Advanced Cyber-Threat Intelligence, Detection, and Mitigation Platform for a Trusted Internet of Things
Grant Agreement: 786698

D5.3 - CYBER-TRUST proactive technology tools

Work Package 5: Key proactive technologies cyber-threat intelligence

Document Dissemination Level

P	Public	X
CO	Confidential, only for members of the Consortium (including the Commission Services)	

Document Due Date: 31/01/2020
Document Submission Date: 31/01/2020

Co-funded by the Horizon 2020 Framework Programme of the European Union

Deliverable Information Sheet

Document Information

Deliverable number:	**D5.3**
Deliverable title:	CYBER-TRUST proactive technology tools
Deliverable version:	V1.0
Work Package number:	WP5
Work Package title:	Key proactive technologies and cyber-threat intelligence
Due Date of delivery:	31/01/2020
Actual date of delivery:	31/01/2020
Dissemination level:	Public
Editor(s):	Stefano Cuomo, Simone Naldini (Mathema)
Contributor(s):	Nicholas Kolokotronis, Costas Vassilakis, Christos Tryfonopoulos, Konstantinos–Panagiotis Grammatikakis, Ioannis Koyfos, George Pikramenos, Christos–Minas Mathas, Paris Koloveas (UOP)
Reviewer(s):	Clément Pavué (SCORECHAIN) Gohar Sargsyan, Raymond Binnedijk (CGI)
Project name:	Advanced Cyber-Threat Intelligence, Detection, and Mitigation Platform for a Trusted Internet of Things
Project Acronym	Cyber-Trust
Project starting date:	1/5/2018
Project duration:	36 months
Rights:	Cyber-Trust Consortium

Version History

Version	Date	Beneficiary	Description
0.1	25/11/2019	Mathema, UOP	Table of contents and distribution of work
0.2	06/12/2019	Mathema, UOP	Table of contents has been finalised
0.3	20/12/2019	UOP	The eVDB short description is provided
0.4	06/01/2020	UOP	The crawler component has been described
0.6	10/01/2020	UOP	The TMS component has been described
0.8	17/01/2020	UOP	The iIRS component has been described
0.9	20/01/2020	Mathema, UOP	Consolidation of the document
1.0	28/01/2020	Mathema	Version ready for review
1.1 Final	31/01/202	Mathema	Final Version

Acronyms

ACRONYM	EXPLANATION
A	Actor
AMPQ	Advanced Message Queuing Protocol
APIs	Application Programming Interface
AS	Autonomous System
CIDR	Classes Inter-Domain Routing
CT	Cyber-Trust
CTI	Cyber-Threat Intelligence
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerabilities Scoring System
D	Deliverable
DB	Database
eVDB	Enriched Vulnerability Database
FR	Functional Requirement
GUI	Graphic User Interface
ID	Identification
IDS	Intrusion Detection System
iRC	iIRS Client
iRE	iIRS Decision-making Engine
iRG	iIRS Attack Graph Generator
iIRS	Intelligent Intrusion Response
IoC	Indicator of Compromise
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
JSON	JavaScript Object Notation
LCPD	Local Conditional Probability Distribution
M	Month
MISP	Malware Information Sharing Platform
NFR	Non-Functional Requirement
REST	Representational State Transfer
SQL	Structured Query Language
T	Task

TMS	Trust Management Service
UI	User Interface
URL	Uniform Resource Locator
VM	Virtual Machine
WP	Work Package
XML	Extensible Markup Language

Executive summary

This report is a contractual deliverable within the Horizon 2020 Project Cyber-Trust: Advanced Cyber-Threat Intelligence, Detection, and Mitigation Platform for a Trusted Internet of Things. It provides detailed description of the results of the task T5.4 “Cyber-Trust Proactive technology tools” related to the implementation of the tools of work package and according to the platform architecture as described in D4.4 [3].

This document provide a a technical documentation of the prototype implementing the algorithms and methods related to the key technology used for the pre-reconnaissance cyber-threat intelligence. The main tools here described are Crawling service module, the Enriched vulnerability database (EVDB), the Trust management service and the Intelligent Intrusion Response (iIRS) which aim at improving the security of the Cyber-Trust platform through the collection and aggregation of data and information from multiple sources. The ultimate goal of the presented tools is to make the IoT devices network safer by preventing cyber-attacks whenever possible, and aiming to mitigate the effects of unpredictable attacks.

Further to a general overview or each of these tools detailed information are given about technical details such as, for instance, Functionality Coverage, Application and Physical Architecture, Application programming interfaces and Technology stack. A final chapter present the unit test approach is present to verify that the individual artefacts comprising the software component operate as expected.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

1.1 Purpose of the document

The main objective of this deliverable is to provide a technical documentation of the prototype implementing the algorithms and methods related to the key technology used for the pre-reconnaissance cyber-threat intelligence.

In particular, the content of the deliverable includes details about the Crawling service module, the Enriched vulnerability database (EVDB), the Trust management service and the Intelligent Intrusion Response (iIRS).

The main objective of these tools is to improve the security of the Cyber Trust platform, through the collection and aggregation of multiple data and information from different sources.

Deepnet web forums or marketplaces and clearnet social platforms can be identified among these sources. The collection of information is aimed at identifying and previously mitigating threats to IoT devices.

The methods available to accomplish this goal are the search (supervised and unsupervised) of social networks, forums or marketplaces that may contain information regarding possible threats; the use of classification and characterization methods to evaluate the threats found, and the ranking of the identified threats.

The ultimate goal is, through further data processing, and a careful evaluation of the methods of presenting the results, to make the IoT devices network safer by preventing cyber-attacks whenever possible and aiming to mitigate the effects of unpredictable attacks.

1.2 Relations to other activities in the project

The deliverable is mainly linked to task 5.1 (Threat intelligence techniques) 5.2 (Trust establishment and risk assessment) and which are summarized in deliverable 5.1 (State-of-the-art on proactive technologies). Given the importance of the graphic representation and the proposal to the user, the deliverable is also linked to D6.3 (Cyber-Trust Network tools) and D6.4 (Cyber-Trust visualization tool).

1.3 Structure of the document

The document is structured in order to describe the components related to the key proactive technologies used in the Cyber-Trust platform. In particular:

1. Crawling service
2. Enriched Vulnerability DataBase (EVDB)
3. Trust Management Service
4. Intelligent Intrusion Response

For each of these components, information will be given about:

- A general overview
- Functionality Coverage
- Application Architecture
- Application programming interfaces
- Technology stack

- Physical Architecture
- User Interface (where relevant)

A chapter about the unit test approach is present to verify that the individual artefacts comprising the software component operate as expected.

2. Crawling service

2.1 Overview / objectives

The Crawling Service component lies at the core of the cyber-threat intelligence gathering envisioned by Cyber-Trust. It is responsible for:

- Collecting public cyber-threat intelligence information from the social/clear/deep/dark web, including related forums, marketplaces and security-related websites.
- leveraging the collected information to identify emerging threats, zero-day vulnerabilities and new exploits to IoT devices.
- Making the leveraged information available to the rest of the Cyber-Trust platform by storing it in the eVDB.

To do so it utilizes an ensemble of state-of-the-art data processing and machine learning techniques to identify the web pages that should be crawled and to extract/contextualize all relevant threat information. The Crawling Service also offers a user interface, through which the crawling process can be supervised, managed and tuned. It interacts only with the eVDB Sharing Service, which is used for storing and sharing of the actionable intelligence that has been discovered.

2.2 Functionality coverage

2.2.1 Related requirements

Error! Reference source not found. lists the functional requirements related to the Crawling Service and the provisions made by the component to support the fulfilment of these requirements.

Table 2-1: Functional requirements and use-case references

REF_ID	Description of implementation	Use Case
FR78	**Requirement:** Specific user (based on access role) will be able to configure crawler's parameters ("Tune crawling" functionality). **Implementation:** The crawler accepts parameter modification either as standalone user input or via modifying appropriate setup files that are subsequently accessed by the component. The user input may be provided via a dedicated GUI, or by direct access to the appropriate files. Access control is performed by another component (A06).	UCG-16-05 UCG-19-04
FR84	**Requirement:** The user (based on access role) will supervise the cyber-threat discovery in order to add new (after proper evaluation), update existing and approve crawling new seeds. **Implementation:** The crawler uses a machine-learning model to extract features from the submitted seeds and create an appropriate model that is used to guide the crawling. Addition of new seeds and/or modification of existing ones causes an update to the crawler model and may thus be used to direct the focused crawl. The user input may be provided via a dedicated GUI,	UCG-19-04

	or by direct access to the appropriate seed files. Access control is performed by another component (A06).	

Table 2-2 lists the non-functional requirements related to the Crawling Service and the provisions made by the component to support the fulfilment of these requirements.

Table 2-2. Non-functional requirements and use-case references

REF_ID	Description of implementation	Use Case
NFR26	**Requirement:** The platform must have "Rate seeds" functionality in order for the respective user to rate the crawling seeds. **Implementation:** Seeds may be added or removed as appropriate to modify the machine-learning model that is used to guide the crawling; these operations provide a seed rating mechanism that reinforces the crawling model accordingly.	UCG-06-06
NFR44	**Requirement:** The crawler will be able to crawl the clear, deep and dark web. **Implementation:** A common crawler infrastructure is used for accessing clear, deep and dark web; specialised components are utilised for TOR proxying, authorization management, form interaction, and other more specialised tasks required.	UCG-16-05
NFR45	**Requirement:** The crawler will continuously crawl popular social media streams, popular security-related websites and deep/dark web forums and marketplaces. **Implementation:** The crawler frontier is maintained in-memory for efficiency reasons and is enriched by adding new URLs as it visits new websites; periodically the frontier is persisted to ensure fault-tolerance and assert continuous operation. Focused versions of the crawler are meant for continuous exploration of the web, while in-depth versions may be launched to harvest content (by resorting to link filtering) from the social, deep and dark web.	UCG-16-05

2.2.2 Related use cases

Table 2-3 lists the use cases related to the Crawling Service and the provisions made by the component to support the fulfilment them.

Table 2-3. Use-cases related to the crawling service (A10)

REF_ID	Description of implementation
UCG-06-06	**Use case:** Provide feedback/rating on sources of vulnerabilities

	Implementation: Seeds may be added or removed as appropriate (by expert users) to modify the machine-learning model that is used to guide the crawling; these operations provide a seed rating mechanism that reinforces the crawling model accordingly.
UCG-16-05	**Use case:** Crawl the clear/deep/dark web and update the eVDB **Implementation:** The crawling service continuously crawls popular social media streams, popular security-related websites and deep/dark web forums and marketplaces. Cyber-threat information on 0-day vulnerabilities, exploits, signatures, executables, and other related information is sought. The collected data use appropriate references to eVDB objects to update the eVDB component.
UCG-19-04	**Use case:** Tune the crawling parameters and evaluate existing seeds **Implementation:** The expert user supervising the crawling service can add, annotate, and approve the crawling of new seeds (i.e., websites of interest), tune the parameters that enable their discovery and adjust crawler's performance, and evaluate the existing seeds in terms of usefulness.

2.3 Technology update

The Cyber-Trust Crawling Service extends the current paradigms and implementations in a variety of domains including thematical and focused crawling, post classification, natural language understanding, and entity extraction by considering additional dimensions (e.g., multi-stage post classification) and functionality (e.g., integrated thematic and focused crawling). Moreover, the integrated services that are offered and their seamless orchestration, create a novel framework that is able to fully support the cyber-threat intelligence lifecycle through:

- thematic and in-depth crawling of relevant sites in the social/clear/deep/dark web driven by advanced machine learning models to direct the crawl for higher efficiency,
- state-of-the-art classification of collected pages that works in tune with the thematic crawling for higher effectiveness,
- highly scalable, modern, NoSQL solution for storage of all relevant data,
- a combination of rule- and machine learning-based natural language understanding for leveraging the collected data to information.

2.4 Application architecture

The crawler architecture (illustrated in **Error! Reference source not found.**) consists of three major components:

- the *crawling module* (blue part),
- the content ranking module (red part), and
- the information extraction module (purple part).

The proposed architecture has been entirely designed on and developed using open-source software; it employs an open-source focused crawler, an open source implementation of word embeddings for the latent topic modelling, open-source NoSQL database storage for all persistent data, and an open-source natural language understanding engine.

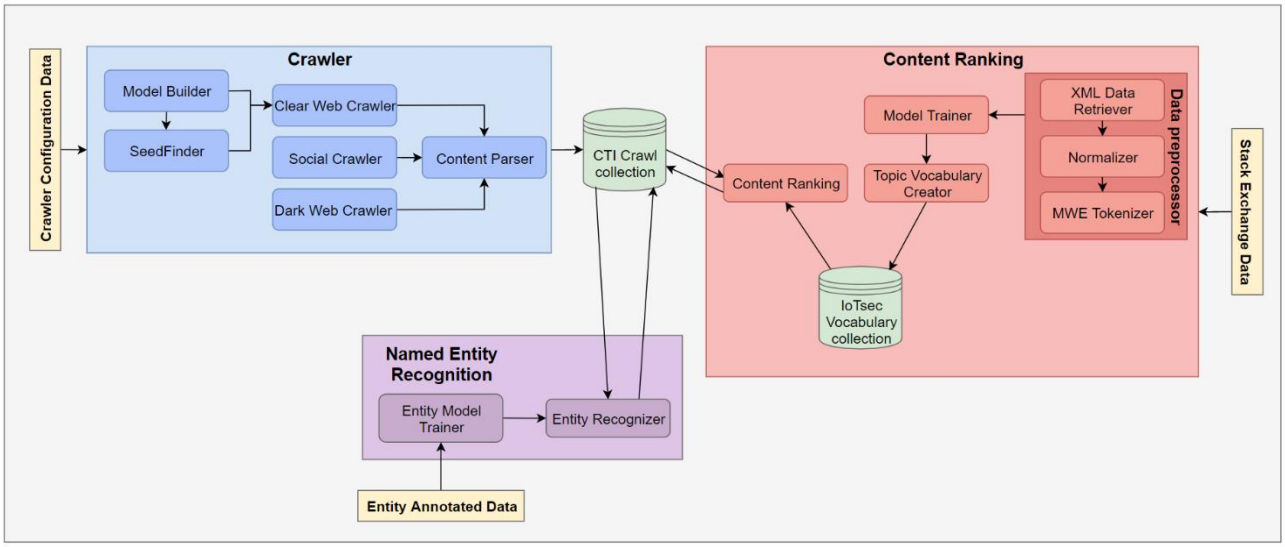

Figure 2-1: Crawling Service architecture (including details on the different modules)

The idea behind this modular architecture and a multi-stage framework approach is attributed to the openness of the topic at hand and is briefly outlined below.

- The crawling module harvests content from a variety of CTI sources in the clear, social, and deep/dark web by employing a thematically focused crawler to direct the crawl towards websites of interest to the CTI gathering task. This is realized by resorting to a combination of machine learning techniques (for open domain crawl) and regex-based link filtering (for structured domains like forums).
- The harvested content is stored in an efficient NoSQL datastore and is retrieved for further inspection in order to decide its usefulness to the task. This is achieved by employing statistical language modelling techniques to represent all information in a latent low-dimensional feature space and a ranking-based approach to the collected content (i.e., rank it according to its potential to be useful). These techniques allow us to train our language model to
    - capture and exploit the most salient words for the given task by building upon user conversations
    - compute the semantic relatedness between the crawled content and the task at hand by leveraging the identified salient words, and
    - classify the crawled content according to its relevance/usefulness based on its semantic similarity to CTI gathering.

Notice that the post-mortem inspection of the crawled content is necessary, since the thematically focused crawl is forced to make a crude decision on the link relevance (and if it should be visited or not) since it resorts on a limited feature space (e.g., alt-text of the link, words in the URL, or relevance of the parent page).

- The identified relevant content is then analysed using advanced natural language understanding methods to perform named entity recognition for entities that are of interest (i.e., cyber-threat intelligence). These methods employ sets of annotated entity data such as malware names, product names, CVEs, etc., to facilitate named entity recognition, entity/concept linking and open information extraction.

