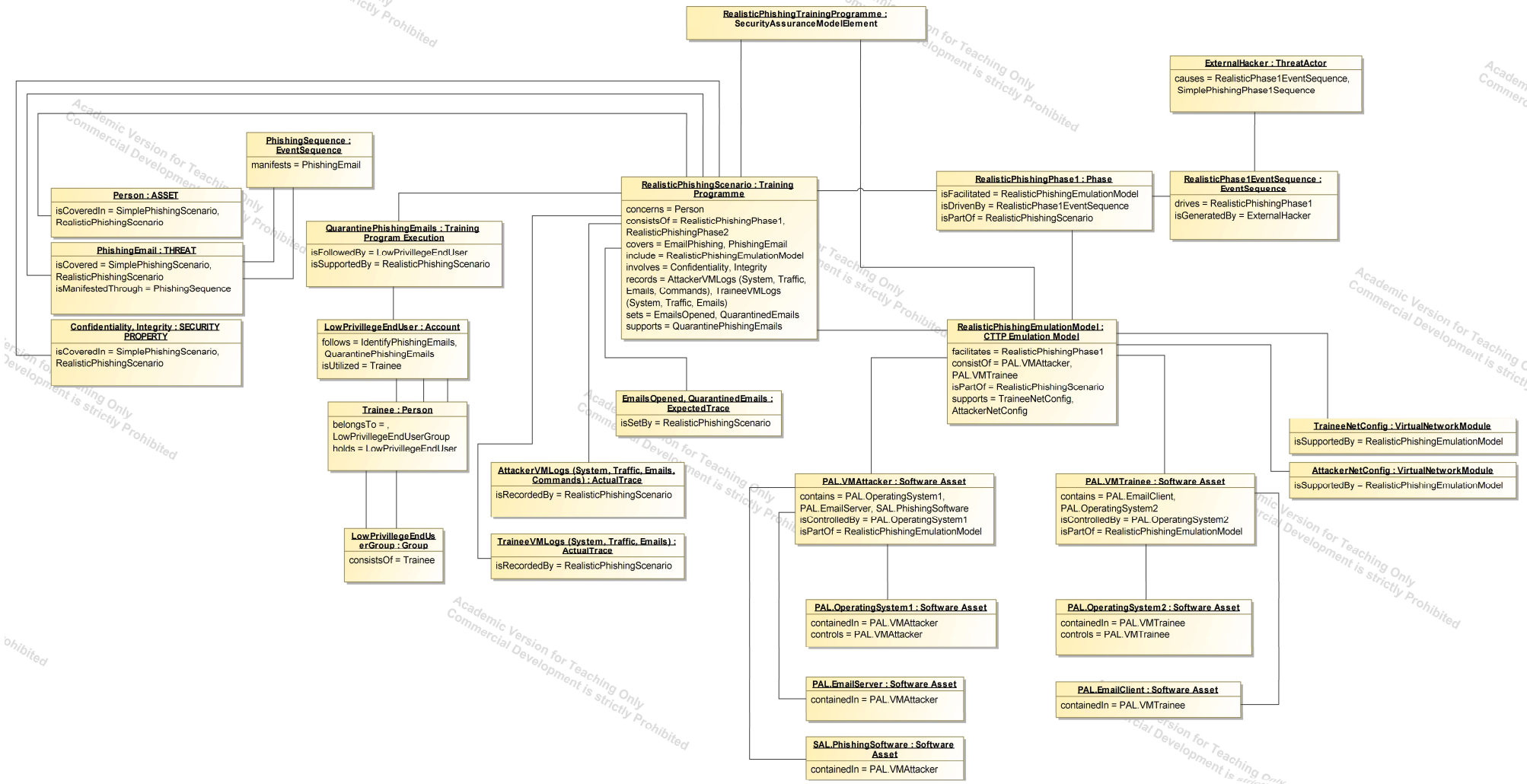

Figure 25 Realistic Phishing training scenario