Deliverable 7.3

Security certification methodology development

Technical References

Document version	:	1.0
Submission Date	:	31/08/2022
Dissemination Level	:	Public
Contribution to	:	WP7 – Security and Privacy Claims
Document Owner	:	UMU
File Name	:	BIECO_D7.3_31.08.2022_V1.0
Revision	:	3.0

Project Acronym	:	BIECO
Project Title	:	Building Trust in Ecosystem and Ecosystem Components
Grant Agreement n.	:	952702
Call	:	H2020-SU-ICT-2018-2020
Project Duration	:	36 months, from 01/09/2020 to 31/08/2023
Website	:	https://www.bieco.org

This project has received funding from the European Union´s Horizon 2020 Research and Innovation Programme under Grand agreement No. 952702.

Revision History

REVISION	DATE	INVOLVED PARTNERS	DESCRIPTION
0.0	10/08/2020	7Bulls	Creation of the document
0.1	15/08/2020	7Bulls	Initial draft
1.0	10/05/2022	UMU	New table of content
1.1	13/06/2022	GRAD	Contribution to subsection 3.3.2
1.1	22/06/2022	CNR	Contribution to (sub-)section 3.3.3
1.2	23/06/2022	UMU	Contribution to sections 3, 4, 5
1.3	04/07/2022	CNR	Finalizing (sub-)section 3.3.3 and contribution to section 3.9 (Communication and Auditing)
1.4	06/07/2022	UMU	Added annex with claims
1.5	07/07/2022	UMU	Added text to section 4
1.6	07/07/2022	GRAD	Contribution to sections 3 and 4
1.7	08/07/2022	CNR	Contribution to sections 3.5 and 7
1.8	08/07/2022	IESE	Contribution to section 3.2 and 5
2.0	15/07/2022	UMU	Edition of deliverable
2.1	20/07/2022	HS, TTT	Review of deliverable
2.2	27/07/2022	UMU	Final edit of deliverable addressing reviewers' comments
2.3	29.08.2022	UNINOVA	Review and edition of Coordinator
3.0	31.08.2022	UNINOVA	Finalization and Submission by Coordinator

List of Contributors

Sara N. Matheu (UMU), Marcin Byra (7bulls), Adrián Sánchez (UMU), Mónica Alonso (GRAD), Eva Sotos (GRAD), Javier Yépez (GRAD), Said Daoudagh (CNR), Eda Marchetti (CNR), Antonello Calabrò (CNR), Enrico Schiavone (RES), Enrico Araniti (RES), Ioannis Sorokos (IESE)

Reviewer(s): Oliviu Matei (HS), Mohammed Abuteir (TTT), Sanaz Nikghadam-Hojjati (UNINOVA), José Barata (UNINOVA)

BIECO project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No **952702**.

Acronyms

Acronym	Term
ABAC	Attribute-Based Access Control
AIAG	Automotive Industry Action Group
ANSI	American National Standards Institute
ANSSI	National Cybersecurity Agency of France
APG	Attack Path Graph
APT	Attack Path Tree
ASIL	Automotive Safety Integrity Level
CAPEC	Common Attack Pattern Enumeration and Classification
C2C-CC	Car 2 Car Communication Consortium
CC	Common Criteria
CCRA	Common Criteria Recognition Arrangement
CEP	Complex Event Processor
CESG	Communications-Electronics Security Group
CFT	Component Fault Trees
CHASSIS	Combined Harm Assessment of Safety and Security for Information Systems
CPA	Commercial Product Assurance
cPP	Collaborative Protection Profiles
CSA	Cybersecurity Act
CSPN	Certification de Sécurité de Premier Niveau
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
CWRAF	Common Weakness Risk Analysis Framework
CWSS	Common Weakness Scoring System
DCT	Data Collection tool
DM	Data Mining
D-MUC	Diagrammatic Misuse Cases
DoS	Denial of Service
DPIA	Data Protection Impact Assessment
DREAD	Damage potential, Reproducibility, Exploitability, affected users, Discoverability
DSL	Domain Specific Languages
D-UCs	Diagrammatic Use Cases
EAL	Evaluation Assurance Levels
EC	European Commission
ECSC	European Cyber Security Certificate
ECSO	European Cyber Security Organisation
ENISA	European Union Agency for Network and Information Security
ETSI	European Telecommunications Standards Institute
EU	European Union
EUCC	Common Criteria based European candidate cybersecurity certification scheme
EUCS	European Cybersecurity Certification Scheme for Cloud Services
EVITA	E-safety Vehicle InTrusion protected Applications
FMEA	Failure Mode and Effect Analysis
FMVEA	Failure Mode, Vulnerabilities and Effects Analysis
FSD	Failure-Sequence Diagrams

GDPR	General Data Protection Regulation
GROOT	GdpR-based cOmbinatOrial Testing
HARA	Hazard Analysis and Risk Assessment
HARM	Hailstorm Application Risk Metric
HAZOP	Hazard and Operability studies
HEAVENS	HEAling Vulnerabilities to ENhance Software Security and Safety
IACS	Industrial Automation and Control Systems
ICT	Information and Communications Technology
IEC	International Electrotechnical Commission
ISA	International Society of Automation
ISO	International Organization for Standardization
ITU	International Telecommunication Union
LMB	Left Mouse Button
ML	Machine Learning
MBST	Model-Based Security Testing
MBT	Model-Based Testing
MIA	Model Inference Algorithm
MITM	Man In the Middle
MRA	Mutual Recognition Agreement
MUD	Manufacturer Usage Description
MUSD	Misuse sequence Diagrams
NCCA	National Cybersecurity Certification Authority
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
OCTAVE	Operationally Critical Threat, Asset, and Vulnerability Evaluation
OEM	Original Equipment Manufacturer
OSSTMM	Open-Source Security Testing Methodology Manual
OWASP	Open Web Application Security
OWASP	Open Web Application Security Project
PP	Protection Profile
QR	Quick Response
RTU	Remote Terminal Units
RUMI	Relied Upon Message Interface
SAHARA	Security-Aware Hazard Analysis and Risk Assessment
SANS	SysAdmin, Audit, Network and Security
SAST	Static Application Security Testing
SCA	Static Code Analysis
SCADA	Supervisory control and data acquisition
SDLA	Security Development Lifecycle Assurance
SecL	Security Level
SOG-IS	Senior Officials Group Information Systems Security
SOTA	State Of The Art
SSA	System Security Assurance
ST	Security Target
STAMP	System Theoretic Accident Model and Processes
STPA	System Theoretic process Analysis
STPA-Sec	Systems-Theoretic Process Analysis for Security
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, elevation of Privileges
SUT	System Under Test

TAL	Trust Assurance Levels
T-MUC	Textual Misuse Cases
TOE	Target of Evaluation
TP	Tolerance Profiles
TTCN	Testing and Test Control Notation
T-UCs	Textual Use Cases
TVRA	Threat, Vulnerability, and Risk Analysis
UL	Underwriters Laboratories
UML	Unified Modelling Language
UNECE	United Nations Economic Commission for Europe

Executive Summary

This deliverable reports the work done in T7.3, whose purpose is the instantiation of the methodology defined in Task T7.2. To this end, tools developed and improved within WP3, WP4, WP5, WP6 and WP7, to identify, model and measure the risk associated with each threat or vulnerability, are used to instantiate the different phases of the methodology. The main objective of this task is to provide an instantiation of an automated and evidence-based security assessment integrating BIECO tools to support the different phases.