2.5 Application programming interfaces

The ACHE Crawler exposes a REST API (see Table 2-4), which can be used to perform operations on active crawls, extract relevant metrics and monitor the progress of each crawl in real time.

Table 2-4. REST APIs for managing the ACHE crawler

API URL specification	Description	Input Variables
POST /crawls/{crawler_id}/startCrawl	Create and start a new crawler.	**CrawlType:** The type of crawl **Seeds:** A list of seed URLs
POST /crawls/{crawler_id}/seeds	Add seeds to an existing selected classifier.	**Seeds:** A list of seed URLs
GET /crawls/{crawler_id}/status	Returns the status of the selected crawler.	-
GET /crawls/{crawler_id}/metrics	Returns the metrics of the selected crawler. Part of those metrics can be seen in Figure XX.	-
GET /crawls/{crawler_id}/stopCrawl	Stops the selected crawler.	-

2.6 Technology Stack

2.6.1 ACHE Crawler

ACHE is a focused web crawler that harvests web pages satisfying specific criteria; it differs from generic crawlers due to the use of page classifiers that allow it to distinguish between relevant and irrelevant pages. Page classifiers may be regular expressions or a machine-learning-based classification models and allow ACHE to prioritize links in order to efficiently locate relevant content while avoiding the retrieval of irrelevant pages.

2.6.2 MongoDB, MongoExpress

MongoDB is a general-purpose NoSQL document database that stores data in JSON-like documents; this design provides implementation simplicity, model expressivity and a natural adaptation to the data at hand over the typical row/column model. MongoExpress is a web-based MongoDB admin interface where we can explore our stored data and perform actions such as *simple* and *advanced querying*, *deleting*, *sorting* and *editing* each individual document, etc.

2.6.3 Gensim Toolkit - Word2Vec

A set of language modeling and feature learning techniques in natural language processing where words or phrases from the vocabulary are mapped to vectors of real numbers; they essentially refer to distributed representations of text in an n-dimensional space. This is a popular domain adaptation technology that allows machine learning models to map niche datasets that are all written in the same language but are still linguistically different.

2.6.4 Privoxy (TOR proxying)

Privoxy is a non-caching web proxy with advanced filtering capabilities for enhancing privacy, modifying web page data and HTTP headers, controlling access, and removing ads and other obnoxious Internet junk. Privoxy has a flexible configuration and can be customized to suit individual needs and tastes. It has application for both stand-alone systems and multi-user networks.

2.6.5 SpaCy (Named Entity Recognition)

Named-entity recognition (or entity identification, entity chunking, entity extraction) is a subtask of information extraction that seeks to locate and classify named entity mentions in unstructured text into pre-defined categories. For our purpose such categories may include organizations, malware, exploits, IP locations, temporal expressions, monetary values (usually in bitcoin), and others.

2.6.6 Python 3.6

Python is a widely used high-level programming language, mainly used for data manipulation and analytics tasks. Notable Python libraries used for the implementation are: Numpy, NLTK, Gensim, BS4 (BeautifulSoup), Newspaper3k, Boilerpipe3 and SpaCy.

2.6.7 Docker

Docker is a tool designed to create, deploy, and run applications in the form of containers. With containers we can package up an application with all the parts it needs, such as libraries and other dependencies, and ship it all out as one package. With this form of deployment, the application can run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

2.7 Physical architecture

The physical architecture of the Crawling Service is comprised of several Docker containers, orchestrated by a Bash script. Specifically, there are different docker containers for:

- the crawler module
- the content parser sub-module
- the content ranking module
- the named entity recognition module

The number of crawler containers may vary, depending on the number of website-specific crawlers that have been created. The containers are activated in the order that they were presented above, with a varying time delay between each activation, which allows the previous container to sufficiently complete its function.

2.8 User Interface

The ACHE Crawler admits a monitoring dashboard, a snapshot of which can be seen in Figure 2-2, to allow the user to monitor the progress of each crawl in real time.

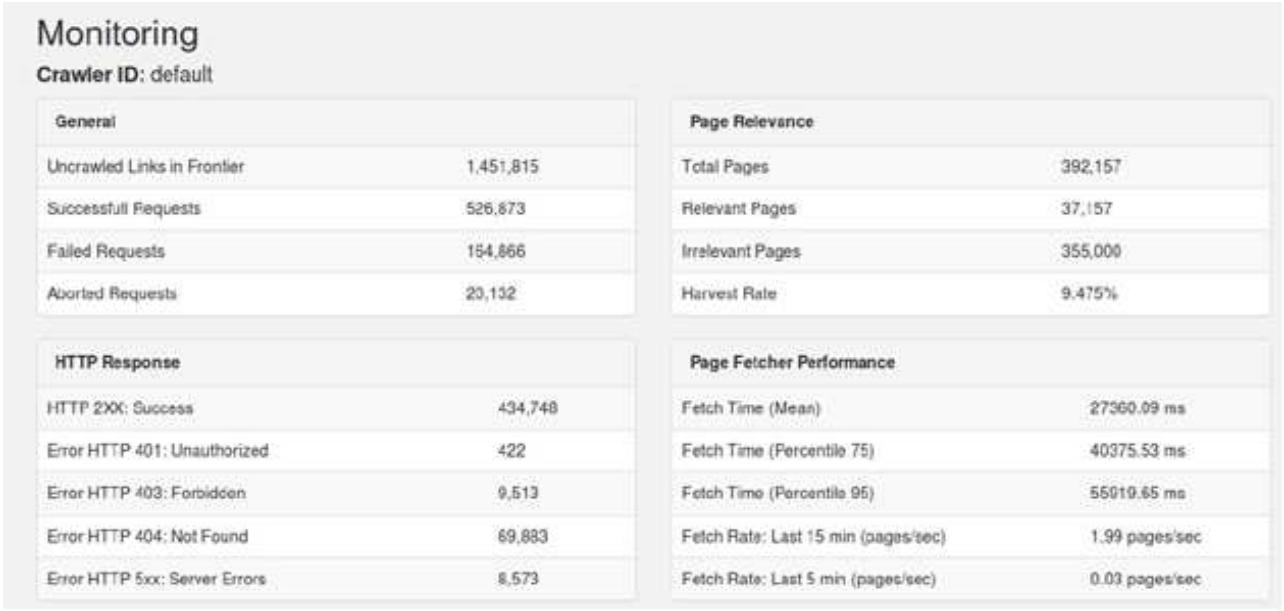

Figure 2-2. Monitoring of an ACHE crawl (visualization of the REST API)

The display of data from the Crawling service is proposed to Cyber-Trust users through the UI (see Figure 2-3), using ad hoc charts and graphs, aimed at improving the readability of the data.

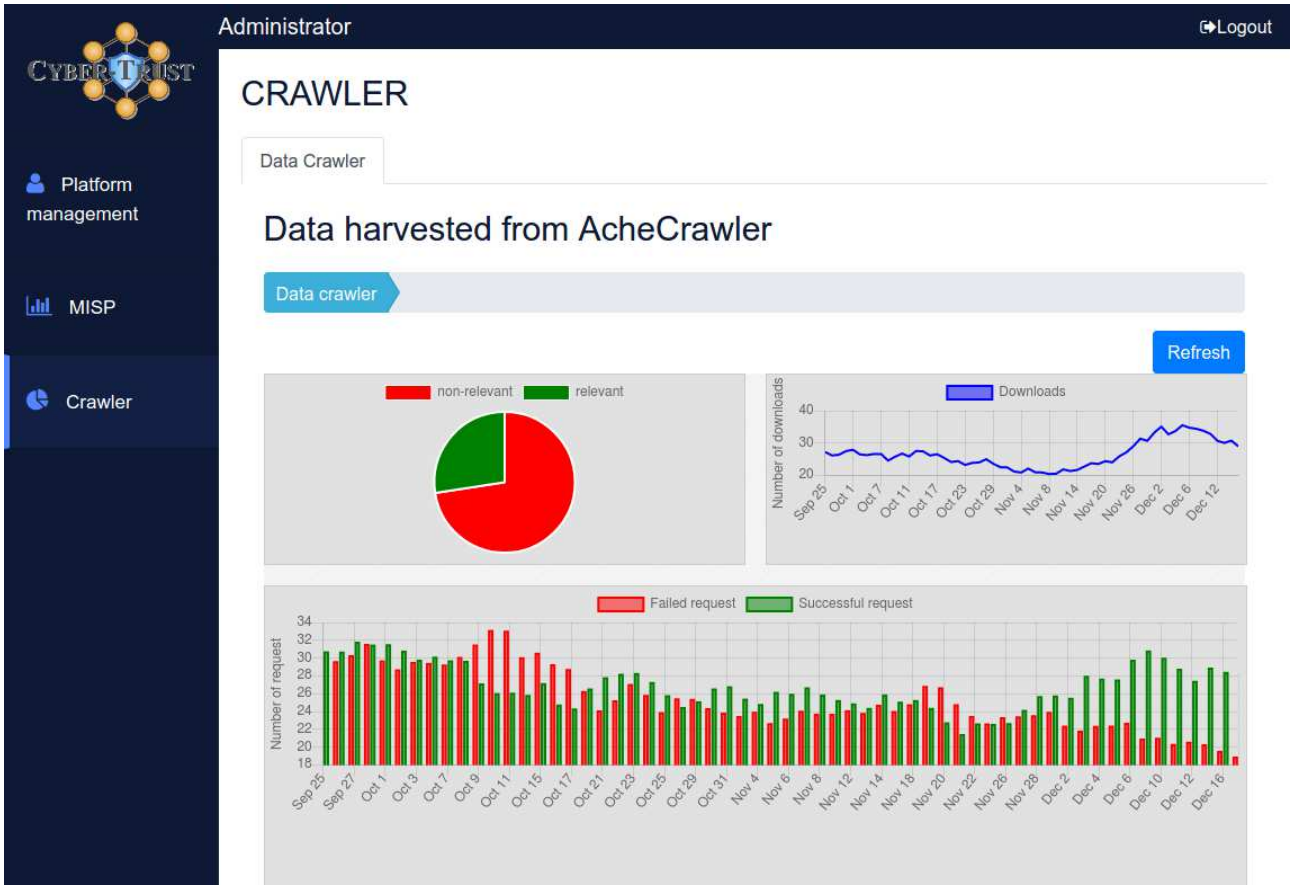

Figure 2-3. Visualization of the crawler in Cyber-Trust platform.

3. Enriched vulnerability database

The *Enriched Vulnerability Database* (eVDB) is a core component of Cyber-Trust (CT) platform that is actually comprised of two parts: the eVDB admin module [A07] and the sharing service [A09]. The eVDB admin module is responsible for the usage and maintenance of the database that stores enriched data about the vulnerabilities, exploits, etc., that are collected through CTI techniques [15]. The eVDB sharing services link the eVDB to Cyber-Trust registration portal [A06] and in principle with the rest of the components that require up to date information about cyber-threat intelligence. It also enables the dissemination of results and information regarding vulnerabilities, exploits, cyber-attacks, etc. with affiliate members and individuals.

The implementation details of the eVDB component, namely the functionality coverage (incl. requirements and use cases), technology update, application architecture, application programming interfaces, technology stack, physical architecture, and user interface, are described in detail in deliverable D5.2.

4. Trust management service

4.1 Overview / objectives

The objective of the trust management service [A05, A08] is to serve an authority within the Cyber-Trust architecture which undertakes the following tasks:

- Consolidates observations on the status, behaviour and associated risk of devices into a comprehensive trust score, which indicates the degree to which each device is deemed to be trustworthy.
- Can be queried by other Cyber-Trust entities to provide the abovementioned assessments, for the perusal of the entities. Indicatively, trust assessments can be used for the visualization of trust within the network, for making decisions whether actions originating from or being directed to some device should be allowed or not, for raising alerts to security officers and so forth.
- Provides timely notifications to other entities within the Cyber-Trust platform to alert them of noteworthy events related to the level of trust associated with devices. In particular, demotions of device trust level below some threshold and the restoration of previously demoted trust of devices are emitted, allowing relevant components of the Cyber-Trust platform to take appropriate actions, such as enabling or disabling defense mechanisms.

4.2 Functionality coverage

4.2.1 Related requirements

The TMS is involved in a number of scenarios of the Cyber-Trust platform, where the trust level of one or more devices needs to be reassessed or consulted. In more detail, the TMS is involved in the following scenarios:

- *Monitoring and vulnerability assessment:* when a device is found to deviate from normal behavior (or return to it after a period of deviation), or be vulnerable to new threats, the TMS triggers the recomputation of the device's trust level.
- *Network-level attacks*: when a network-level attack is identified, the TMS exploits the information provided by the iIRS to adjust the trust value of involved devices.
- *Device-level attacks*: Similarly, when a device is involved in some attack, the TMS arranges for recomputing the trust level associated with the device.

These user scenarios have co-shaped a number of functional and non-functional requirements. The relevant functional requirements are described in Table 4-1, while the associated non-functional requirements are described in Table 4-2.

Table 4-1. Functional requirements and use-case references for the TMS

REF_ID	Description of implementation	Use Case
FR9	**Requirement:** Every device connected to the Cyber-Trust platform has visual representation of the Trust level (scoring) before the identification of abnormal behavior (e.g. cyber-attack)	UCG-05-07, UCG-05-05

	Implementation: The TMS underpins this requirement by providing the trust level of the device to the visualization module.	
FR10	**Requirement:** Every device connected to the Cyber-Trust platform has visual representation of the Trust level (scoring) during abnormal behavior (e.g. cyber-attack) **Implementation:** The TMS underpins this requirement by providing the trust level of the device to the visualization module. The trust assessment is updated synchronously as new data are received by the TMS, therefore the visualization will reflect the evolution of the trust level.	UCG-05-07, UCG-05-05
FR11	**Requirement:** Every device connected to the Cyber-Trust platform has visual representation of the Trust level (scoring) after the mitigation of any abnormal behavior (e.g. cyber-attack). The TMS underpins this requirement by providing the trust level of the device to the visualization module. The trust assessment is updated synchronously as new data are received by the TMS, therefore the visualization will reflect the evolution of the trust level. **Implementation:** The TMS underpins this requirement by providing the trust level of the device to the visualization module.	UCG-05-07, UCG-05-05
FR21	**Requirement:** The user will be informed for the importance of the alert based on the overall Score of the device (it will be derived based on the abnormal behavior, detected vulnerabilities etc.) **Implementation:** The TMS sends notifications when the trust level of device is demoted beyond a certain threshold or restored. These notifications may be exploited by other components, notably visualization and user notification modules, to appropriately convey the information to the user.	UCG-06-01, UCG-06-02, UCG-13-01, UCG-16-03
FR69	**Requirement:** The administrator (Trust DB) will be able to update the Trust score of a device manually. The update will include at least three options: Change status, Delete, Take offline. Field for additional information will be provided (e.g. comments). **Implementation:** A relevant API is provided, allowing authorized users to explicitly set the trust level of the device. Explicitly set trust levels are not directly modified by the trust score update procedure, however major discrepancies between explicitly set and computed scores will raise alerts.	UCG-10-04
FR73	**Requirement:** The user will be able to request (through the UI) the trust level of specific device(s) **Implementation:** The TMS provides an API through which authorized entities can retrieve the trust score of a device.	UCG-13-01
UP_FR8	**Requirement:** For each device users are going to visualise the reason for a certain Trust Level Score. **Implementation:** The TMS API will return, upon request, the base data that contributed to the shaping of the reported trust level.	UCG-13-01, UCG13-02

Table 4-2. Non-functional requirements and use-case references for the TMS

REF_ID	Description of implementation	Use Case
NFR43	**Requirement:** Prioritization of cyber-threats: the threats are ordered in descending order of their score. The score will derive based on vulnerability and impact attributes (technical impact, exploitability etc.) **Implementation:** Stems directly from the implementation of the use case.	UCG-16-04
NFR21	**Requirement:** Creation of the Trust DB **Implementation:** Instructions and/or automations for creating the TrustDB will be provided.	-
NFR22	**Requirement:** Trust DB will store records only hashed data **Implementation:** Data that are primarily stored in other databases will be maintained as hashes with relevant pointers.	UCG-04-01

4.2.2 Related use cases

Table 4-3 lists the use cases related to the TMS and the provisions made by the component to support the fulfilment them.

Table 4-3. Use-cases related to the TMS

REF_ID	Description of implementation
UCG-10-05	**Use case:** Manually curate device profile **Implementation:** The TMS provides an API through which device trust scores can be explicitly set.
UCG-13-01	**Use case:** Retrieve trust level from TMS **Implementation:** Trust levels are computed by the TMS as relevant events occur and stored in the trust database. The trust database realizes an API through which authorized entities can retrieve the trust level assessments, either for a single device or for a bulk of devices.
UCG-13-02	**Use case:** Compute device trust level **Implementation:** The TMS intercepts notifications sent by other Cyber-Trust platform components, and exploits the information therein to compute the trust level. The notifications are received through the Cyber-Trust system message bus.
UCG-15-02	**Use case:** Compute device risk level **Implementation:** The TMS computes a new value for the risk level of a device. Information about the current device trust level, the current status of network attacks and network traffic related to the device (as compared with the baseline), the device vulnerabilities and their exploitability, the device health level and views of peer-level TMSs are taken into account to produce a comprehensive risk score.

UCG-16-04	**Use case:** Identify and prioritize cyber-threats **Implementation:** Distinct cyberthreats are considered and their total impact on the protected network and its resources is assessed, producing a per-cyberthreat score. Cyberthreats are then ordered in descending score order to produce the result.

4.3 Technology update

The Cyber-Trust TMS extends the current TMS paradigms and implementations by considering additional dimensions in the computation of the trust scores, notably the status of the devices and the associated risk. For the computation of the associated risk, the business value of assets can be considered where available. The TMS implementation will be able to adapt to its runtime environment: in resource-rich environments the full capabilities of the TMS will be included, which necessitate extensive computations and ample resources, while in constrained environments some features will not be realized, with the respective functionalities being consumed as services offered by corresponding, trusted, feature-rich installations.

4.4 Application architecture

Figure 4-1 illustrates the conceptual view of the Trust Management Service. Its architecture is designed to allow for exposing a coherent API, enabling any adaptation aspects to be implemented internally considering all the appropriate contexts (network & resource availability, situation criticality etc.). Reception of information needed to recompute the trust and risk scores - including device status, behaviour an associated risk aspects - are mainly intercepted through asynchronous messaging, through a dedicated communication channel, following the pub/sub paradigm. In this way, the TMS is decoupled from event producers and their timings; however, content consumption via APIs can be also used. Reciprocally, the TMS publishes events regarding notable changes of trust and risk levels, while also offering the same information under REST APIs. Adaptation, where needed, will be supported by an adaptation component to be developed and maintained separately from the computational aspects, promoting separation of concerns.

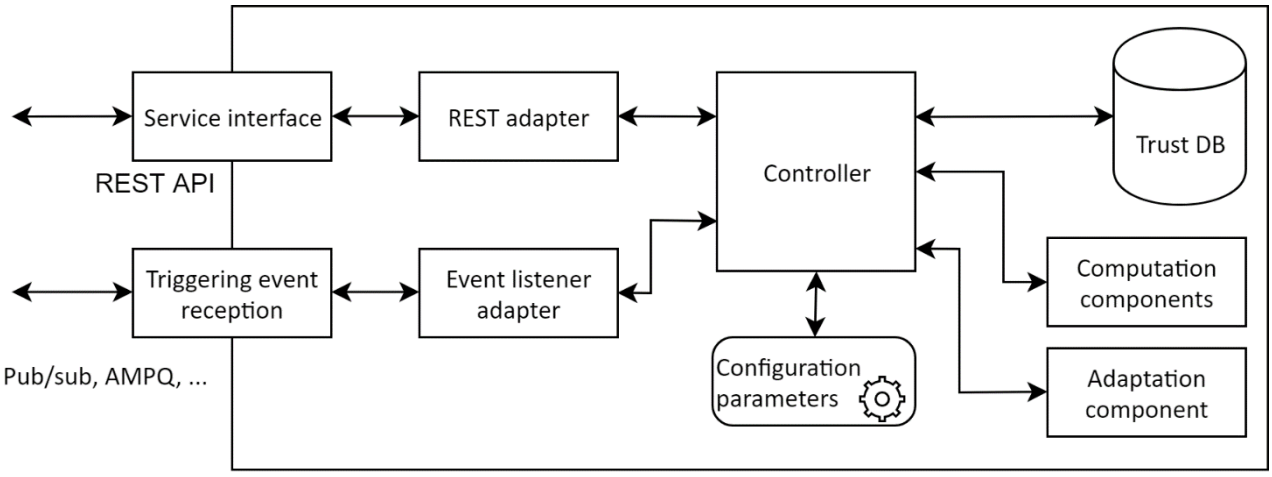

Figure 4-1. TMS high-level design

The overall high-level architecture of the TMS is depicted in Figure 4-1, while Figure 4-2 depicts the data view of the TMS, indicating:

(a) the data maintained internally in the TMS database;

(b) the messages that the TMS subscribes to in order to obtain the necessary information to compute trust and risk levels, as well as the sources of these messages, according to the overall Cyber-Trust architecture;

(c) the messages that the TMS makes available to the asynchronous communication infrastructure, for the perusal of other Cyber-Trust components.

Trusted Peer TMS are curated directly by users.

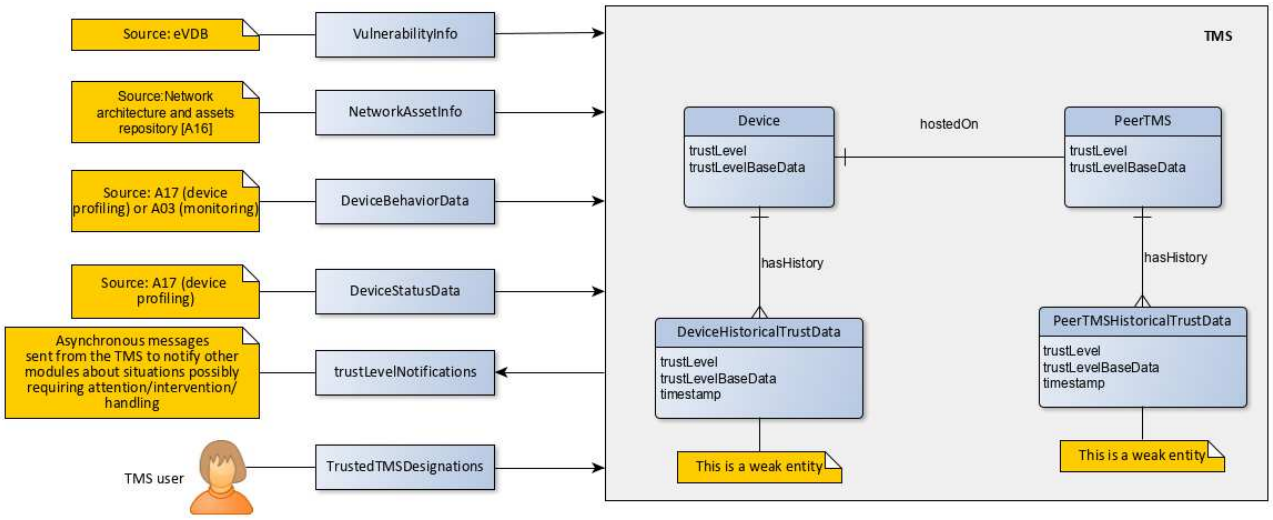

Figure 4-2. TMS data view

4.5 Application programming interfaces

The TMS exposes the REST APIs listed in the following subsections for direct invocation by other Cyber-Trust modules. As noted in subsection 4.4, the TMS additionally employs a loose coupling communication pattern, through the exchange of messages via the message bus; the respective messages consumed through the message bus will be elaborated on in the context of WP8.

4.5.1 REST APIs for managing device trust

Table 4-4 depicts the operations available for managing device trust, along with a brief description of each one.

Table 4-4. REST APIs for managing device trust

API URL specification	Description
GET /trust /info /{deviceId}	Returns the trust level for a device. The client may designate the desired trust dimensions. The information whether the reported trust level is explicit or implicit, is always returned.
PUT /trust/explicitLevel/{deviceId}	Explicitly specify the trust level of the device.
DELETE /trust/explicitLevel/{deviceId}	Delete the explicitly specified trust level of the device, returning to automatic computation.

GET /trust	Returns trust level for a set of devices. The client may designate the desired trust dimensions. The information whether the reported trust level is explicit or implicit, is always returned.

4.5.2 REST APIs for managing peer TMSs

Table 4-5 depicts the operations available for managing peer TMSs, along with a brief description of each one.

Table 4-5. REST APIs for managing peer TMS instances

API URL specification	Description
GET /peerTMS/{peerTMSId}	Returns information for a registered peer TMS
DELETE /peerTMS/{peerTMSId}	Deletes/unregisters a peer TMS.
PUT /peerTMS/{peerTMSId}	Creates or modifies a peer TMS.
GET /peerTMS	Returns information for a designated set of TMS
GET /peerTMS/list/all	Returns information for all registered TMS

4.5.3 REST APIs related to risk management

Table 4-6 depicts the operations available for risk management, along with a brief description of each one.

Table 4-6 REST APIs related to risk management

API URL specification	Description
GET /risks/prioritize	returns the top risks, prioritized. The number of risks to return is described in the (optional) numRisks parameter. If missing, a default number is inserted

4.5.4 REST APIs related to trusted user management

Table 4-7 depicts the operations available for trusted user management, along with a brief description of each one. Trust to users reflects on trust to the devices owned by them.

Table 4-7. REST APIs related to trusted user management

API URL specification	Description
GET /trustedUser/{trustedUserId}	Returns information about the designated trusted user.
DELETE /trustedUser/{trustedUserId}	Deletes/unregisters a trusted user.

PUT /trustedUser/{trustedUserId}	Creates or modifies a trusted user.
GET /trustedUser	Returns information for a designated set of trusted users
GET /trustedUser/list/all	Returns information for all registered trusted users

4.6 Technology Stack

The technology stack and tools used for the implementation of the TMS are listed in Table 4-8. The technology stack has not been modified since D4.4 [3].

Table 4-8. Technology stack and applied tools used for the implementation of the TMS

Tool	Description
Swagger	Employed for prototyping the REST APIs of the TMS
Java	The TMS functionality is coded in Java
Spring framework	The Spring framework is employed to intercept and serve REST API requests
MariaDB/MySQL	DBMS for managing the TrustDB
Javax.Persistence	For managing database connections and persistent entities
AMPQ/Asynchronous message protocol	For realizing pub/sub-based communications.

4.7 Physical architecture

In terms of physical architecture, the following deployment options exist:

1. The TMS is deployed as a single VM, running both the TMS and the data store (MariaDB/MySQL).
2. The TMS is deployed as two distinct VMs, one running the TMS while the second one running the data store. This option is preferable if a single data store is shared among multiple Cyber-Trust components.
3. The TMS is deployed as one single Docker container, running both the TMS and the data store. Taking into account that Docker containers are ephemeral, provisions should be made upon deployment to map the filesystem of the Docker container that holds the data to stable storage.
4. The TMS is deployed as two docker containers, one running the TMS and one running the data store. This option is preferable if a single data store is shared among multiple Cyber-Trust components.
5. The TMS is deployed as a Java application within a non-virtualized environment. This option is expected to be used (a) in environments not supporting virtualization and (b) in restricted environments where the overhead introduced by virtualization is not tolerable.

4.8 User Interface

The TMS runs as a service in Cyber-Trust platform and therefore it does not provide a dedicate own user interface (UI). However, certain UI elements are included in Cyber-Trust platform (e.g. information about the trust score of devices) to allow meaningful information to be provided to the user.

5. Intelligent intrusion response

5.1 Overview / objectives

The *Intelligent Intrusion Response (iIRS, A13)* module runs on the smart gateway at the user's home network. Its main responsibility being the real-time computation of mitigation actions that could be employed, with or without user interaction, against sophisticated network attacks. To this end, the iIRS receives alerts, in real-time, by the *Intrusion Detection System (IDS, A04g)* to update its belief about the security status of the smart home network (i.e. the capabilities an attacker might have acquired) – as described in D5.1 [6].

Then the attack graphical security model is calculated, presenting the interconnection between exploits and the security attributes of both network devices and their provided services (the capabilities an attacker has and might acquire) – see D2.5 for more details [7]. Fundamental for the creation of the attack graphical security model is the availability of comprehensive information about both the network and its hosts (i.e. present exploits, connectivity between hosts and subnetworks, etc.), and the attacker's actions.

To aid in optimization of the defence actions and to maximize the user's satisfaction, various attributes about the network devices and their provided services may be defined. Optimal response actions are calculated based on the attack graph and the user's preferences. The employment of those response actions can be either automatic or manual (suggested to the user).

The iIRS consists of three modules, all running on the smart gateway, that communicate using their REST endpoints:

a) the *iIRS Attack Graph Generator (iRG)* which is responsible for the calculation of the attack graph and of the remediation actions that may be employed,

b) the *iIRS Decision-making Engine (iRE)* which is responsible for the dynamic employment of the remediation actions based on the network's attack graph model,

c) the *iIRS Client (iRC)* which consists of two interfaces that control and display the status of each of the aforementioned modules.

Each of the three modules will be presented in more detail in the following sections.

5.2 Functionality coverage

5.2.1 Related requirements

Table 5-1 and Table 5-2 present the functional (FR) and the non-functional (NFR) requirements respectively that the component meets along with their related use cases, as defined in D4.4 [3].

Table 5-1. Functional requirements and use-case references for the iIRS

REF_ID	Description of implementation	Use Cases
FR55	**Requirement:** The user will be able to characterize each asset on the network and the respective value **Implementation:** The user may define an importance level for each device owned based on his preferences. These choices are used to maximize user's satisfaction and balance between security and availability.	UCG-04-02 UCG-04-03

FR56	**Requirement:** Cyber-Trust will automatically mitigate abnormal behaviour based on the network map, the characterization of the assets, the impact of the attack as well as the impact of the mitigation actions. If the mitigation action has severe impact on certain dimensions of assets that score high value Cyber-Trust will propose possible actions, but it will not implement it automatically. **Implementation:** The impact that the various mitigation actions have on the availability of network services (e.g. by refusing communication requests or shutting down running services) is quantified and can be tailored by the user. The user can choose if such decisions should be made automatically.	UCG-04-03 UCG-06-07
FR76	**Requirement:** The user (e.g. Security officer) will be able to create the cyber-attack graphical security model based on specific network infrastructures (architecture, topology, devices and related information). **Implementation:** The iIRS creates an attack graph presenting how exploits relate to various security attributes and vulnerabilities found in the smart home environment. The information needed to construct the attack graph is obtained mainly from the A16 component, and possibly complemented by the profiling service or the eVDB.	UCG-15-01
FR77	**Requirement:** Development of appropriate UI for entering dynamic parameters regarding the system (i.e. state transition model, expected utility function). These parameters will be used in order to re-calculate attack's likelihood and success probability. **Implementation:** These computations of an attack's likelihood and success probabilities are performed by the iIRS by utilizing information from the A16 component, and possibly from the eVDB and the profiling service.	UCG-15-03
FR80	**Requirement:** Intelligent Intrusion Response System (iIRS) will compute a suitable defense action based on (at least) the system security state and the attacker's profile. **Implementation:** The iIRS computes the optimal defense action based on the information it possesses about the system security state and the attacker's profile.	UCG-18-05
FR81	**Requirement:** The security Officer will be able to initiate the process of defining/updating the applicable mitigation actions on the system of devices through the system's UI (based on new available exploits and possible action for these exploits). **Implementation:** The applicable mitigation actions are either defined manually (by security experts) or automatically based on the information available from the eVDB or the profiling service.	UCG-18-06
FR82	**Requirement:** Based on FR81: The user (based on access role) selects the applicable mitigation actions for each exploit. **Implementation:** The applicable mitigation actions are either defined manually (by security experts) or automatically based on the information	UCG-18-06

	available from the eVDB or the profiling service. Access control rights are enforced by other components, e.g. the authentication service and A06.	