Project Summary

Nowadays most of the ICT solutions developed by companies require the integration or collaboration with other ICT components, which are typically developed by third parties. Even though this kind of procedures are key in order to maintain productivity and competitiveness, the fragmentation of the supply chain can pose a high-risk regarding security, as in most of the cases there is no way to verify if these other solutions have vulnerabilities or if they have been built taking into account the best security practices.

In order to deal with these issues, it is important that companies make a change on their mindset, assuming an "untrusted by default" position. According to a recent study only 29% of IT business know that their ecosystem partners are compliant and resilient with regard to security. However, cybersecurity attacks have a high economic impact, and it is not enough to rely only on trust. ICT components need to be able to provide verifiable guarantees regarding their security and privacy properties. It is also imperative to detect more accurately vulnerabilities from ICT components and understand how they can propagate over the supply chain and impact on ICT ecosystems. However, it is well known that most of the vulnerabilities can remain undetected for years, so it is necessary to provide advanced tools for guaranteeing resilience and also better mitigation strategies, as cybersecurity incidents will happen. Finally, it is necessary to expand the horizons of the current risk assessment and auditing processes, taking into account a much wider threat landscape. BIECO is a holistic framework that will provide these mechanisms in order to help companies to understand and manage the cybersecurity risks and threats they are subject to when they become part of the ICT supply chain. The framework, composed by a set of tools and methodologies, will address the challenges related to vulnerability management, resilience, and auditing of complex systems.

Partners

Disclaimer

Table of Contents

List of Figures

List of Tables

1. Introduction

Continuous technological advances will enable the development of new ICT systems, shaping innovative digital ecosystems for the benefit of society. As recognized by the European Union (EU) cybersecurity regulation "Cybersecurity Act" ("CSA"), this requires that certification schemes provide a high level of flexibility to adapt to a changing technological environment to avoid the risk of becoming outdated. The aim of task T7.2 was to define a security evaluation methodology as a basis for certification, flexible enough to give to the security evaluator the freedom to instantiate the different steps using any kind of technique or tool.

Starting from that point, this deliverable proposes a concrete instantiation of the security evaluation methodology supported by the tools and methodologies developed in WP3, WP4, WP5, WP6, and WP7. Each phase of the methodology is described and evaluated using tools to identify, test and measure the risk, based on the security claims established. Therefore, the automated and evidence-based security assessment is performed, integrating BIECO tools and providing a final result of the methodology evaluation.

This document draws the conclusions of the work developed in WP7, which began with the identification of a basic set of security and privacy claims (D7.1[1]), continued with the definition of a high-level security evaluation methodology (D7.2[2]) and now ends with an instantiation proposal within the BIECO project (D7.3).

This deliverable is organized as follows. **Chapter 2** provides a short overview of the security evaluation methodology developed in D7.2, explaining the steps and introducing some basic definitions used in the document. **Chapter 3** presents the main goal of this task: the instantiation of the methodology, describing each phase. Starting with establishing the context (claims and labels), goes through risk identification (system description, sensitivity calculation, test prioritization, vulnerability identification, safety impact calculation) and security testing (GraphWalker, Fuzzing, GROOT). Next, it presents risk estimation and risk evaluation phases (containing a full description of risk calculation), and finally, the labelling, treatment and continuous communication and auditing.

Chapter 4 is a proof-of-concept description, based on UC4. In other words, it presents the instantiation of the BIECO methodology on the use case developed by UNINOVA. **Chapter 5** describes how the methodology can be adapted in different scenarios that could arise within the supply chain scenario and the certificate composition, and **Chapter 6** summarizes the document. Moreover, there are two annexes: one describes all the claims related to each use case, and the other one is a technical description of the risk evaluation tool, SecurityScorer.

[1] Deliverable 7.1: "Report on the identified security and privacy metrics and security claims to evaluate the security of a system"
[2] Deliverable 7.2: "Security certification methodology definition"

2. Security Evaluation Methodology Overview

The security evaluation methodology developed in D7.2 defines a set of steps to evaluate the security of a system (Figure 1). It builds a framework on top of two main streams: security testing to identify security vulnerabilities and security risk assessment to measure the associated risk.

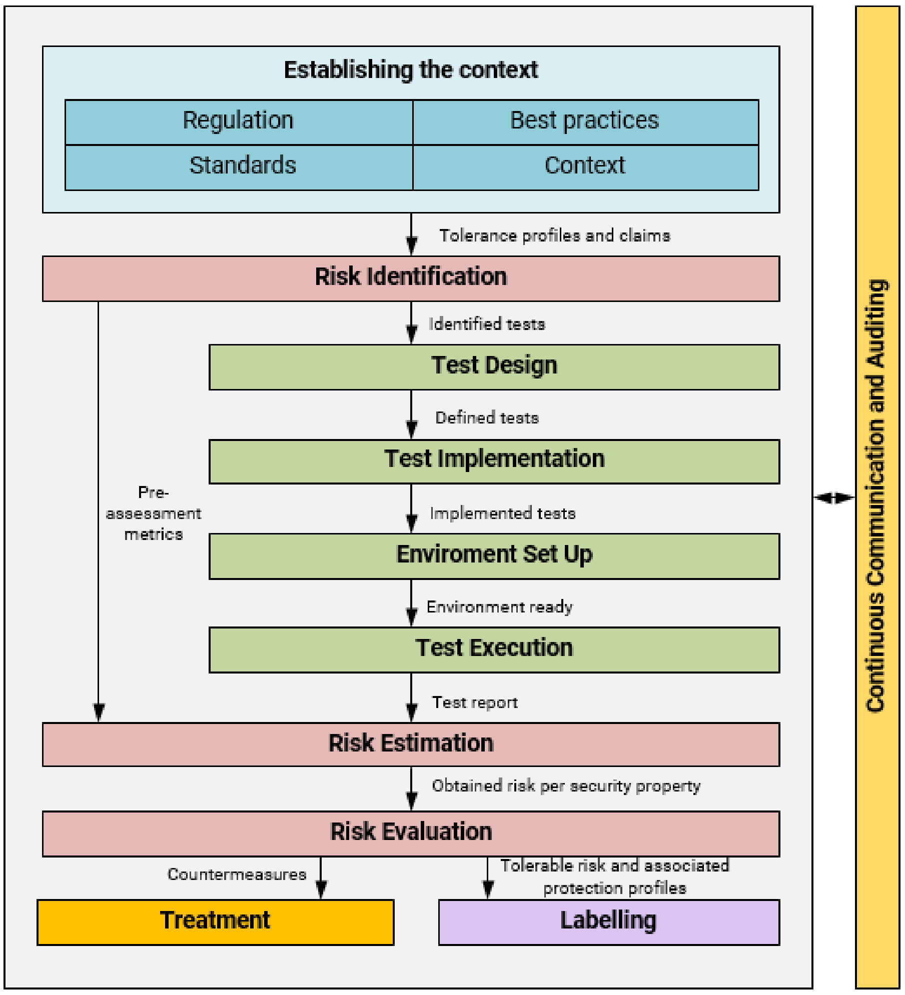


Figure 1 Security evaluation methodology

The first phase is the **context** phase, which considers the existing regulation, the best practices, current standards etc. to build an initial set of security claims that can be used as starting point for the security evaluation.

From this initial set and taking into account the particular Target of Evaluation (TOE), in the **risk identification** phase, a set of applicable threats can be selected. This set can be also extended with specific threats not considered in the initial set, by examining the special characteristics of the TOE.

Once the threats are selected, we start the **security testing** block. The **test implementation** phase deals with the design and implementation of the tests necessary to verify if the system is vulnerable to these threats. The needed entities and context to execute the tests is established in the **environment set up** phase and the tests are executed in the **test execution** phase, generating at the end of the process a test report.

The test results of the previous phase are used to estimate the risk of every component of the TOE during the **risk estimation** phase. Towards this end, information from the risk identification phase is required, regarding the components, the identified threats, and their impact. The **overall risk evaluation** phase combines the risk coming from every component, obtaining an overall measure of the system security.

At the end of the evaluation process, a label showing the evaluation results is generated. Other actions are also possible to mitigate the security flaws encountered during the process. Additionally, the methodology also considers a transversal and supportive process for **continuous communication and auditing** meant to deal with the lifecycle management of the TOE.

This deliverable focuses on the instantiation of these steps within the BIECO framework, using tools and methodologies developed within the project. Next chapter details each of the steps of the methodology and how the instantiation has been performed.

3. Security Evaluation Methodology Instantiation

This section presents the instantiation of the different steps of the methodology using specific methodologies, tools, and techniques from BIECO. It is worth noting that at the date of finishing WP7, some of the tools that are considered within the methodology are still under development, so the integration will be completed within WP8. Moreover, it should be noted that the methodology is intended to be generic, and that the purpose of the proposed tools for the instantiation is to support the user in following such methodology, so all the tools are mandatory to be used. This will be reflected in the possible workflows offered by the BIECO platform (WP8). Finally, the methodology can be also performed manually, and additional or alternative tools can be used instead.

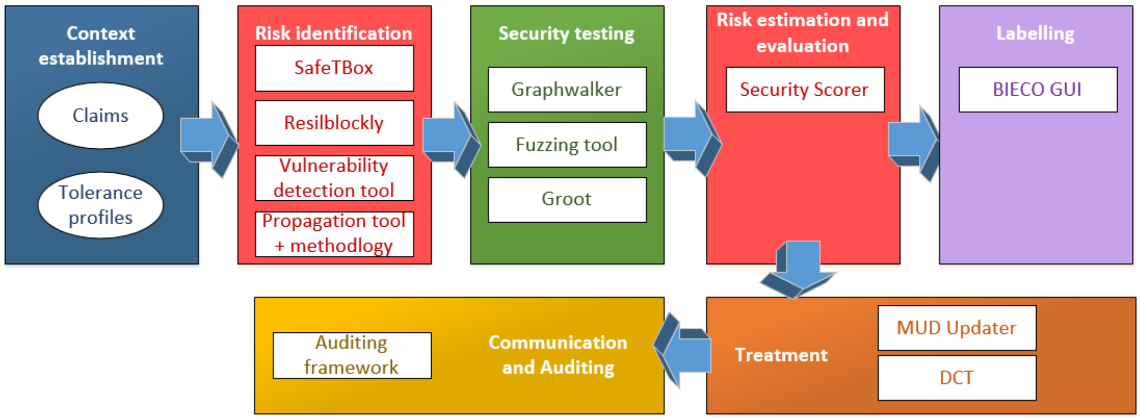


Figure 2 Overview of the BIECO tools proposed for the methodology instantiation

Figure 2 shows an overview of the tools developed and improved within BIECO WPs that have been considered within the methodology instantiation. Next subsections detail for each phase, which tools and methodologies can be used to support the related activities and how to integrate their outputs in the security evaluation methodology.

3.1. Establishing the Context

In this phase we establish the basis for the evaluation, answering the question *what we are going to evaluate?* In particular we consider three different sources within BIECO framework: the security and privacy claims obtained from current standards, best practices and regulation and the tolerance profiles, which reflects the security level that should be achieved by the System Under Test (SUT).

3.1.1. The Security and Privacy Claims

The security and privacy claims are the basis on which to evaluate the SUT, since they represent the security properties that the system must meet in order to be certified. The claims can be given by the owner of the SUT in order to demonstrate its compliance with basic security principles or even with a particular security standard, or they could be given by a third party that needs the SUT to comply with certain claims in order to be integrated into her/his system, thus guaranteeing the security of the whole ecosystem.

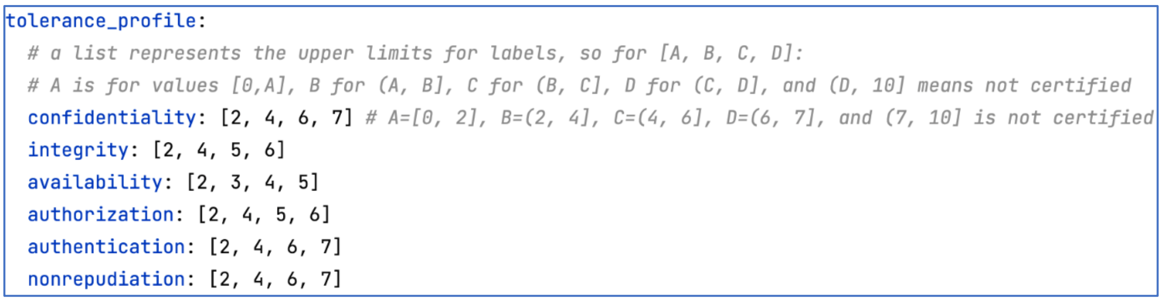
In BIECO, the claims defined in D7.1 are used as a basis for making a selection on which to evaluate the SUT, and are extended as necessary with additional requirements imposed by the client. Annex 1 shows the claims that have been selected by the use case owners (ICT GW, MICROFACTORY – FIRMWARE UPDATE, AI INVESTMENT) for the security evaluation.

3.1.2. The Tolerance Profiles

The tolerance profiles indicate, based on the particular context of the system, to what extent the selected claims must be fulfilled, that is, what is the acceptable risk to be certified and what security levels are established within said limits.

In BIECO, the tolerance profiles are established by the client and used as input for the evaluation process. As the tolerance profile influences the certified security level, the profile must always be part of the issued certificate (linked to the label), giving more detailed information about the process and facilitating future compositions (see Section 5).

The tolerance profile in BIECO follows a YAML format presented in Figure 2. This is only a part of the system description file which will be explained in Section **Error! Reference source not found.** below.



```
tolerance_profile:
  # a list represents the upper limits for labels, so for [A, B, C, D]:
  # A is for values [0,A], B for (A, B], C for (B, C], D for (C, D], and (D, 10] means not certified
  confidentiality: [2, 4, 6, 7] # A=[0, 2], B=(2, 4], C=(4, 6], D=(6, 7], and (7, 10] is not certified
  integrity: [2, 4, 5, 6]
  availability: [2, 3, 4, 5]
  authorization: [2, 4, 5, 6]
  authentication: [2, 4, 6, 7]
  nonrepudiation: [2, 4, 6, 7]
```

Figure 3 Tolerance profile example

The example in Figure 3 is the YAML representation of the tolerance profile defined in deliverable D7.2, Section 6.1, Figure 2. Each type of risk (confidentiality risk, integrity risk, etc.) has four labels A, B, C, D, each covering a disjoint part of the [0,10] range. For example, a list of values [2, 4, 6, 7] is assigned to the confidentiality risk of the Smart Car Profile. It means that the creator of the profile treats the result in the [0,2] range as the highest possible mark, hence A label. Then, the result in range (2, 4] is labelled as B, the result in range (4, 6] is labelled as C, and the result in (6, 7] gets D. Finally, if the confidentiality result is above 7 then it is not certified.