Table 5-2. Non-functional requirements and use-case references for the iIRS

REF_ID	Description of implementation	Use Cases
NFR40	**Requirement:** iIRS will use the alerts raised by the IDS in order to update the belief it possesses over the system security state. **Implementation:** The iIRS uses the alerts provided by the intrusion detection system [A04g] to update the belief it has about the system security state (i.e. an attacker's acquired capabilities within a network).	UCG-15-04

5.2.2 Related use cases

Table 5-3 lists the use cases related to the iIRS and the provisions made by the component to support the fulfilment them.

Table 5-3. Use-cases related to the iIRS

REF_ID	Description of implementation
UCG-04-02	**Use case:** Characterize asset's importance. **Implementation:** The user is allowed to define an importance level for each device owned according to his/her preferences. These choices are used by the iIRS to maximize user's satisfaction while balancing between security and availability in the defence actions to apply (i.e. which exploits to block/leave open to ensure network services' availability).
UCG-04-03	**Use case:** Define mitigation actions' impact. **Implementation:** The impact that the various mitigation actions have on the availability of network services (e.g. by refusing communication requests or shutting down running services) is quantified. Along with the smart home owner's preferences, this is used to define the utility function that is required by the iIRS.
UCG-06-07	**Use case:** Communicate iIRS actions to the security officer. **Implementation:** The iIRS after computing the optimal defence action, it informs the user (security officer) through the intelligent UI portal.
UCG-15-01	**Use case:** Compute cyber-attack graphical security model. **Implementation:** The iIRS creates an attack graph presenting how exploits relate to various security attributes and vulnerabilities found in the smart home environment. The information needed to construct the attack graph is obtained mainly from the A16 component, and possibly complemented by the profiling service or the eVDB.
UCG-15-02	**Use case:** Compute device risk level.

	Implementation: The iIRS is involved in this use-case, with TMS being the main actor for computing a device's risk level; the iIRS provides information about network-wide risks and the current status of network attacks.
UCG-15-03	**Use case:** Compute attack's likelihood and success probability. **Implementation:** These computations of an attack's likelihood and success probabilities are performed by the iIRS by utilizing information from the A16 component, and possibly from the eVDB and the profiling service.
UCG-15-04	**Use case:** Compute a belief on current security status. **Implementation:** The iIRS uses the alerts provided by the intrusion detection system [A04g] to update the belief it has about the system security state (i.e. an attacker's acquired capabilities within a network).
UCG-16-03	**Use case:** Receive intrusion detection system(s) alerts. **Implementation:** The iIRS regularly obtains the alerts generated by the IDS [A04g], which constitutes the primary source of input, and evaluates them in order to infer the true system security state, by considering the possible mis-detections and false alarms.
UCG-18-01	**Use case:** Apply Mitigation Policy on Device. **Implementation:** The iIRS is involved in this use-case, with the SDA and SGA (actually, the A04g component) being the main actors for applying the mitigation actions computed. The iIRS communicates the mitigation actions to the IDS for being applied at the network level, and then at the device level as well.
UCG-18-05	**Use case:** Compute optimal intrusion response actions. **Implementation:** The iIRS computes the optimal defence action based on the information it possesses about the system security state and the attacker's profile.
UCG-18-06	**Use case:** Define applicable mitigation actions. **Implementation:** The applicable mitigation actions are either defined manually (by security experts) or automatically based on the information available from the eVDB or the profiling service.

5.3 Technology update

This section documents at a high level the changes on the tools which the modules of the iIRS are built upon (*see* Section 0 for the details). This concerns mainly the iRG which is based on the open source server component of the FIWARE CyberCAPTOR project[1], with a number of significant modifications, extensions and architectural changes to fulfil the requirements of Cyber-Trust.

At the core of the iIRS module, and more specifically the main responsibility of the iRG, is the generation of the attack graphical security model from the network topology information obtained by the A16 component, and the calculation of the static risk analysis model for which the attack graph is primarily generated.

[1] https://cybercaptor.readthedocs.io/

To accommodate the complexity of the smart home networks a new and more advanced risk analysis model has been implemented. This advanced model focuses on the attempt probability (whether an exploit will be chosen over others) and the success probability (whether an exploit will succeed once exploited) of exploits to assess their presented risk for the network.

Along with the new risk analysis model, a new algorithm was implemented to calculate the appropriate firewall rules to be applied or suggested to the user, including the impact of each rule on the attack graph model of their application.

To accommodate the communication requirements of Cyber-Trust and to ensure full compliance with the information bus requirements, the REST API was completely rewritten (including changes in the REST endpoint naming). The new API implements a more robust error checking and reporting routine, with support for digital signing of the message's payload to prevent data tampering attacks.

To support the interconnection of the iRG with the other component's submodules (the iRE and iRC) and the rest of the components of Cyber-Trust (mainly the A16, A04g components and the eVDB) multiple REST endpoints were created to receive and provide data.

Schema changes to the internal database were necessary to support the storage of Common Vulnerability Scoring System (CVSS) version 3.1 information, in addition to the already existing support for CVSS version 2, and to support integration with the eVDB.

Along with the aforementioned additions and changes, the Docker image creation process had to be restructured to support the (currently) private Cyber-Trust GitLab repository, used during the development process, and to be better suited for the handling of cryptographic keys with care for development artefacts.

5.3.1 Attack Graph Generation Tool Comparison

MulVAL is a wide-used tool for producing logical attack graphs and each software associated with it has a pre-defined set of rules. Those rules describe either initial conditions (called *pre-conditions*) or conditions resulting from the application of exploits (called post-conditions). Table 5-4 shows the differences between MulVal, CyberCAPTOR, Doctor and Cyber-Trust's iIRG. Our rules stay similar to those of CyberCAPTOR, as the demands for our attack graph's generation are not altered and at the same time are more advanced, compared to those of the original MulVAL.

Table 5-4. Comparison of rules used in the attack graphs.

Rule	M	C	D	iRG	Example
vulExists	✓	✓	✓	✓	vulExists(_host, _vulID, _program)
vulProperty	✓	✓	✓	✓	vulProperty(_vulID, _range, _consequence)
haclPrimit	✓	✓	✓	✓	haclprimit(_src, _dst, _prot, _port)
attackerLocated	✓	✓	✓	✓	attackerLocated(_host)
hasAccount	✓	✓	✓	✓	hasAccount(_principal, _host, _account)
netWorkServiceInfo	✓	✓	✓	✓	**Doctor and CyberCAPTOR version:** networkServiceInfo(_ip, _program, _protocol, _port, _user) **MulVAL version:**

					networkServiceInfo(_host, _program, _protocol, _port, _user)
installed	✓	✓	✓	✓	installed(_h, _program)
isInVlan		✓	✓	✓	isInVlan(_ip,_vlan)
vlanToVlan		✓	✓	✓	vlanToVlan(_vlan1,_vlan2,_protocol,_port)
ipToVlan		✓	✓	✓	ipToVlan(_ip,_vlan,_protocol,_port)
vlanToIP		✓	✓	✓	vlanToIP(_vlan,_ip,_protocol,_port)
defaultLocalFilteringBehaviour		✓	✓	✓	defaultLocalFilteringBehavior(_toip,_behavior)
localFilteringRule		✓	✓	✓	localFilteringRule(_fromIP,_toIP,_port,_behavior)
hasIP		✓	✓	✓	hasIP(_host,_IP)
IpInSameVLAN		✓	✓	✓	ipInSameVLAN(_ip1,_ip2)
localAccessEnabled		✓	✓	✓	localAccessEnabled(_ip,_fromIP,port)
execCode		✓	✓	✓	execCode(_host, _user)
netAccess		✓	✓	✓	**Doctor and CyberCAPTOR version:** netAccess(_ip,_protocol,_port) **MulVAL version:** netAccess(_machine,_protocol,_port)
canAccessHost		✓		✓	canAccessHost(_host)
hacl		✓	✓	✓	hacl(_src, _dst, _prot, _port)
attackGoal		✓	✓	✓	attackGoal(_)
advances		✓		✓	advances(_, _)
accessFile			✓		accessFile(_machine,_access,_filepath)
canAccessFile			✓		canAccessFile(_host, _user, _access, _path)
vnfManagedBy			✓		vnfManagedBy(_host,_vnfm)
cvss			✓		cvss(_vulID, _ac)
inCompetent	✓				inCompetent(_principal)
competent	✓				competent(_principal)

clientProgram	✓				clientProgram(_host, _programname)
setuidProgramInfo	✓				setuidProgramInfo(_host, _program, _owner)
nfsExportInfo	✓				nfsExportInfo(_server, _path, _access, _client)
nfsMounted	✓				nfsMounted(_client, _clientpath, _server, _serverpath, _access)
localFileProtection	✓				localFileProtection(_host, _user, _access, _path)
accessMaliciousInput	✓				accessMaliciousInput(_host, _principal, _program)
principalCompromised	✓				principalCompromised(_victim)
dos	✓				dos(_host)
logInService	✓				logInService(_host, _protocol, _port)
dependsOn	✓				dependsOn(_h, _program, _library)
installed	✓				installed(_h, _program)
bugHyp	✓				bugHyp(_,_,_,_)
canAccessFile	✓				canAccessFile(_host, _user, _access, _path)
isWebServer	✓				isWebServer(_host)
vmOnHost			✓		vmOnHost(_vm, _host, _software, _user)
vmOnDomain			✓		vmInDomain(_vm, _orchestrator)
vnfOnPath			✓		vnfOnPath(_vnf,_host1,_host2,_port,_daemon,_user)
localServiceInfo			✓		localServiceInfo(_servicename, _host, _program, _user)
hasNDNFace			✓		hasNDNFace(_host, _face)
isNDNRouter			✓		isNDNRouter(_host)
ndnServiceInfo			✓		ndnServiceInfo(_host, _software, _user)
ndnLink			✓		ndnLink(_host1, _face1, _host2, _face2
ndnOutputCompromised			✓		ndnOutputCompromised(_ndnRouter,_signatureMode
ndnOutputCompromised Local			✓		ndnOutputCompromisedLocal(_ndnRouter)
ndnOutputCompromised Remote			✓		ndnOutputCompromisedRemote(_ndnRouter1, ndnRouter2,_signatureMode

ndnTrafficIntercepted			✓		ndnTrafficIntercepted(_ndnRouter)

The exploits supported by the rules of Table 5-4 can lead to many interaction rules, where there exists no one-to-one mapping between the exploits and the interaction rules, which can be generated in different ways by multiple combinations. In principle, the **attacker's goal** is linked with the desired ability to execute arbitrary code at a certain IoT device. This is defined as follows:

```
execCode(_attacker, _host, _permission)
execCode(_host, _permission)
```

where _name represents variables (with the names being self-explained in the above rule). By eliminating a variable, this means that the rule should hold for any value of the eliminated variable. Hence, it is common that the attacker's goal is stated as follows:

```
attackGoal(execCode(_host, root))
attackGoal(execCode(smartTV, root))
```

where the attacker aims at obtaining root privileges at any or a specific machine — a smart TV in the above example.

5.3.2 Active Mitigation Action Calculation

An efficient and effective algorithm was implemented to support the generation of active remediations, as are requested at run-time by the iRE, to achieve temporary changes to the attack graph by changing the network topology. The most basic way to change the network topology was to change the interconnectivity of hosts, both in the same subnetwork and across subnetworks, and thus effectively block access to vulnerable services by employing firewall rules at the gateway.

At first, the algorithm begins with the desired node to be blocked (that is, to be temporarily removed along with its subgraph from the attack graph) and moves towards the leaves of the graph. It explores (using depth first search) whether any node has enough information to generate a firewall rule and stores their connections and relations in a tree structure. This structure can represent multiple sets of firewall rules that can be applied to block the specified (by the iRE) attack graph node.

Starting from the node to be blocked:

- When a node, regardless of its type, can generate a firewall rule, the required information is added to the tree, and exploration on this part of the graph is terminated. The depth first search pattern continues with the next attack graph path.
- When an "OR" attack graph node is reached, a new "AND" operator node is added to the tree. As to render invalid an "OR" attack graph node, all of its parent nodes need to be invalidated.
- When an "AND" attack graph node is reached, a new "OR" operator node is added to the tree. Symmetrically with the previous case, to render an "AND" attack graph node invalid, at least one of its parent nodes needs to be invalidated.

- When a "LEAF" attack graph node is reached, a NULL tree node is added. This is necessary for the trimming phase, as every tree path that doesn't end in a firewall rule node needs to be removed.

The initial tree generated when searching for active remediations on a root node ("OR" type) of the attack graph and after the removal of paths ending in NULL tree nodes.

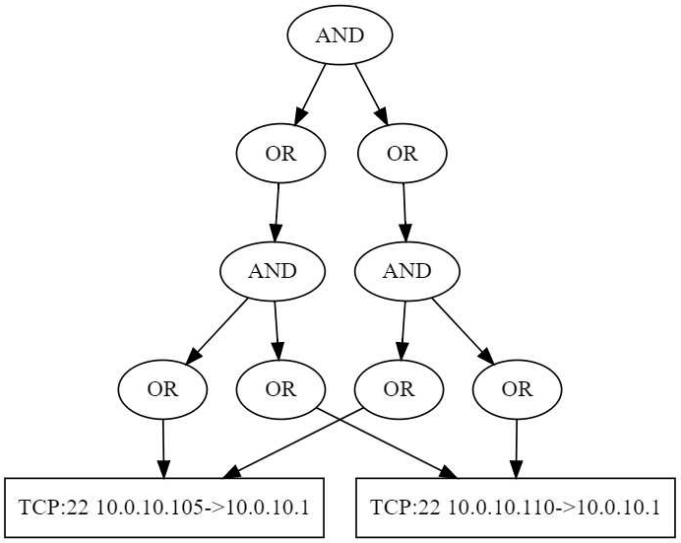

Figure 5-1. Initial tree generated by the remediation generation algorithm

After the tree is generated, the trimming process is applied to remove tree paths that result in a NULL node, and a process collapsing its operators is repeatedly applied on the tree to simplify its structure, making it easier to process when generating the final solutions. The final form of the tree, after the collapse of extraneous operators.

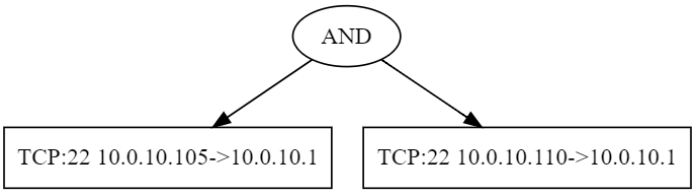

Figure 5-2. Simplified tree generated after the pruning and collapsing process

The final solutions generated from the above process are in a canonical form that resembles the *disjunctive normal form* (DNF) in logical expressions and Boolean circuits; i.e. it is a disjunction of conjunctions:

$$(R_1 \wedge \cdots \wedge R_k) \vee (R_1 \wedge \cdots \wedge R_n) \vee \cdots \vee (R_1 \wedge \cdots \wedge R_m)$$

where R_i represents a firewall rule. This allows iRE the choice between multiple solutions (of possibly many firewall rules) that block the specified attack graph node, a choice that can be made by the user (in manual mode) or by the iRE directly (in auto mode) by ranking each group based on a set of defined criteria.

5.3.3 Risk Analysis

Various models have been proposed regarding the risk analysis on attack graphs or attack trees. Our model starts with the general idea proposed by [13] that focuses on the attacker's capabilities and the likelihood of a particular attack being executed. The *Local Conditional Probability Distribution* (LCPD) table is computed

for each node, which defines the probability of a node to be compromised given the combined value of the states of its parent(s) – see Figure 5-3.