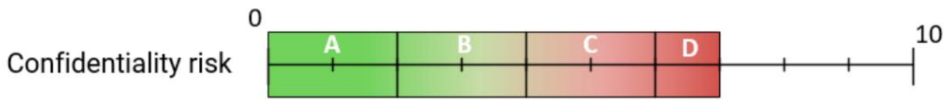


Figure 4 Example confidentiality risk profile

Figure 4 shows the confidentiality risk of the smart car profile in a visual form to fully explain the encoding of the YAML representation.

3.2. Risk Identification

The risk identification phase focuses on describing the system, its components and its degree of dependency, and identifying possible vulnerabilities that may be present in them through a preliminary analysis. The identified vulnerabilities are associated with the corresponding vulnerability claims, and during the testing phase it will be verified whether they are present in the system or not.

In the proposed instantiation the needed information comes from different sources:

- The system description is provided by the user in a YAML file. The YAML file includes the sensitivity parameter that reflects the degree of dependency of each system component. For the calculation of this value, the user can follow the methodology proposed in D3.4[3] and the results of the WP3 propagation tool.
- The list of possible vulnerabilities and their associated impact can be obtained from the WP6 Resilblockly preliminary analysis, which additionally includes mechanisms to integrate the propagation of vulnerabilities in the calculation of the impact using attack paths analysis. Moreover, the safety impact dimension can be refined in critical systems using SafeTBox tool, also developed within WP6.

Additionally, the tests to design and execute can be prioritized based on the results of the vulnerability detection tool from WP3.

Next subsections provide additional details about how the information is presented, integrated and used inside the methodology.

3.2.1. System Description

In the methodology, the system should be decomposed in its applicable security properties and affected components. Figure 5 shows the UML formal representation of this decomposition.

[3] Deliverable 3.4:" Report of the Tools for Vulnerability Propagation"

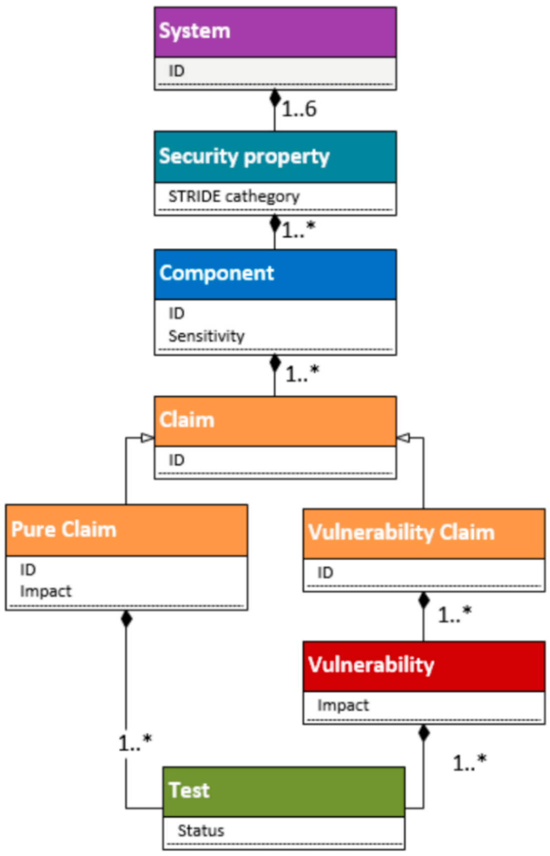


Figure 5 Decomposition of the system in the risk identification phase

Each system component will have a list of claims to check; claims that can be pure, if they are directly linked with tests or vulnerability claims if they are related to the presence of knows vulnerabilities, which are empirically verified through tests. At the end of the system decomposition, we have the tests that should be designed and implemented in the security testing phase. Additional information can be found in D7.2.

The system description is introduced into the methodology in the form of a YAML file (Figure 6, left). The tolerance profile scheme introduced in Section 3.1.2 is only a part of the file. Another essential part is the decomposition of the system as presented in Figure 5. The figure and the reasoning behind it were introduced in Deliverable 7.2 (Section 6.2 Risk Identification) along with the definitions of sensitivity, impact, etc. However, it is presented here for the ease of understanding the YAML scheme.

```
Methodology:yaml
+-tolerance_profile
|          +-confidentiality: table[int]
|          +-integrity: table[int]
|          +-availability: table[int]
|          +-authorization: table[int]
|          +-authentication: table[int]
|          +-nonrepudiation: table[int]
+-components
|          +-Component [name]*
|                     +-sensitivity: double
+-claims
|          +-claim [name]*
|                     +-component: string
|                     +-security_properties
|                     |          +-security_property [name]
|                     +-reference?: string
|                     +-impact?: double
|                     +-tests?
|                     |          +-test [name]*
|                     +-vulnerabilities?
|                                +-vulnerability [name]*
+-vulnerabilities
           +-vulnerability [name]*
                      +-reference: string
                      +-impact: double
                      +-tests
                                 +-test [name]*
```

```yaml
tolerance_profile:
  ...

components:
  component_1:
    sensitivity: 5.0
  component_2:
    sensitivity: 7.0

claims:
  # pure claim
  claim_c49:
    component: component_1
    security_properties:
      - authentication
    reference: C49
    impact: 1.0
    tests:
      - NewOwnerTest
  # vulnerability claim
  claim_vuln_auth:
    component: component_2
    security_properties:
      - integrity
    vulnerabilities:
      - vuln_integrity

vulnerabilities:
  vuln_integrity:
    reference: oddone
    impact: 3.0
    tests:
      - FindOwnersTest
```

Figure 6 YAML file format and example

Figure 6 (right) is a listing with an example system description file. The tolerance profile part is hidden due to the fact it was already discussed. The system description part is organised as follows (note that what is called below a *list* is formally a YAML map, but we intend to keep the vocabulary simple).

- *Tolerance profile*
- *Components* – a list of component names. Each component has a numeric property: sensitivity with a numeric value between 0 and 10. Subsection 3.2.2 details how to calculate this value through the methodology developed in T3.4.
- *Claims* – a list of claims, both "pure" claims and the claims with associated vulnerabilities (see Figure 6 and Deliverable 7.2). Each claim has:
    - a reference to a component,
    - a list of security properties it is related to,

  Moreover, each "pure" claim has:

    - A reference to the claim ID in documentation

    - An impact value obtained from 4 different dimension (safety, operational, financial and privacy & legislation).
    - A list of related tests

  The claims with vulnerabilities have only a list of vulnerabilities instead.

- *Vulnerabilities* – a list of vulnerabilities. Each vulnerability has:
    - A reference to a vulnerability ID in documentation
    - An impact value
    - A list of related tests

The scheme allows to conveniently pass the system description and tolerance profiles to the BIECO software. It is then used by the security scorer tool to internally build a system schema and, by combining this information with the outputs of the risk assessment and security testing phases, calculate the numerical value of the risk. It is comprehensively described in Sections 3.6 and 3.7 of this deliverable.

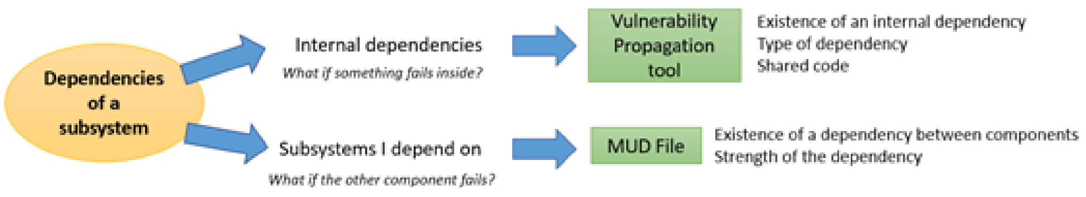
3.2.2. Sensitivity Calculation

Current systems are made up of different interconnected components that work together to provide a service. A *component* is defined as *a static building block of a system which can be a module, a class or interface, a package, or a subsystem* [21]. In this context, a failure in one of the system components, may have cascade effects over other components, or even produce a generalized system failure. Therefore, analyzing the existing dependencies among the system components and its degree, can help to determine the impact that a vulnerability would have over the rest of the system components.

As advanced in the previous section, each system component has a numeric property: sensitivity with a numeric value between 0 and 10, which measures the degree of dependency with other system components.

While this measure can be manually introduced by the user, the BIECO framework provides tool and methodology support for its calculation, developed within WP3, T3.4.

On the one hand, the methodology developed in T3.4 for the measurement of the system dependencies consider internal dependencies, obtained from the vulnerability propagation tool (T3.4). In particular, the inputs needed are the total code, the code shared between classes or entities (that can be indirectly obtained by analysing the relations) and the type of relationship they have (inheritance, composition, aggregation, etc.). All these values can be obtained, directly or indirectly, from the propagation tool report. On the other hand, the methodology considers the external dependencies, obtained from the MUD (Manufacturer Usage Description) file specified behaviour (WP6), in terms of offered services and network accesses.



Dependencies of a subsystem
Internal dependencies
What if something fails inside?
Vulnerability Propagation tool
Existence of an internal dependency
Type of dependency
Shared code
Subsystems I depend on
What if the other component fails?
MUD File
Existence of a dependency between components
Strength of the dependency

Figure 7 Dependencies of a subsystem

At the end, the total degree of dependency of a component N is measured as

$$D_N = \max_{0 \le i \le n} \{intD_N, D_1, .., D_s\}$$

Where *D1, D2, …, Ds* are degree of dependency of the components that the component N depends on.

3.2.3. Test Prioritization

One of the characteristics of the cybersecurity evaluation is that it can never be guaranteed that a system is 100% secure since the time spent in evaluating it completely may not be feasible. In this sense, knowing in advance which claims may or may not be fulfilled with a certain level of certainty can shorten the evaluation and make it necessary to perform fewer tests. Knowing this premise, in BIECO WP3 it is proposed a tool to detect existing vulnerabilities within the source code from different code languages with a certain degree of confidence. In this sense, if the tool detects that a vulnerability associated to a claim could be present in the system with a very high confidence, the tests associated to this claim can be skipped or moved to the end of the priority list, as they are very likely to be failed.

The tool uses as input data the source code to be analyzed and the type of programming language in which the source code was developed. By means of ML (Machine Learning) algorithms, the vulnerability detection tool detects the existence of possible not registered vulnerable code within a module or component of the same. The use of ML algorithms provides a trust associated with said detection which will help the analyst to assess the prioritization of the tests to perform such as the ones referring the detection of vulnerabilities within the source code. These claims (from D7.1) are:

- Claim 28: The source code must not contain SQL injection vulnerabilities
- Claim 29: The source code must not contain command injection vulnerabilities
- Claim 30: The source code must not contain code injection vulnerabilities
- Claim 31: The source code must not contain path traversal vulnerabilities
- Claim 32: The source code must not use components with known vulnerabilities

Due to the fact that the development of the tool is still ongoing, further information will be provided in subsequent deliverables (D3.5[4]). Furthermore, its integration with the different tools and methodologies will be developed and analyzed throughout the duration of WP8.

[4] Deliverable 3.5: "Update Report of the tools for vulnerability detection and forecasting"

3.2.4. Vulnerability Identification

While pure claims are directly associated with tests, vulnerability claims are associated to the existence of specific vulnerabilities from CVE. Even if this process can be manually performed, BIECO has available the ResilBlockly tool developed within WP6, which among its outputs it gives the lists of possible vulnerabilities and weaknesses associated to each system component. The button to access this functionality is depicted as an exclamation mark in a yellow triangle (see Figure 8).

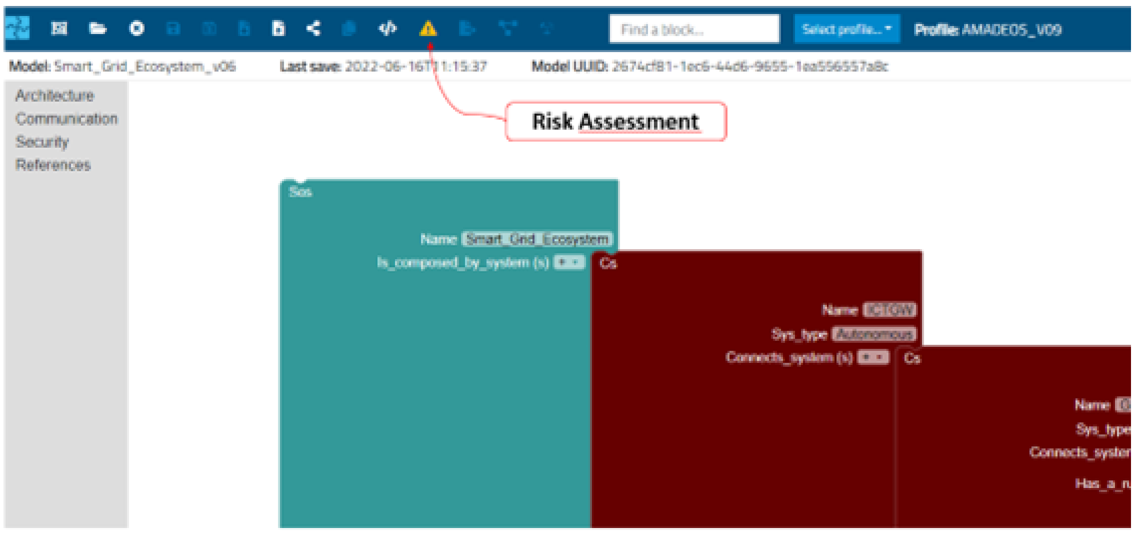
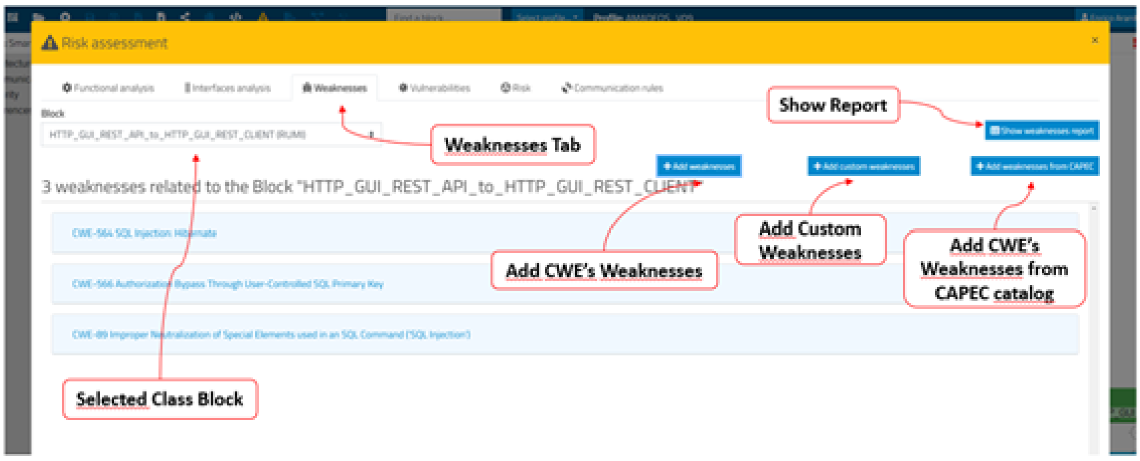