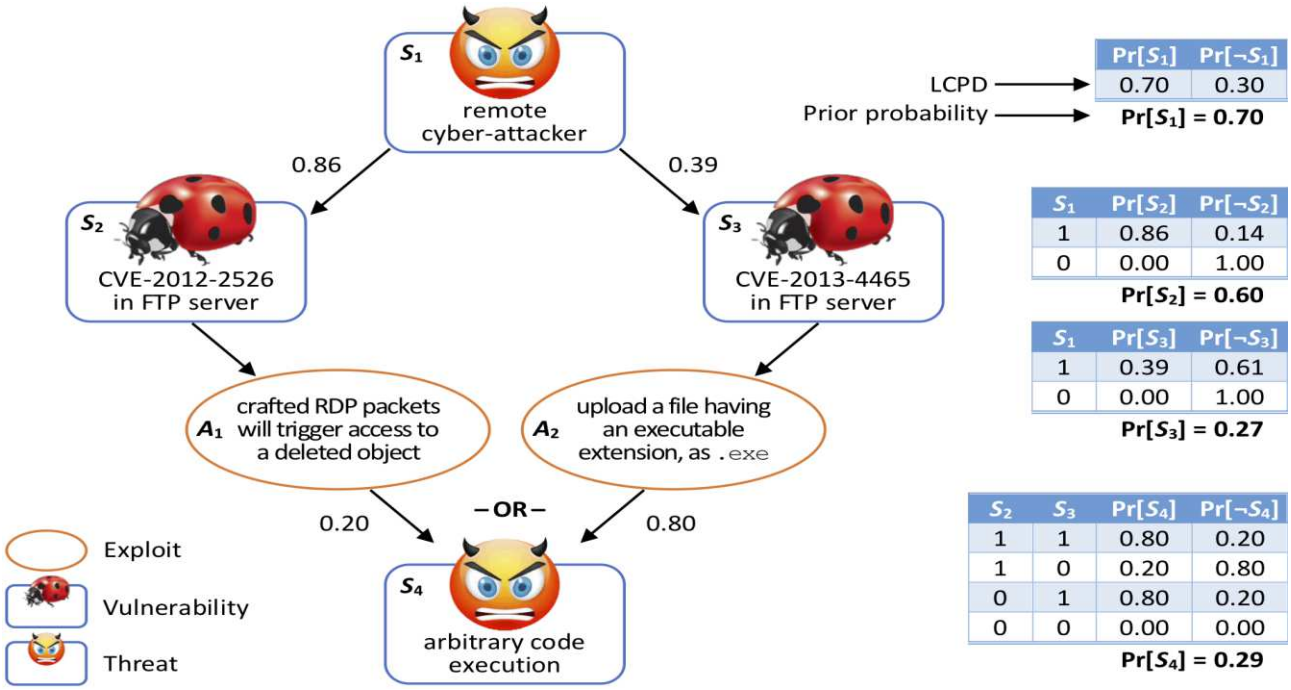

Figure 5-3. An example of a Bayesian attack graph

The values on LCPDs occur based on the CVSS metrics. Specifically, according to the type of the destination node on an edge of the graph, we define the attempt and success probabilities if there is a vulnerability as a parent/source node. The destination node on a random edge can be either an AND (exploit) or an OR (security condition) node of the logical attack graph generated by MulVAL. In particular, for an AND node

$$Pr[\text{attempt}] = \begin{cases} E \cdot RL \cdot RC, & \text{if temporal metrics are available} \\ 1 - (1 - C) \cdot (1 - I) \cdot (1 - A), & \text{otherwise} \end{cases}$$

where *E* is the exploit code maturity, *RL* the remediation level, *RC* the report confidence, *C* the confidentiality, *I* the integrity, and *A* the availability. Likewise, for an OR node we have

$$Pr[\text{success}] = \begin{cases} 2{,}11 \cdot AV \cdot AC \cdot PR \cdot UI, & \text{if CVSS v3 metrics are available} \\ 2{,}00 \cdot AV \cdot AC \cdot Au, & \text{otherwise} \end{cases}$$

where *AV* is the attack vector, *AC* the attack complexity (or access complexity in CVSS v2), *PR* the privileges required, *UI* the user interaction, and *Au* the authentication. To compute the unconditional probability for each node, the following expression needs to be computed

$$Pr[\mathrm{X}_i] = \sum_{\boldsymbol{X}-X_i} Pr[\mathrm{X}_1, \ldots, \mathrm{X}_n] = \sum_{\boldsymbol{X}-X_i} \prod_{j \neq i} Pr\left[\mathrm{X}_j \mid \mathrm{Pa}(\mathrm{X}_j)\right]$$

where $\boldsymbol{X} - X_i$ indicates that the sum is over all the possible states of all the random variables except X_i. However, this calculation is a NP-hard since the complexity of a typical algorithm has $O(2^n)$ complexity. Applying brute force techniques for the computation of the unconditional probabilities is not a reasonable

approach because all computations need to be done at almost real time with low memory needs. Thus, we use a different approach to compute the unconditional probabilities using the *Loopy Belief Propagation* (LBP) algorithm. This approach works only on factor graphs that are associated with a Bayesian network.

The message passing algorithm focuses on passing pre-computed values through the network as messages. There are two different kind of messages: from Factor nodes to Variable nodes and from Variable nodes to Factor nodes. LBP updates all the messages for all factors and variables at the same time using values from previous iterations. The algorithm runs until we reach the maximum pre-defined number of iterations or until an ℓ_2 metric get smaller than 10^{-2}. The unconditional probabilities are computed as the product of incoming final messages from neighbouring factor nodes to the corresponding variable node. Not only there is a significant time complexity reduction in our model, but the memory used is notable less.

The Risk Analysis model also considers the criticality associated with each device as an input from the user. As it can be seen from the iRG client interface (see Section 5.8), there are various possible choices regarding the criticality of the Hosts. More specifically, the user can assign a value of

- None (10),
- Negligible (10),
- Minor (30),
- Medium (50),
- Severe (70) and
- Catastrophic (90).

5.3.4 Optimal decision-making

The iIRS's decision making engine consists of two components: the server and the client. The server performs all necessary computations including the processing of security information passed from other system components (e.g. the attack graph, remediation actions and security alerts), local optimal policy estimation, attack propagation belief update and user preference tuning. It is an essential component of the final platform. In contrast, the client is a component most useful for testing and development and its main functionality is to enable the visual representation of simulated attacks and defence scenarios. Both parts of the software were developed from scratch (no open source software is available) and is documented in Sections 5.4.5 and 5.8 respectively.

5.4 Application architecture

This section presents the iIRS architecture from two different viewpoints: the high-level view and the data-centric view and presents the internals of the three major subcomponents of the iIRS.

5.4.1 High-level architecture

The high-level architecture of Figure 5-4 illustrates the existence of the three main responsibilities of the iIRS:

a) the generation of the attack graph model and the calculation of the risk state of the network, performed on the iIRS Attack Graph Generator (iRG);
b) the calculation of the optimal defence actions, performed on the iIRS Decision-making Engine (iRE); and
c) the display of all the relevant information in a user-friendly way, performed by the iIRS Client (iRC).

Each subcomponent of the iIRS communicates internally, through localhost, via their exposed REST endpoints while presenting a unified API externally, to the other Cyber-Trust components. This allows greater flexibility

on their deployment, compartmentalization of their data and code, and allows for independent development of each submodule.

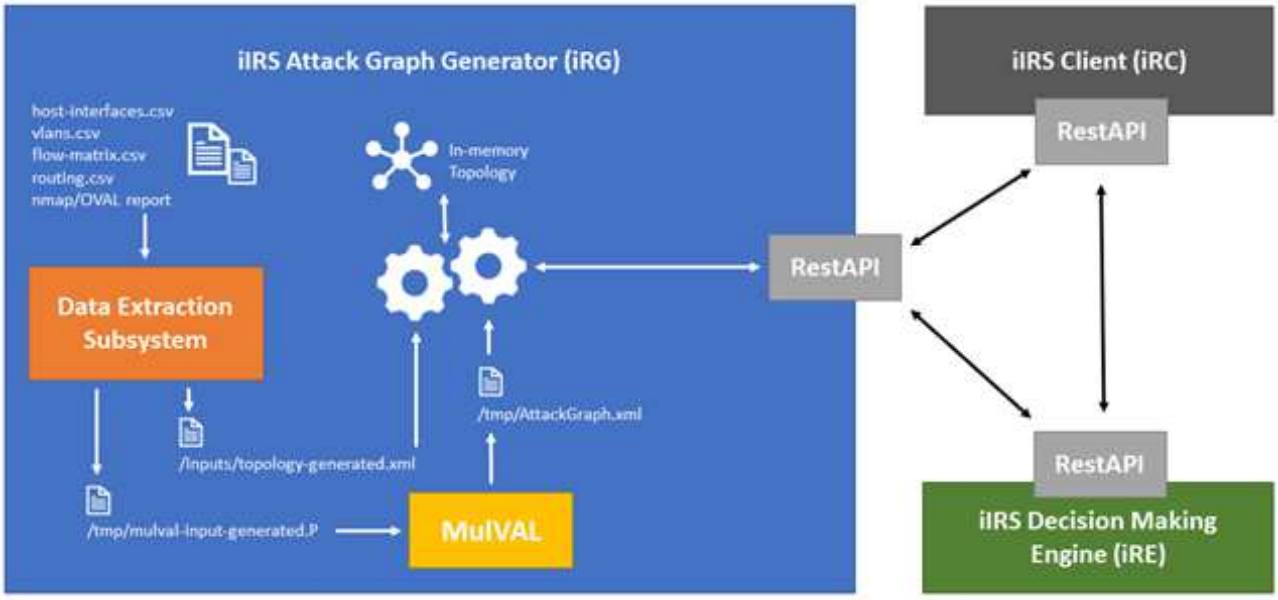

Figure 5-4. Architecture of the iIRS component

5.4.2 Data-centric architecture

The data-centric view (see Figure 5-5) presents and puts the iIRS into the greater perspective of Cyber-Trust, by displaying the data requirements and requests along with the modules that provide or consume that data. From that view the associated modules and their relation with the iIRS [A13] become apparent.

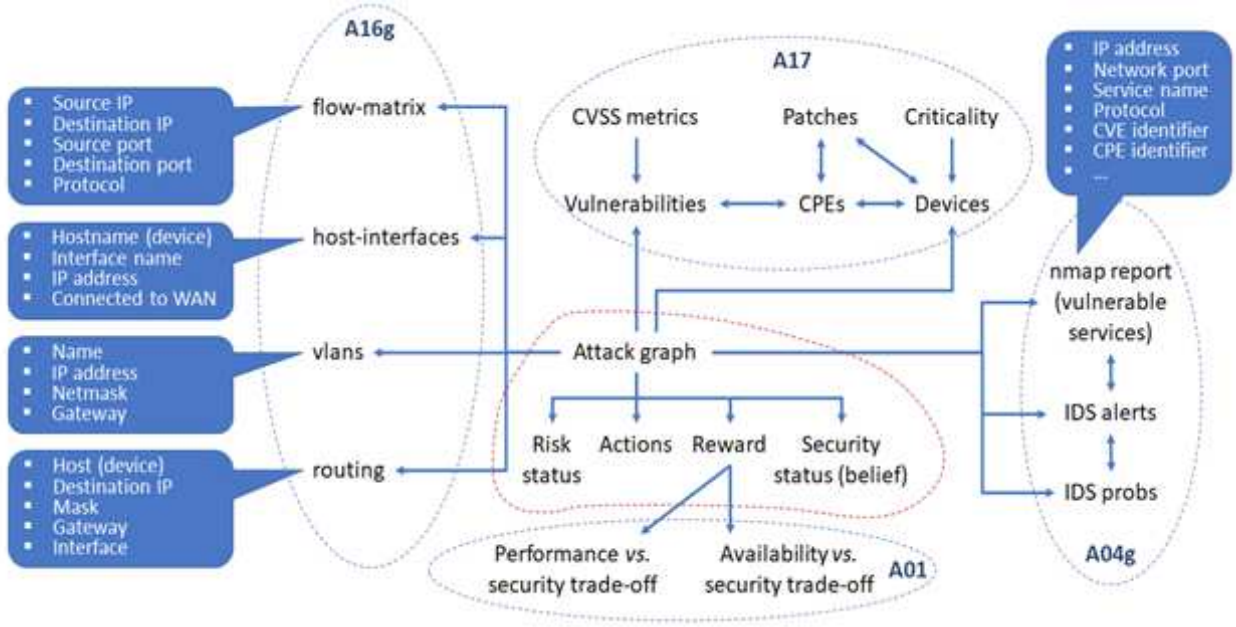

Figure 5-5. Data-centric view of the iIRS component

The iRG, and by extension the iIRS, is connected with the A16g, A04g and the A17 (eVDB) components, from which it receives the following information:

- Detailed information about the network topology, the subnetworks and information about each host; from the A16g module through Rest API calls #7 and #9-12.
- Information about the exploitable vulnerabilities of each network host; from the A04g module via Rest API call #8.
- Information about available remediations, CVSS metrics, etc.; from the A17 (eVDB) module, updated through Rest API call #3.

The iRE is connected internally with the iRG and externally with the A04g component, from which it receives:

- The generated attack graph model along with information about the real-time actionable remediation actions (firewall rules blocking specific attack graph nodes, as detailed in previous sections); from the iRG.
- Alerts about the current state of the smart home network allowing the iRE to act in response; from the A04g module.

5.4.3 Remediation DB

IRG's Remediation DB has drastically changed to that of the CyberCAPTOR. It stands as a subcomponent of the iRG server, has its own daily updating mechanism and contains up to the latest CVEs found on the eVDB component. *Remediation DB* is also capable of communicating with the National Vulnerability Database in case the former communication is not possible. When eVDB sends the appropriate data to the integration bus, the remediation DB is instantly updated. The "vulnerability" table (see Table 5-5) is the basic table of the IRG's Remediation DB.

Table 5-5. The "vulnerability" table of the remediation DB

Attribute	Type	Example
id	INTEGER PRIMARY KEY AUTOINCREMENT	123899
cve	TEXT UNIQUE	CVE-2019-9974
description	TEXT	diag_tool.cgi on DASAN H660RM GPON routers with firmware 1.03-0022 lacks any authorization check, which allows remote attackers to run a ping command via a GET request to enumerate LAN devices or crash the router with a DoS attack.
cvss_id	INTEGER	123899

Table "cvss" (*see* Table 5-6) now supports both CVSS 3.1 and CVSS 2, giving priority to the most recent version of the standard. We have also initialized the temporal metrics with "-1" as we don't want them to corrupt the computations of various probabilities when they are unavailable. eVDB provides the temporal metrics only for a subset of the vulnerabilities due to the lack of such information from the sources of information.

Table 5-6. The “cvss” table of the remediation DB

Attribute	Type	Example
id	INTEGER PRIMARY KEY AUTOINCREMENT	123899
score	REAL	9.7
attack_vector	TEXT	NETWORK
attack_complexity	TEXT	LOW
authentication_priviledges	TEXT	NONE
user_interaction	TEXT	NONE
scope	TEXT	UNCHANGED
confidentiality_impact	TEXT	HIGH
integrity_impact	TEXT	NONE
availability_impact	TEXT	HIGH
exploit_code_maturity	TEXT DEFAULT ‘-1’	-1
remediation_level	TEXT DEFAULT ‘-1’	-1
report_confidence	TEXT DEFAULT ‘-1’	-1

The “patches” table (*see* Table 5-7) has also been adjusted to contain patches having been labelled with the “Patch”, “Vendor Advisory” and “Third Party Advisory” tags, whereas the rest of the advisories that are being imported to the eVDB from various vulnerability databases are considered to be irrelevant (or do not contain useful information) for remediation purposes.

Table 5-7. The “patches” table of the remediation DB

Attribute	Type	Example
id	INTEGER PRIMARY KEY AUTOINCREMENT	54402
link	TEXT	http://www.vupen.com/english/advisories/2009/1911
description	TEXT	ADV-2009-1911
tags	TEXT	Patch, Vendor Advisory

The 'link' attribute shows the URL provided for the patch and some information can be seen in the attribute 'description' where comments are often provided by vendors or third parties. Finally, the 'tags' attribute contains the labels associated with a remediation.

5.4.4 iIRS Attack Graph Generator (iRG) Server

The iRG is responsible for the generation of the attack graph from the topology information (obtained by the A16 component), the calculation of the initial risk score (representing the initial security state belief of the network) and the retrieval of real-time actionable remediation actions. The two major subcomponents of the iRG (as seen in Figure 5-4) are responsible for the first two responsibilities:

- the Data Extraction subsystem, a python script that compiles the topology information in XML form from the data reported by the A16; and
- the MulVAL attack graph generator, the system responsible for the generation of the actual attack graph from a set of Datalog (Prolog-like) rules and the Datalog-converted topology.