Figure 8 Risk Assessment functionality in Model Designer

In order to identify threats, attacks and vulnerabilities that apply to each asset, and to reduce the intrinsic difficulty of this process, ResilBlockly leverages the MITRE lists of known threats, and in particular:

- CWE (Common Weakness Enumeration) catalogue [116] for the weaknesses.
- CVE (Common Vulnerabilities and Exposure) [113] and NVD (National Vulnerability Database) [115] catalogs for the vulnerabilities.
- CAPEC (Common Attack Pattern Enumeration and Classification) catalog [114] for the attack pattern.

The association of **weaknesses** is provided to the user of ResilBlockly Model Designer and allows to perform the identification of weaknesses and their association with the *Class Blocks* of the Model. After having modelled a system, the identification of weaknesses can be initiated in the *Model designer* by clicking on the Risk Assessment icon and then choosing the *Weaknesses* tab (as shown in Figure 9).



3 weaknesses related to the Block "HTTP_GUI_REST_API_to_HTTP_GUI_REST_CLIENT"

CWE-564 SQL Injection: Hibernate

CWE-566 Authorization Bypass Through User-Controlled SQL Primary Key

CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Figure 9 Weaknesses tab in Risk Assessment

The tool allows the choice of *Class Blocks* (that will implicitly be considered as assets) and the association of weaknesses to each of them. In the example of Figure 9 the block chosen is the RUMI (Relied Upon Message Interface) called *HTTP_GUI_REST_API_to_HTTP_GUI_REST_CLIENT*.

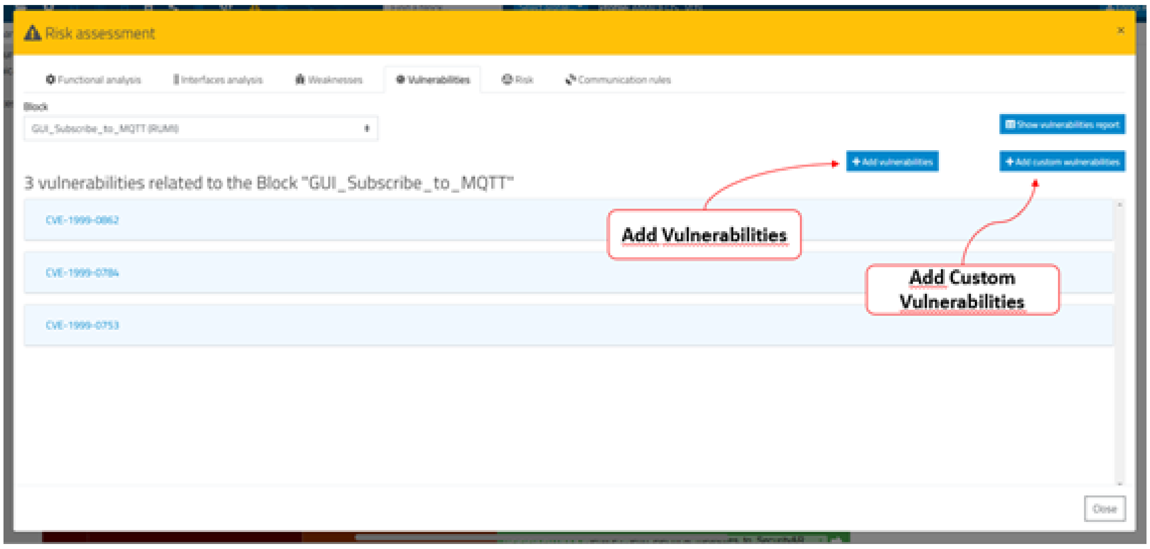
The process of association of weaknesses in the Risk Assessment allows the model designer user to search for and select a CWE (either directly or by performing the research of attack patterns in the CAPEC and retrieving the related weaknesses). Furthermore, the tool allows, by clicking on the *Add custom weaknesses* button the specification of custom weakness and their association to the asset. Each weakness is created entering the name, description, extended description, background details, likelihood Of Exploit (High, Mid, Low, None, Default, Unknown, Not Applicable, Quantified), and the status (Deprecated, Draft, Incomplete, Stable). This feature may be useful when the extensive search of weaknesses in the CWE catalogue does not allow to find the desired one.

After associating the weaknesses with the model's class blocks, the user can view a summary report, which can also be exported in CSV format. The fields available in the exported CSV report for the associated Weaknesses are:

- Exclude (with a yes or now depending on whether the weakness has been excluded or not respectively);
- Exclusion reason (the reason eventually provided by the user within the tool);
- Predefined (yes if the Weakness is inherited from the profile, no if it has been added in the Model);
- Component (the model element to which the weakness is associated) ;
- Weakness ID (the CWE-ID or custom id);
- Weakness Type (CWE or custom);
- Weakness title;
- Weakness description;
- Details (the link to CWE catalogue).

As for the weaknesses, the user can click on Risk Assessment and select the *Vulnerabilities* tab for performing the association of **vulnerabilities** from CVE catalogue.

Figure 10 shows the graphical interface of the *Vulnerabilities* tab, that allows the choice of the *Class Block* (as in the example the *GUI_Subscribe_to_MQTT (RUMI)*) that is implicitly considered as asset.



3 vulnerabilities related to the Block "GUI_Subscribe_to_MQTT"

CVE-1999-0862

CVE-1999-0784

CVE-1999-0753

Figure 10 Vulnerabilities tab in Risk Assessment

The association of the vulnerabilities is started by pressing the *Add Vulnerabilities* button as indicated in Figure 10. The interface that opens allows the search and retrieval of vulnerabilities from the CVE catalogue. The search can leverage keywords characterizing the title or the description of a CVE entry (e.g., the *sql* word). As seen for the weaknesses, the user can create also custom vulnerabilities by clicking on the button *Add custom vulnerabilities* shown in Figure 10, providing a name and description. As for the weaknesses, this feature may be useful when the extensive search of vulnerabilities in the CVE catalogue does not allow to find the desired one.

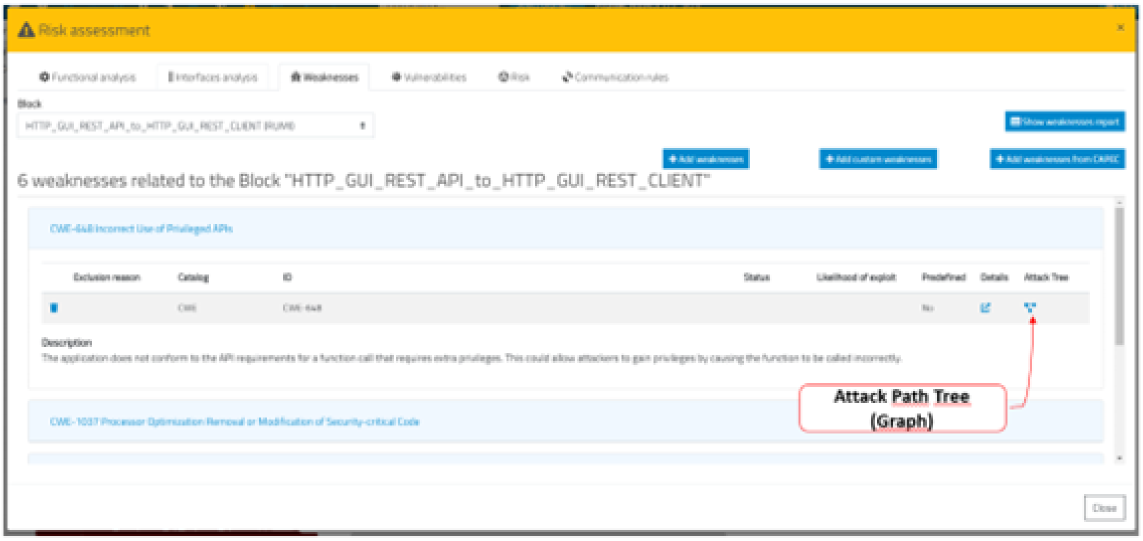
The “Export” button downloads the report in CSV format, where the fields available are:

- Exclude (with a yes or now depending on whether the vulnerability has been excluded or not respectively);
- Exclusion reason (the reason eventually provided by the user within the tool);
- Predefined (yes if the vulnerability is inherited from the profile, no if it has been added in the Model);
- Component (the model element to which the vulnerability is associated);
- Vulnerability ID (the CVE-ID or custom id);
- Vulnerability Type (CVE or custom);
- Vulnerability title;
- Vulnerability description;
- Details (the link to CVE catalogue).

3.2.5. Attack Paths Calculation

A more refined analysis of the impact associated to the identified vulnerabilities can be performed using also the Resilblockly tool. Based on CWE-CAPEC relationship, and in particular on the *related attack pattern* and *related weakness* fields existing in them, respectively, it is possible to build a useful graphical representation having as a root a weakness identified during the keyword-based search, and associated to a system component, and having as its children attack patterns that are *related to* it, and potentially have been as well identified during the keyword-based identification. Then, connecting these attack patterns with additional patterns that *canPrecede* them, we obtain a structure that we call Attack Path Tree (APT).

The generation of an Attack Path Tree (Graph) in ResilBlockly is done by clicking on the icon present for each CWE associated with a Class Block within the Weakness Tab of the Risk Assessment, as shown in the Figure 11.



Block
HTTP_GUI_REST_API_to_HTTP_GUI_REST_CLIENT (RUMI)

6 weaknesses related to the Block "HTTP_GUI_REST_API_to_HTTP_GUI_REST_CLIENT"

CWE-648 Incorrect Use of Privileged APIs

Exclusion reason	Catalog	ID	Status	Likelihood of exploit	Predefined	Details	Attack Tree
	CWE	CWE-648			No		

Description
The application does not conform to the API requirements for a function call that requires extra privileges. This could allow attackers to gain privileges by causing the function to be called incorrectly.

CWE-1037 Processor Optimization Removal or Modification of Security-critical Code

Figure 11 Generation of an Attack Path Tree (Graph)

Figure 12 shows one APT example for the weakness CWE-648 associated to the *HTTP_GUI_REST_CLIENT_to_HTTP_GUI_REST_API (RUMI)* Class Block: the weakness is represented on top of the APG (Attack Path Graph) and highlighted in yellow.

The tool automatically retrieves, where available:

1. Related attack patterns, tree representing them as red rectangles (e.g., CAPEC 107 and 234), and places them on the Level 1 of the APT;
2. Preceding attack patterns, still represented by red rectangles but and placed on the below levels of the APT (e.g., CAPEC 63);
3. Related weaknesses, represented with blue circles, and connected to all their related attack patterns. In this step the APT becomes an APG.

The displayed Attack Path Graph shows the name of weaknesses and attack patterns on mouseover (as depicted in Figure 13) and includes an URL to the dedicated page in the corresponding CWE or CAPEC catalogue (Figure 14).

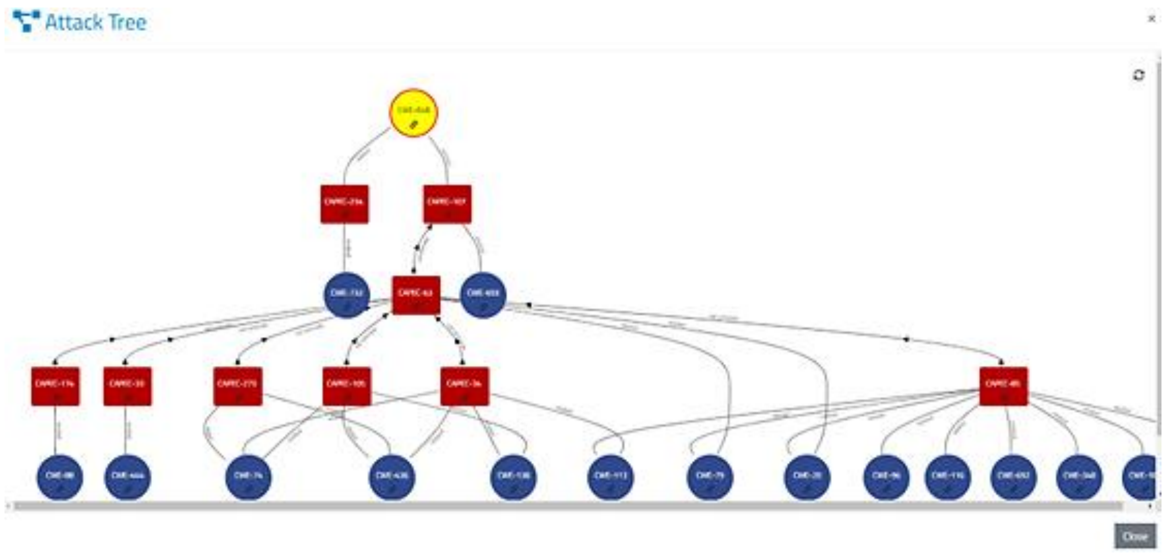


Figure 12 Attack Path Tree (Graph) example related to CWE-648 in ResilBlockly

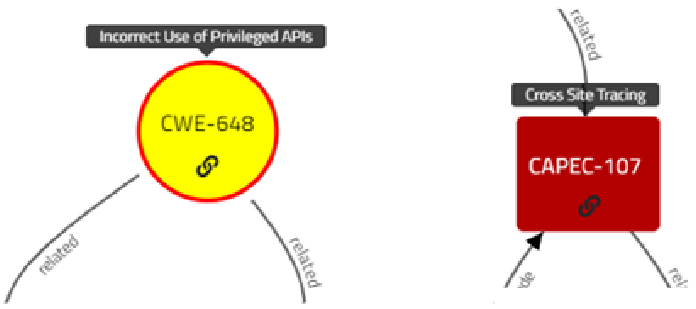


Figure 13 The name of a weakness (on the left) and of an attack pattern (on the right) in an APG example related to CWE-648 shown at mouseover