The generation of the XML topology also requires information about the reported vulnerabilities, e.g. the CVSS score of the vulnerability (required to calculate the initial risk score and the probabilities associated with each vulnerability). This information is obtained by the A17 and eVDB components and stored locally in the remediation DB, ensuring quick access and constant availability to its information (even if connections with external systems are not available).

After the generation of the attack graph by MulVAL, the main iRG Java application loads both the topology information and the attack graph in memory on which the risk analysis process and the actionable remediations are calculated.

5.4.5 iIRS Decision-making Engine (iRE) Server

The iRE server communicates with various other Cyber-Trust platform components including the iIRS's attack graph generator (iRG), the IDS and the user interface, while the client only communicates with the server (see also Figure 5-6). In this section we give a detailed description of the architecture, the main functionality and technical aspects of the iRE server component, whereas the iRE client is further described in Section 5.8.

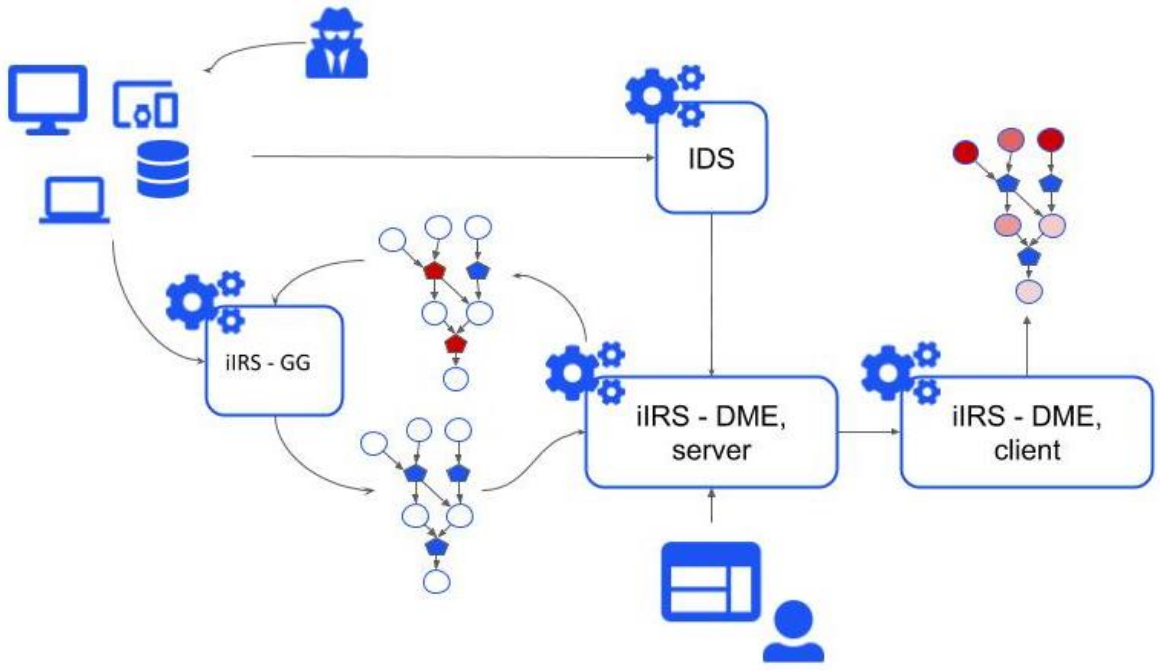

Figure 5-6, High level diagram of the iRE's interactions with other components

We start with the server description. Note that the server's API is stateful; at any moment a state is maintained with all the system specific parameters. When a request is received, it is served according to the current server state, which consists of the following parameters:

- "min_iterations": integer (default value 2000), specifies the minimum number of simulations to be run while exploring the POMCP tree for locally computing the optimal policy at the current belief
- "security_availability_tradeoff": float (default value 0.5), specifies the relative weights of the security and availability costs in the POMDP instantaneous reward function
- "Max_processes": integer (default value 16), specifies the maximum number of processes that will be generated to run simulations in parallel
- "cvss_weights": tuple of floats (default value 0.33,0.33,0.33), specifies the relative weights of cvss metrics in the POMDP instantaneous reward. The considered metrics are Impact, Availability and Exploitability and comprise the security risk part of the reward.

The server's state can be altered using the API POST call "/parameters" (see API specification) which expects a message in json format the keys of which correspond to the available parameters in the state and the values correspond to the desired new values. Parameters not included in the message are kept the same. Any other API call to the server will keep these parameters the same.

When instantiated the server expects to receive an attack graph specification message in json format through the "/uploadTopology" api call (see API specification). The message includes information about the attack graph topology, the goal conditions and the associated IP addresses and ports for each node. This information is leveraged by the iIRS decision engine server to internally construct the data structures necessary for running the simulations. The goal conditions are nodes with an increased security cost typically denoting a severe security breach (e.g. root access to machine). The IP addresses and ports are used to match the attack graph exploits to the received security alerts from the IDS (when these arrive).

The iIRS decision engine is initiated on receipt of an attack graph. However, the attack graph evolves in time as new devices appear in the local network or get removed. The graph generator engine updates the topology of the attack graph when appropriate and makes a new call to the decision engine to update the information held for running simulations. On such subsequent calls of "/uploadTopology" the decision making engine will kill all processes running and store the belief on the attack propagation through the system. Simulations on the new attack graph will immediately start running. Depending on the differences between the old and new attack graphs information for the previous belief may be utilized.

After receiving the attack graph, the server will make an API call to the iIRS graph generator engine at the endpoint "/attack_graph/remediations/all", which will return the available remediation actions and the associated attack graph nodes that they affect. These are received in json format. After the actions are received the server will request a security alert from the IDS and once this is obtained, the server will start running simulations to compute the optimal remediation action at the current belief. The simulation parameters are specified by the server's state.

When the minimum number of simulations has been completed the server returns the computed optimal policy at the local belief. In particular, this is passed to the iIRS - graph generator engine though the API call "/attack_graph/remediations/all". Note that the server generates many processes for running simulations. Each process repeatedly computes a simulated trajectory and updates the values of nodes on a shared attack graph object held internally in the server. The number of processes is set by the "max_processes" parameter and is by default the minimum between 16 and the number of cores in the host machine. Note also that the number of cores in the host machine is always chosen if the specified "max_processes" parameter is above it.

5.5 Application programming interfaces

The structure of messages exchanged both internally (by the iIRS subcomponents) and externally (between Cyber-Trust components) via the information bus is presented in this subsection.

```
{
    “source”: “smart_gateway_module_id”,
    “type”: “information_bus_topic”,
    “timestamp”: unix_epoch,
    “payload”: {
        “test”: “content”
    },
    “signature”: {
        “alg”: “algorithm_name”,
        “sig”: “base_64_encoded_signature”
    }
}
```

This structure contains information that permits the identification of the originating source (the smart gateway device ID), the information bus topic the message is posted to, the timestamp allowing identification of old and possibly expired information, and a section containing the digital signature of its contents to detect any data tampering attempts – see Table 5-8.

Table 5-8. Generic header information in Cyber-Trust asynchronous messages

Field	Description	Type	Example
source	The ID of the smart gateway the iIRS runs on.	String	-
type	Predefined keyword to identify the information bus topic on which the message is posted.As defined in the information bus specification.		“newvulnerabilities", “mitigation”, “updateinfo”, “alert”
timestamp	The timestamp generated at response-time in UNIX epoch format.	Number	1578832835
payload	The contents of the message.	Object	-
signature	Payload signature information.		
signature/alg	The algorithm to sign the payload in the format: HashAlgorithm + “With” + EncryptionAlgorithm.	String	“sha256WithRSAEncryption”
signature/sig	The signature of the payload in Base64 encoding.		-

5.5.1 iIRS Attack Graph Generator (iRG)

All JSON messages originating from the iRG contain an extra structure included in the payload of the previously presented structure. This structure provides further information about the API version of the iRG

that could be used programmatically to detect changes to the API, and about the status of the call: whether the process was successful or failed including an error message intended for the user or for debugging purposes (see also Table 5-9).

```
{
    “source”: “smart_gateway_module_id”,
    “type”: “information_bus_topic”,
    “timestamp”: unix_epoch,
    “payload”: {
        “api”: api_version,
        “result”: {
            “status”: “OK”,
            “message”: “message_for_the_caller”
        },
        “test”: “content”
    },
    “signature”: {
        “alg”: “algorithm_name”,
        “sig”: “base_64_encoded_signature”
    }
}
```

Table 5-9. iIRS specific header information in Cyber-Trust asynchronous messages

Field	Description	Type	Example
api	The version of the iRG Server API in the format “##.##.##” and “##.##.##b” for beta versions. To track and detect if updates to the JSON structure (in the payload) were made.	String	“2.0.0b”, “2.0.1”
result	A structure containing information about the process performed by the call.	Object	-
result/status	A binary status flag indicating whether an operation was successful or failed. Its values are restricted to “OK” and “ERROR”.	String	“OK”, “ERROR”
result/message	An explanatory message, intended for a human caller, describing the status flag. Error messages starting with the phrase: “Internal error” refer to errors concerning the internal process of the iRG server and doesn’t indicate errors on the part of the caller.		"Internal error, the simulated attack graph couldn't be generated." "The remediation ID=5 is invalid. There are only 4 remediations for that path ID=(0 to 3)."

The REST API calls supported by the iRG subcomponent (see Table 5-10) are separated in three major groups:

a) *system calls* (#1-3) providing information about the status of the iRG instance and triggering operations that can be executed at any time (without disrupting the iRG's normal operation);

b) *pre-initialization calls* (#6-12) which upload the required data to generate the attack graph model; and

c) *post-initialization calls* (#4, #5 and #15-26), calls requiring a successful system initialization (via calls #13 or 14) and session tracking using cookies.

As indicated by the last two call groups, the iRG usage workflow starts with the posting of the required information by the A16 component via the pre-initialization calls; continues with the system initialization calls (#4 and #5) providing a session cookie to the caller; and finishes with the post-initialization calls which provide session-specific data (e.g. the attack graph, its remediation actions, etc.).

Table 5-10. The REST API calls supported by the iRG

#	REST Endpoint	Description	REQ_ID and Use Cases
1	GET /system/test	Test call that generates a generic response, for connectivity testing purposes.	
2	GET /system/info	Retrieves information about the iRG instance.	
3	GET /system/database/update	Updates the internal remediation DB of iRG with the most recent information from the eVDB.	FR81 UCG-18-06
4	GET /topology	Retrieves the network topology in XML form.	UCG-06-07
5	GET /topology/hosts	Retrieves the list of network hosts (incl. their security requirements).	FR55 UCG-04-02 UCG-04-03
6	POST /topology/hosts	Sets the security requirements of network hosts.	
7	POST /topology/net-ip	Sets the IP ranges (in CIDR format) of the network(s) that are considered during the network topology model construction. Used by A16 to help the iRG filter its results.	FR76 UCG-15-01
8	POST /topology/vuln-scan-report	Uploads the vulnerability scan report results. Used by A16 to report hosts' vulnerability information.	
9	POST /topology/hosts-interfaces	Uploads the host descriptions, incl. their network interface information, their IP addresses, etc. Used by A16 to report the characteristics of all network hosts.	
10	POST /topology/vlans	Uploads the description of each subnetwork, incl. the address of its gateway, its address space, etc.	

		Used by A16 to report all subnetworks covered by the smart gateway module.	
11	POST /topology/flow-matrix	Uploads a matrix describing all host connections, even across subnetworks. Used by A16.	
12	POST /topology/routing	Uploads the routing tables of all subnetworks. Used by A16 to report interconnectivity between the various subnetworks.	
13	GET /initialize	Triggers the iRG initialization procedure and generates the attack graph using the data stored in-memory (either by disk files, or by the data uploaded by calls 7-12).	
14	POST /initialize	Triggers the iRG initialization procedure and generates the attack graph using the XML already generated XML topology provided in the request.	
15	GET /attack-graph	Retrieves the MulVAL-generated attack graph.	UCG-06-07
16	GET /attack-graph/score	Returns the initial risk score of the attack graph.	UCG-15-01 UCG-15-02
17	GET /attack-graph/topological	Retrieves the topological form of the attack graph. This form presents the attack graph in terms of attacks applicable directly on network hosts and the ways an attacker may move between hosts.	UCG-06-07
18	GET /attack-graph/remediations	Get all actionable remediations (active remediation actions) for the whole attack graph. This mostly concerns the application of firewall rules to solve a part of the attack graph.	UCG-18-06
19	POST /attack-graph/remediations/ block-nodes	Get actionable remediations (active remediation actions) to block a list of attack graph nodes. This mostly concerns the application of firewall rules (as does call #18) to solve the specified nodes.	
20	GET /attack-path/list	Retrieves all the generated attack paths and their individual risk scores.	UCG-06-07
21	GET /attack-path/number	Retrieves the total number of attack paths.	
22	GET /attack-path/{id}	Retrieves a specific attack path and its individual risk score.	
23	GET /attack-path/{id}/topological	As in call #22, but in topological form.	

24	GET /attack-path/{id}/ remediations	Retrieves all remediations (active and proactive) for the specified attack path.	UCG-18-06
25	GET /attack-path/{id}/ remediation/{id}	Retrieve the details of a specific remediation action for a specific attack path.	
26	GET /attack-path/{id}/ remediation/{id}/validate	Calculate the new attack graph after the enforcement of a specific remediation action for a specific attack path.	

5.5.2 iIRS Decision Making Engine (iRE)

The communication with the iRE is performed by means of REST API call exchanging data in JSON format. The endpoints are illustrated below in Table 5-11.

Table 5-11. The REST API calls supported by the iRE

#	REST Endpoint	Description	REQ_ID and Use Cases
1	GET /parameters	Get current parameters values	UCG-04-02
2	POST /parameters	Set parameter values	UCG-04-02
3	POST /uploadTopology	Upload an attack graph for inference; Initiate decision making engine	UCG-15-02 UCG-18-05
4	GET /getBelief	Get current belief of system state	UCG-15-02 UCG-15-04
5	GET /alerts	Communication with the IDS	UCG-16-03

5.6 Technology Stack

The list of key technologies and tools utilized by all components of the iIRS—namely the *iIRS Attack Graph Generator (iRG)*, the *iIRS Decision-making Engine (iRE)* and the *iIRS Client (iRC)*—are presented in this section and are shown in Table 5-12.

Table 5-12. Technology stack used in iIRS

Tool	Version	Details	Subcomponent

Debian-based OS	Any	A minimal Ubuntu 14.04 LTS image (phusion/baseimage:0.9.16) is used to build the Docker images.	iRG Server, iRG Client
Git	Most recent	Required to clone the iIRS repository.	iRG Server, iRG Client
FIWARE CyberCAPTOR	4.4.3	The iRG Server is based on the CyberCAPTOR Server and the iRG Client is based on the CyberCAPTOR Client.	
Java 1.7	1.7.0_201	Both the iRG Server and MulVAL are coded in Java; MulVAL requires this exact version.	iRG Server, MulVAL
Apache Tomcat 7	7.0.52.0	Java servlet container providing the iRG Server REST API.	iRG Server
Apache Maven 3	3.0.5	Maven is used to manage the required Java libraries required to build the iRG Server.	iRG Server
SQLite 3	3.8.2	An SQLite DB is used to store information about vulnerabilities and their remediations.	iRG Server, Data Extraction Subsystem
MulVAL	Cyber-Trust Git repo	Generates the attack graph using a set of rules, written in Datalog, which is then parsed by the iRG Server.	iRG Server, MulVAL
XSB (Prolog/Datalog)	3.6	The Datalog engine on which MulVAL is based upon.	MulVAL
gcc, g++, make, flex, bison	Most recent	Required to build both XSB and MulVAL.	MulVAL
Data Extraction Submodule	Cyber-Trust Git repo	Required to parse and produce the XML topology files required from the iRG Server.	iRG Server, Data Extraction Submodule
Python 3	> 3.4	The Data Extraction Submodule is coded in Python.	Data Extraction Submodule
PIP for Python 3	> 1.5	Python 3 package manager.	Data Extraction Submodule
SQLAlchemy (Python Library)	0.9.4	An object-relational mapper used by the Data Extraction Submodule to manage the SQLite DB.	Data Extraction Submodule
netaddr (Python Library)	0.7.11	Provides functionality for Level 3 (IPv4 and IPv6) and Level 2 (MAC) network addresses.	Data Extraction Submodule

AngularJS (JavaScript Library)	> 1.3.15	The iRG Client makes extensive use of the Angular JS framework.	iRG Client
D3 (JavaScript Library)	Any	Provides extensive graph visualization capabilities used to produce the attack graph visualizations.	iRG Client
Bootstrap Framework	> 3.3.5	The base on which the responsive web interface of the iRG Client is built upon.	iRG Client
Flask (Python Library)	Most Recent	Used for DME API	IRE Server
graphviz	Most Recent	Used for visualization of attack graph at DME client	iRE Server, iRE Client

5.7 Physical architecture

All iIRS subcomponents are designed to be deployed as Docker images. This allows them to be completely separated and helps contain possible security attacks within each subcomponent Docker image.

- two Docker images of iRG: iRG Server and iRG Client (which constitutes the base of the iRC) and
- two Docker images of iRE: iRE Server and iRE Client, are deployed currently on two different virtual machines on the OTE testbed.

This allows further separation on the development of iRG and iRE, and proves that each subcomponent can run smoothly on independent (connected) systems.

Both machines on the OTE testbed run on Ubuntu version 18.04 LTS and are equipped to use two CPU cores, 4GiB of RAM and 32GiB of storage. Connection to the VMs on which the A16, A04g, A17, and eVDB run, is required during the initial inter-module connection tests and connection to the information bus will be a requirement during the (current) integration phase.

5.7.1 iRG Docker Images

The iRG Docker images follow the same two-stage image creation process. The first stage requires the private SSH cryptographic keys to be transferred and the official (currently private) Cyber-Trust GitLab repo is cloned. The second stage follows a similar process to the one of the FIWARE CyberCAPTOR Docker image creation, but with changes in the source of the code (as it now resides in the memory storage of the first stage), the source of dependencies (with added SHA-256 integrity checks), and the source of updated version of the Remediation DB (currently hosted by UOP).

This two-stage process ensures that the private cryptographic keys and any other sensitive artefacts aren't present in the final Docker image, the second stage of the process.

Another significant change is the addition of a deliberate way to break the Docker cache, used during development, based on the current (at the time of building the Docker image) date. This saves time when building the Docker image multiple times in succession, as the image only repeats the necessary steps to rebuild the main iRG Server Java application or assemble the iRG Client HTML, JS and CSS files.

The Docker commands to build and execute the containers follow, showcasing the usage of the deliberate Docker cache invalidation, the transfer of the private SSH keys required to clone the Git repository, and the ports each container uses (port 10000 for the iRG Server and port 8880 for the iRG Client).


```
# To clone the Cyber-Trust Git repo:
git clone git@gitlab.com:cybertrust/tool-development/intelligent-intrusion-
response.git
cd ./intelligent-intrusion-response

# To build the iRG Server:
sudo docker build \
            --build-arg CACHE_DATE="$(date)" \
            --build-arg SSH_PRIVATE_KEY="$(cat ~/.ssh/id_rsa)" \
            --build-arg GIT_BRANCH="$(git symbolic-ref --short HEAD)" \
            --tag ag-engine-server ./attack-graph-generator/server/container/

# To build the iRG Client:
sudo docker build \
            --build-arg CACHE_DATE="$(date)" \
            --build-arg SSH_PRIVATE_KEY="$(cat ~/.ssh/id_rsa)" \
            --build-arg GIT_BRANCH="$(git symbolic-ref --short HEAD)" \
            --tag ag-engine-client ./attack-graph-generator/client/container/

# To execute both containers in the background:
sudo docker run -d --name ag-engine-server -p 10000:8080 ag-engine-server
sudo docker run -d --name ag-engine-client -p 8880:80 ag-engine-client
```

5.7.2 iRE Docker Images

The deployment of the iRE components, is achieved through docker on the OTE testbed. The iIRE engine is deployed on a virtual machine. The server listens on port 17891 while the visualization content is served on port 4200. Internal communication between the iIRE server and client is performed through port 8088. The server utilizes two CPU cores and 4GB of RAM.

5.8 User Interface

The iIRS has its own, independent of Cyber-Trust's platform, user interface. The client component of iRG communicates with the iRG server. REST API calls as described in Section 5.5 offer the information needed for the visualization. The user can initialize the Attack Graph Generator by uploading the topology XML file to the home page (see Figure 5-7). When the attack graph is successfully initialized, the client responds with an appropriate message.

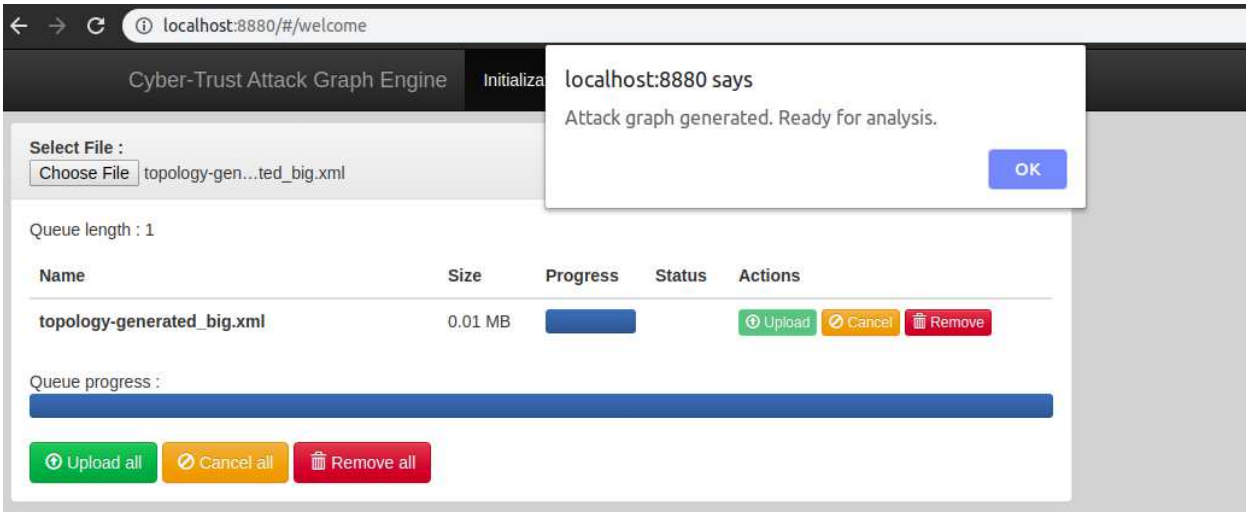

Figure 5-7. iRG Client – the initialization page

The Configuration page provides information regarding the Hosts as well as the remediation and the user can adjust the criticality associated with each machine (see Figure 5-8). Hosts were produced during the initialization procedure, with the rest of data shown in (5.8). Patch and Firewall – Rule radio buttons were a function provided by the FIWARE CyberCAPTOR project. Our remediations are done on the Server part of the iRG and Firewall Rules are proposed by the iRE.

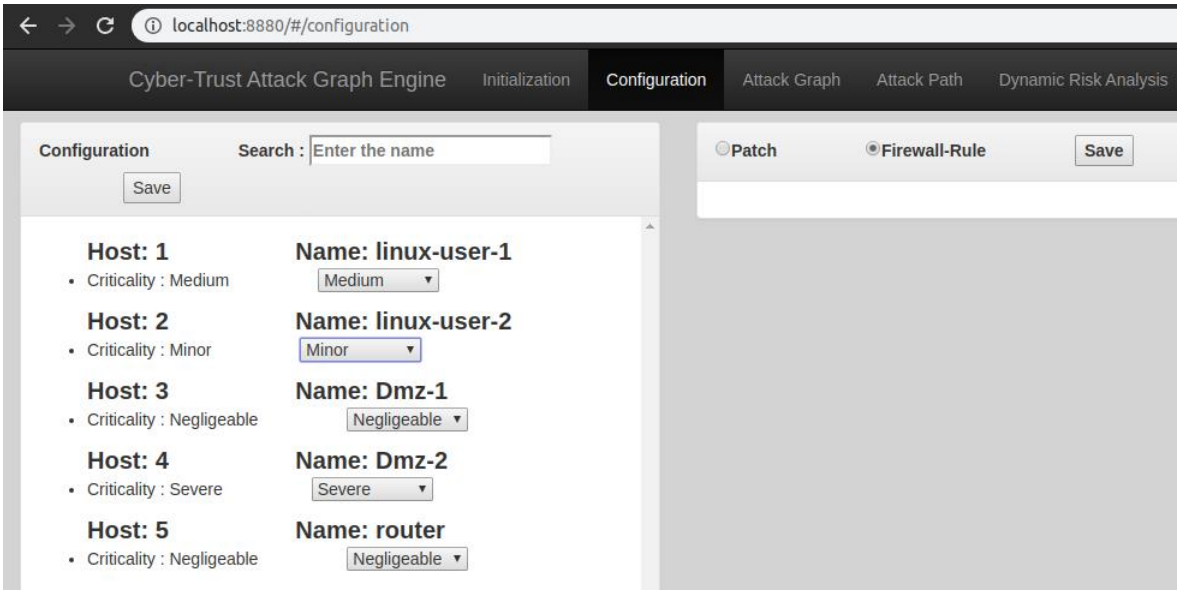

Figure 5-8. iRG Client – the configuration page

The Attack Graph page shows the topological and logical form of the network (see Figure 5-9 and Figure 5-10). In the logical form the attack graph is represented by circles of specific colors with each color being a different type of node. Orange stands for LEAF node, red for LEAF with Vulnerability, blue for AND node and light blue for OR. By hovering over a node, information such as the name, metric and rule fact can be seen.

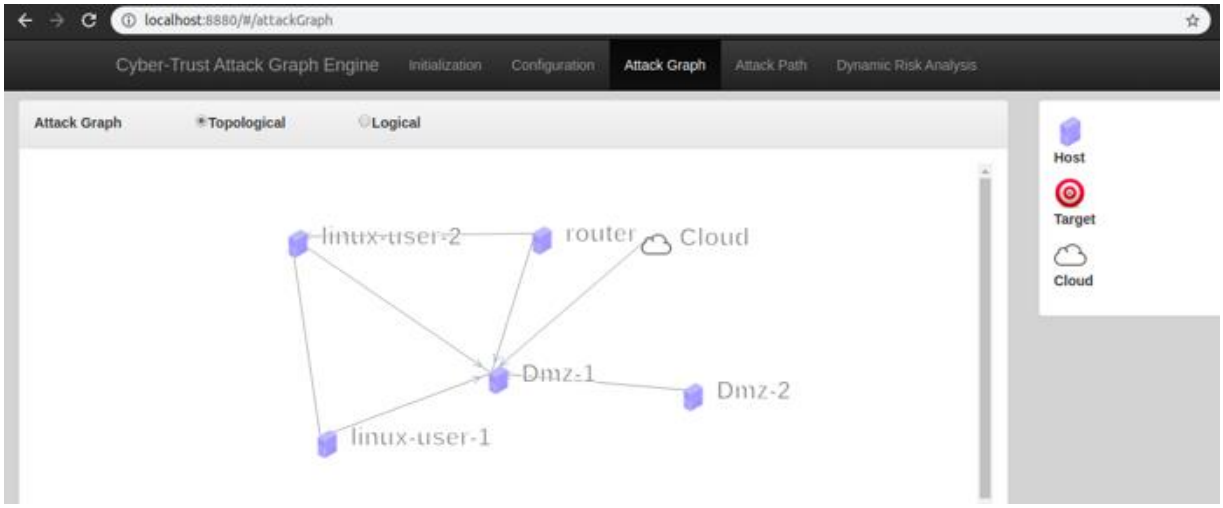

Figure 5-9. iRG Client – the topological view of the attack graph page

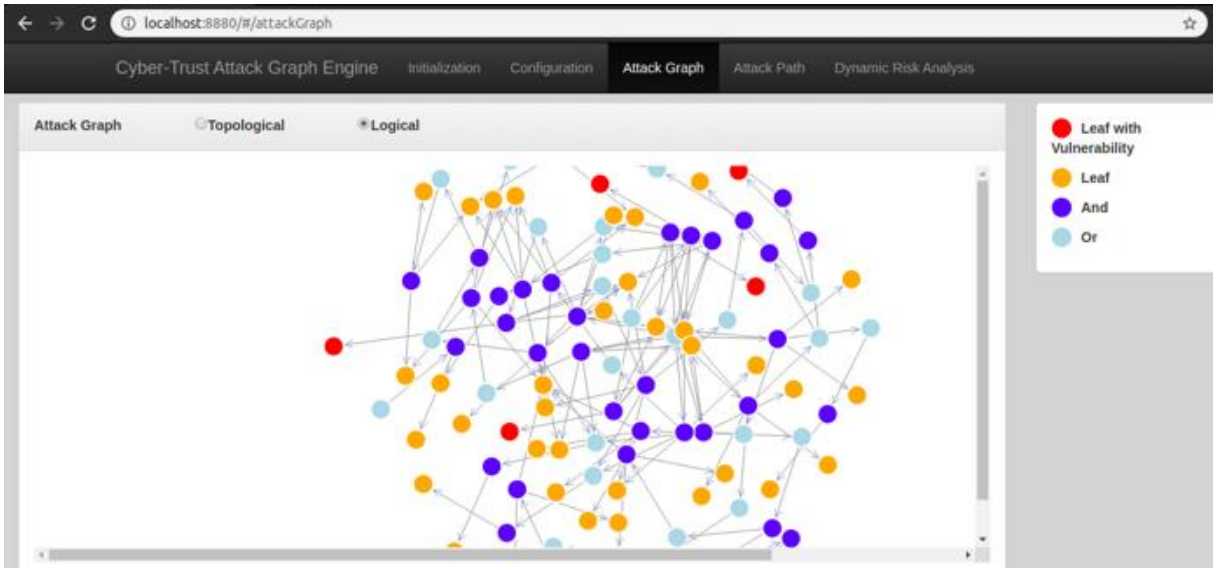

Figure 5-10. iRG Client – the topological view of the attack graph page

The Attack Path page shows almost the same data as the Attack Graph page regarding the visualization part (see Figure 5-11). In the topological form, the target machine can be seen. The user can select between the different available attack paths and there is an impact meter to measure the path's severity.

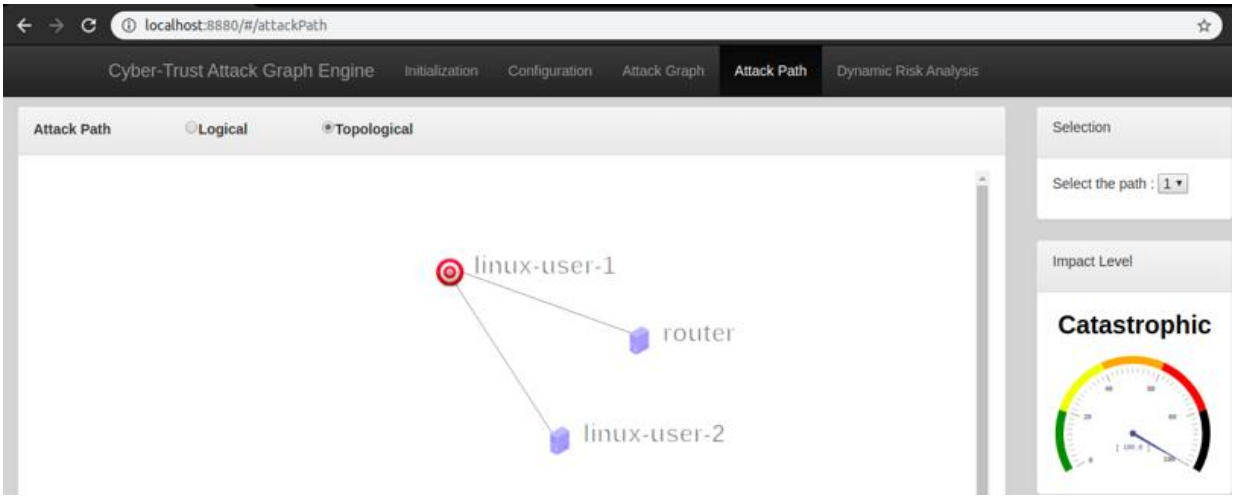

Figure 5-11. iRG Client – the attack path page

Each path has its own remediation options presented right down below the graph visualization (see Figure 5-12). The remediation may provide multiple actions to be taken, in order to prevent the attacker from reaching the goal on the associated attack path. Those actions can be either a firewall – rule or a solution provided by the NVD. As seen at 5.4.2 - remediation DB, we keep the Patch, Vendor Advisory and Third-party Advisory links regarding the vulnerabilities.

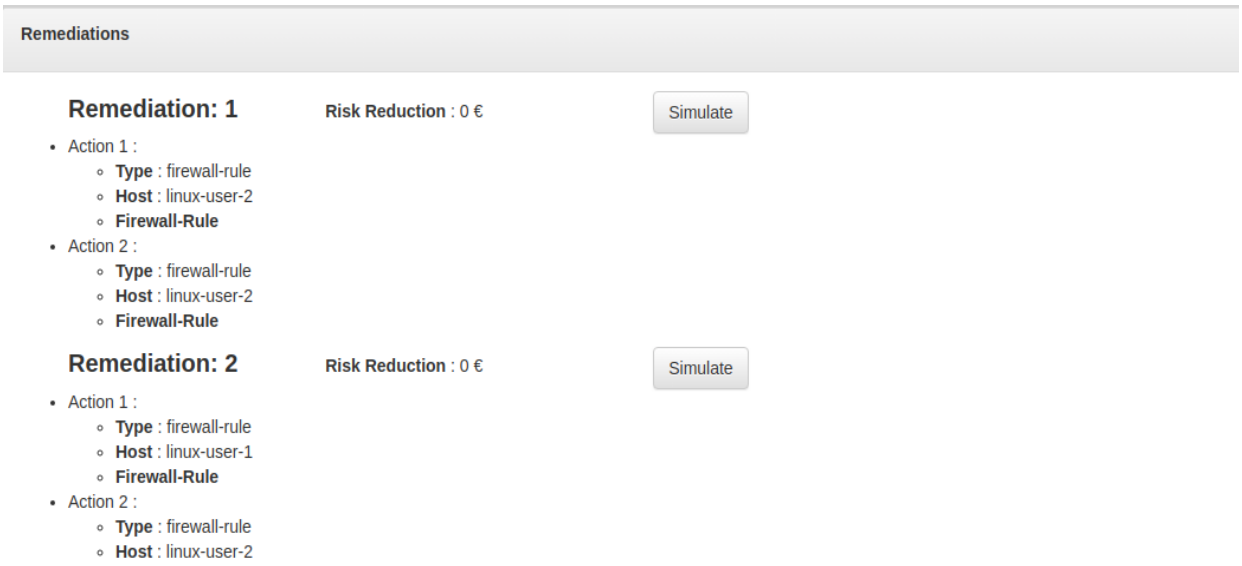

Figure 5-12. iRG Client – the suggested remediation actions

As mentioned in Section 5.3.4, the iIRS decision engine client is a simple web interface for visualizing the propagation of simulated attacks, the belief on the systems security state and the computed policy for testing and development. It only communicates with the decision making server and obtains any other relevant information about the system including security alerts and attack graph topology from there. The communication occurs through a web socket which by default is specified by the IP address of the server and by port 8088. The actual rendering of graphics is done by the server and is propagated to the client in svg format. The visualization is composed of 4 main parts (see also Figure 5-13):

- the system state and agents' actions,
- the belief state,

- the reward curves, and
- the alerts received.

When combined, these provide all interesting information regarding the state of the system for the purposes of testing and monitoring.

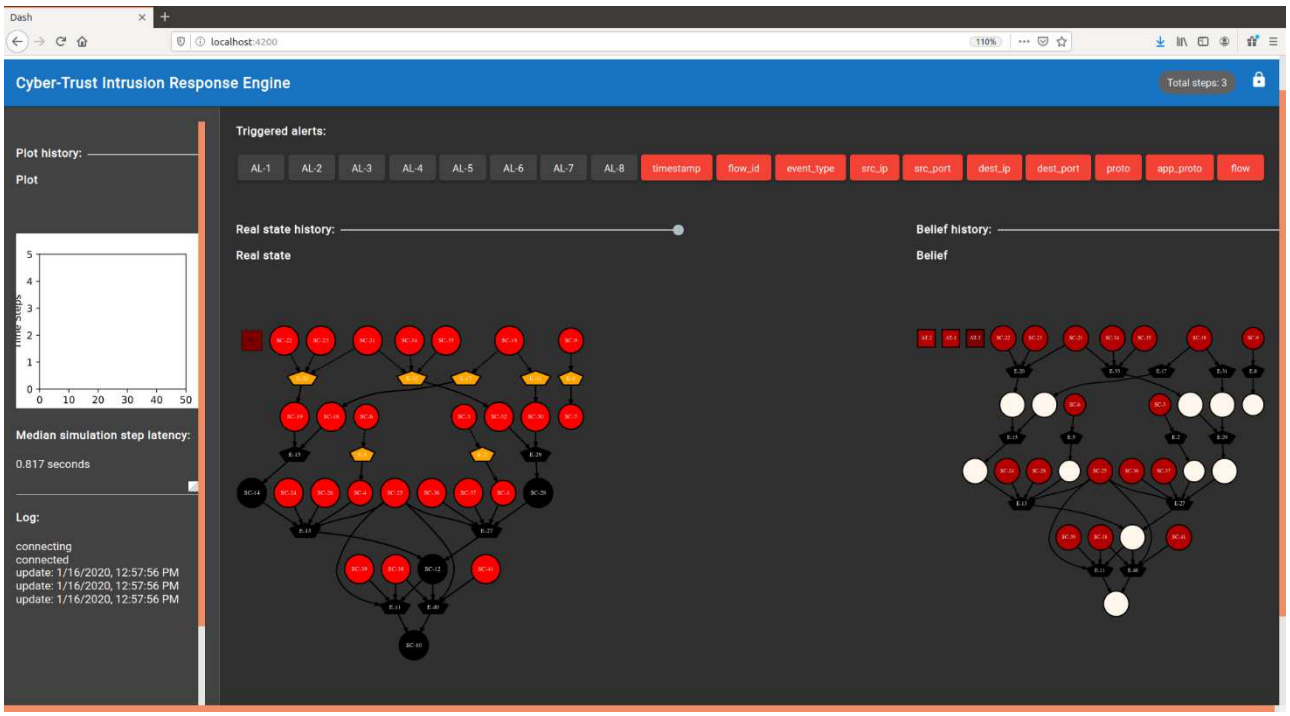

Figure 5-13. The dedicated user interface of the iRE client

On the page's main screen two visualizations of the current attack graph are displayed. The left one encodes information of the actual system security state, the attacker's actions, the decision engine's action, and the true attacker type. Circular nodes of the graph represent security condition and their colour indicates whether they are compromised. In particular, red colour indicates a compromised security condition while black colour indicates uncompromised security conditions. Pentagons represent hyperedges of the attack graph which correspond to exploits. Directed arrows display the dependencies between exploits and security conditions. The colour of the pentagons represent the attacker's and decision engine's actions. In particular, orange colour indicates an attacker attempt on an exploit whereas green colour represents an exploit blocked by the system. If a particular exploit is both attempted and blocked, this is represented with blue colour.

The right graph represents the system's belief on its security state and attacker type. Darker red colours illustrate increased confidence that a particular security condition is compromised while lighter shades indicate less confidence. The same goes for the attacker type which is represented by the squares adjacent to the belief graph.

On the left side bar a plot of the received system reward is displayed. As the decision engine takes actions during a simulated attack, instantaneous rewards are discounted and added to trace a reward curve. Steeper curves (increasing rapidly during the simulation start) indicate that the decision engine makes mistakes at early steps. The curve's asymptote is a sample (i.e. for a given trajectory) from the computed policies value. On average, the value of the asymptote will be equal to the value of the initial belief under the computed policy.

Below the reward plot some logging information is displayed. Finally, on the top of the main screen the system's alerts are displayed. Received and non-received alerts are displayed in red and grey colour respectively.

The attack visualization occurs in discrete time steps in coherence with the mathematical model of the attack which is a discrete time partially observable Markov decision process. As new steps are completed in the simulation the page automatically updates the displayed information. Slide bars are provided, which allow the user to run back to previous time steps if needed. This is very convenient for assessing the computed policy of the decision engine. One slide bar is provided above each graph and the reward plot each controlling the displayed information of the component below it. The system alerts are controlled by the Real State History slide bar located right above the left attack graph on the page's main screen.

6. Unit testing approach

Unit testing refers to the process of verifying that the individual artefacts comprising the software component operate as expected. These artefacts can be individual units of source code, sets of one or more computer programs together with associated control data, as well as usage and operating procedures. The scope of verification in unit testing should involve both the externally observable behaviour of the method and any side effects that the unit has, such as updating repositories.

The artefacts comprising a software component are classified in a number of core layers as shown below.

- *Component's REST API:* The API exposed by a Cyber-Trust component to the external world.

  The code in this layer is responsible for intercepting incoming REST API requests, extracting the input parameters from the protocol-specific message, passing the request to the appropriate business logic module (typically to the service layer), retrieving the results, packing results back into protocol-specific messages and returning the result to the requesting client.

- *Component's service layer:* Defines the component's boundary to the outside world by encapsulating the core business logic.

  Since the functionality of the component is solely exposed through the associated REST API, it is expected that there is a one-to-one mapping between operations exposed by the component's REST API and the elements exposed by the service layer.

- *Component's domain:* Contains the objects realising the business logic of the component (e.g. the attack graph object in the case of the iIRS).

- *Component's persistence:* Serves persistent domain objects to the backend of the system.

  The persistence layer manages the domain objects, which however are not necessarily all the domain objects. This layer may perform data mapping to deal with the representational differences between the repositories layer and the external data repositories (e.g. databases) where the domain objects actually persist.

- *Component's asynchronous communication layer:* Defines the push notifications sent to other components, as well push notifications from other components that are received and processed

  The asynchronous communication layer manages the creation and consumption of asynchronous notification messages, exchanged through Cyber-Trust's message bus. A module's core business logic dictates that such messages should be created when some important information about an event or a condition must be made available to other modules; conversely, when such information is needed from other modules, relevant asynchronous messages are intercepted by the communication layer and passed to the module's core business logic for processing.

Unit testing of each Cyber-Trust component included all the above layers, where the main approach taken is briefly documented in the following sections.

6.1 Unit tests for the REST API layer

The REST API layer in some of the components (e.g. the TMS) was automatically generated by an appropriate piece of software, which was employed in the modelling and development process (the Swagger modelling tool was used that generates the Spring framework skeleton code, which employs standard Spring framework libraries). In such cases, the code within the REST API layer did not require extensive testing. In other cases (e.g. the crawling service, the eVDB, and the iIRS) the REST API of the open source software tools used was extended to cover the needs of Cyber-Trust, and therefore required more thorough testing. In both cases, testing the parameter validation and the return values were found to be quite useful in order to validate that the component properly implements the documented functionality.

6.2 Unit tests for the service layer

The functionality exposed in each component's service layer was targeted by unit tests. To promote efficiency and isolation in unit testing at this level, it was recommended that any dependencies to other external services and data repositories be mocked, using stubs and pre-determined data. The unit tests developed for the service layer investigated whether the correct operation was ensured using valid data, whilst they also considered the response of a Cyber-Trust component to invalid inputs and business logic errors.

6.3 Unit tests for the domain layer

Classes and methods within the domain layer were targeted by unit tests. Typically, the classes packed within a single component have high cohesion and the operations of one class depend on other classes within the component. The approach taken during the unit testing at this layer was that such dependencies are not mocked; however, dependencies to other Cyber-Trust components were mocked. Likewise, the unit tests developed for the domain layer tested for correct operation using valid data, invalid inputs as well as business logic errors.

6.4 Unit tests for the persistence layer

Classes and methods in the persistence layer were targeted by unit tests. Each operation in the persistence layer typically requires no other information than the objects to be managed (and possibly some elementary-type parameters). Therefore, each operation in the persistence layer were tested in isolation from the other parts of a component.

6.5 Unit tests for the asynchronous communication layer

Classes and methods in the asynchronous communication layer, for the modules that such layers had been developed, were targeted by unit tests. At this stage, asynchronous communications layers of individual components were examined in isolation, with the role played by peer communication parties being mocked (i.e. fake senders and receivers were created). Tests related to asynchronous communications and jointly involving Cyber-Trust modules will be conducted at the integration phase.

7. Conclusions

This document shows the current status of the Proactive Technology tools to be integrated into an operational environment. In particular, the following tools, with a detailed technical description, have been described:

1. Crawling service
2. Enriched Vulnerability DataBase (EVDB)
3. Trust Management Service
4. Intelligent Intrusion Response

These tools aim at improving the security of the Cyber-Trust platform through the collection and aggregation of data and information from multiple sources.

It has been presented how these tools make the IoT devices network safer by preventing cyber-attacks whenever possible, and aiming to mitigate the effects of unpredictable attacks.

The integrated prototype will be piloted and tested in Task T8.3 and needed adaptations, further to the evaluation of the test will be performed.

8. References

[1] F. Bao and Ing-Ray Chen. 2012. Dynamic trust management for internet of things applications. In Proceedings of the 2012 international workshop on Self-aware internet of things (Self-IoT '12). ACM, New York, NY, USA, 1-6. DOI=http://dx.doi.org/10.1145/2378023.2378025

[2] F. Bao, I. Chen and J. Guo, "Scalable, adaptive and survivable trust management for community of interest based Internet of Things systems," 2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS), Mexico City, Mexico, 2013, pp. 1-7. doi: 10.1109/ISADS.2013.6513398

[3] R. Binnendijk, et al., "Architecture and design specifications: final", Cyber-Trust, Deliverable D4.4, 2019

[4] A. Harth, Jürgen Umbrich, Stefan Decker: MultiCrawler: A Pipelined Architecture for Crawling and Indexing Semantic Web Data. International Semantic Web Conference 2006: 258-271

[5] J. M. Hsieh, Steven D. Gribble, Henry M. Levy: The Architecture and Implementation of an Extensible Web Crawler. NSDI 2010: 329-344

[6] N. Kolokotronis, et al., "State-of-the-art on proactive technologies", Cyber-Trust, Deliverable D5.1, 2019.

[7] K. Limniotis, et al., "Threat actors' attack strategies", Cyber-Trust, Deliverable D2.5, 2018.

[8] V. Merekoulias et al., "A trust management architecture for autonomic Future Internet," 2010 IEEE Globecom Workshops, Miami, FL, 2010, pp. 616-620, doi: 10.1109/GLOCOMW.2010.5700394

[9] E. Miehling, M. Rasouli, and D. Teneketzis, "Optimal defense policies for partially observable spreading processes on Bayesian attack graphs," in Proc. 2nd ACM Workshop Moving Target Defense, 2015, pp. 67–76.

[10] E. Miehling, M. Rasouli, and D. Teneketzis, "A POMDP Approach to the Dynamic Defense of Large-Scale Cyber-Networks." IEEE Transactions on Information Forensics and Security, vol. 13, no. 10, pp. 2490-2505, 2018.

[11] M. Najork: Web Crawler Architecture. Encyclopedia of Database Systems 2009: 3462-3465

[12] T. H. Nguyen, M. Wright, M. P. Wellman, and S. Baveja, "Multi-stage attack graph security games: Heuristic strategies, with empirical game theoretic analysis," in Proc. ACM Workshop Moving Target Defense, 2017, pp. 87–97.

[13] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using Bayesian attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, Jan/Feb. 2012.

[14] S. Shiaeles, et al., "Use case scenarios", Cyber-Trust, Deliverable D2.3, 2018

[15] S. Skiadopoulos, et al., "Threat sharing methods: comparative analysis", Cyber-Trust, Deliverable D2.2, 2018