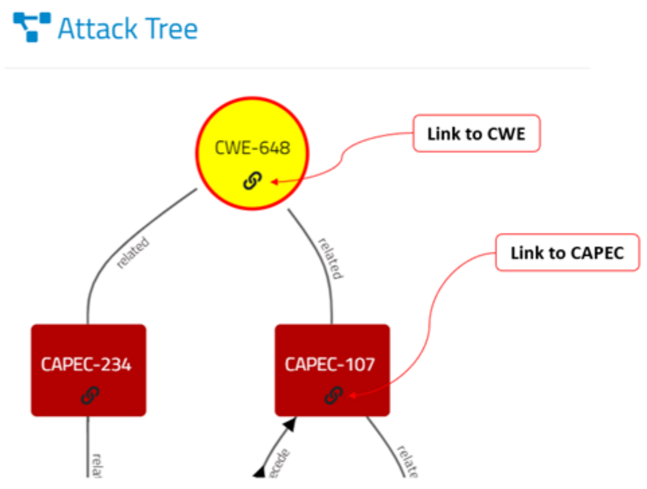


Figure 14 Link to CWE and CAPEC pages

3.2.6. Safety Impact Calculation

As described in D7.2, the methodology considers 4 different dimensions to calculate the impact value: Safety, Operational, Financial and Privacy & Legislation. In domains where application safety is relevant, a safety engineering lifecycle workflow is expected to be applied. This should include Hazard Analysis and Risk Assessment (HARA), which can identify specific Hazards (system-related conditions that could lead to accidents in worst-case operational situations), assess the risk of said Hazards with respect to safety, and identify appropriate objectives for controlling the risk to an acceptable degree. More detail on how such a process can be considered is described in D6.4[5], and bases this view on the ISO 26262 automotive safety standard, among others.

From the set of the Hazards identified via HARA above a relationship to the security claims that could contribute to their risk can be established. In automotive security standard ISO 21434, the process steps of damage scenario specification and impact rating are relevant.

Per RQ-15-01 Note 1 of the standard (pg.45 of the 2021 edition), a damage scenario can include the relation between functionality and adverse consequence, description of harm to the user, and/or relevant assets (system elements e.g., data/function/physical element, whose compromised security properties can lead to damage scenarios). The impact rating of damage scenarios derived above shall follow RQ-15-04, which prescribes assessment in terms of safety, financial, operational, and privacy impact (and additional categories if needed). Each category shall be rated using a scale from 'negligible', 'moderate', 'major' to 'severe'. For safety-related ratings specifically, the security-related damage scenario follows the severity impact rating of ISO 26262-3:2018, summarized in Table 1. This rating is considered also within the security evaluation methodology to measure the safety impact dimension.

Table 1 - ISO 21434 Safety Impact Rating (Table F.1 ISO 21434:2021)

Impact Rating	Safety Impact Criteria
Severe	Life-threatening injuries (survival uncertain), fatal injuries
Major	Severe and life-threatening injuries (survival probable)
Moderate	Light and moderate injuries
Negligible	No injuries

While the above concepts are all based on standards from the automotive domain, we believe the overall approach to be transferrable to other domains. Indeed, the above standards are themselves based on more general, cross-domain standards, such as IEC 61508. However, one important concern for adapting the rating process to other domains is the careful translation of the safety impact rating criteria to the new domain,

[5] Deliverable 6.4: "Mitigation Identification and Design"

accounting for the difference in mode and scale of interaction of the end-user(s) and the overall application.

As a contrasting example, the related aerospace safety standard ARP 4754-A (for commercial passenger aircraft) considers 'catastrophic' impact to be the highest, and includes ground/other aircraft collisions, meaning potentially hundreds of casualties (actual definition specified in AC 25.1329-1B, page 1-3).

Procedurally, to arrive at a safety impact along the above lines, BIECO specifies a security risk assessment method in deliverables D6.2[6] and D6.4. The process begins by performing system modeling and security risk assessment in the ResilBlockly tool, as indicated earlier in section **Error! Reference source not found.** and onwards. The resulting models of ResilBlockly can be exported into an exchangeable file format (based on the Eclipse Modeling Framework's. ecore format [29]).

Using this format, the safeTbox tool can import and transform the ResilBlockly models, such that identified vulnerabilities associated with system elements can be incorporated into Component Fault Trees (CFTs). An abstract example of how this appears graphically in the tool can be seen in Figure 15. As can be seen, the user has modeled a vulnerability with CVE-2004-0625 (as assigned and exported from ResilBlockly) to be linked with a logical gate (OR gate in this case) with an 'Out FM_445', referring to an output (port) failure mode. This represents how the corresponding CVE could trigger undesireable behavior in one of the subject component's output ports, which could then lead to failure and have corresponding safety related impact.

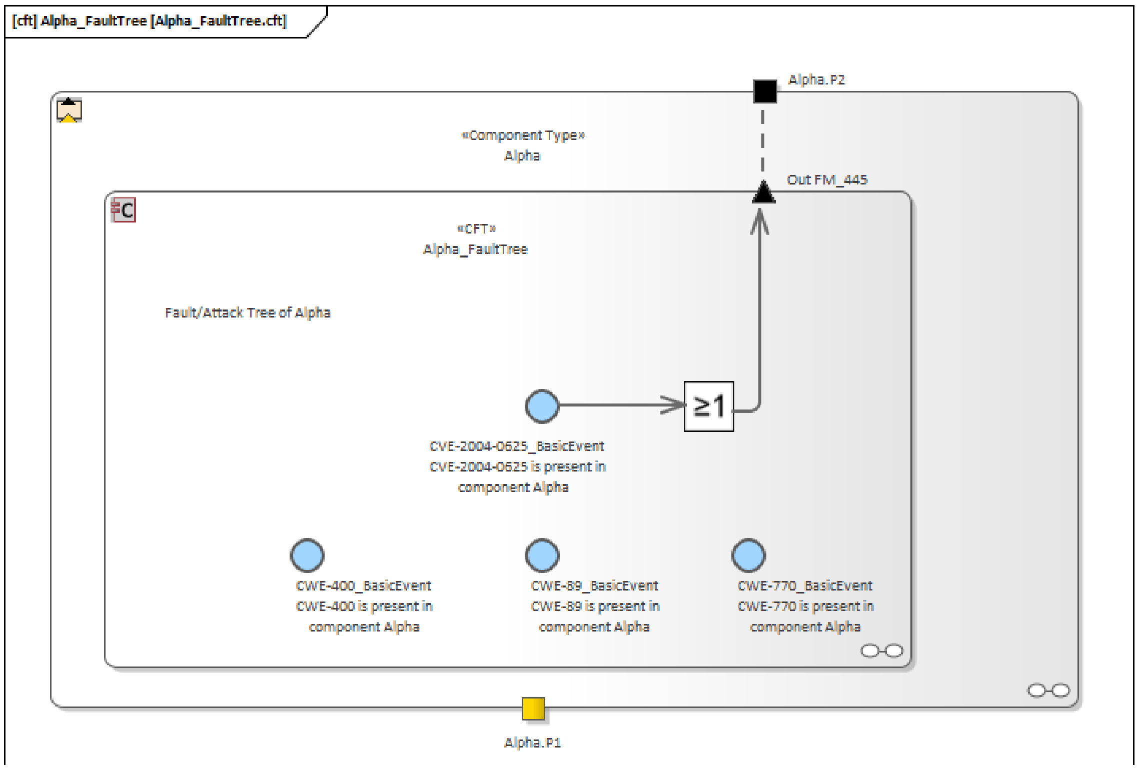
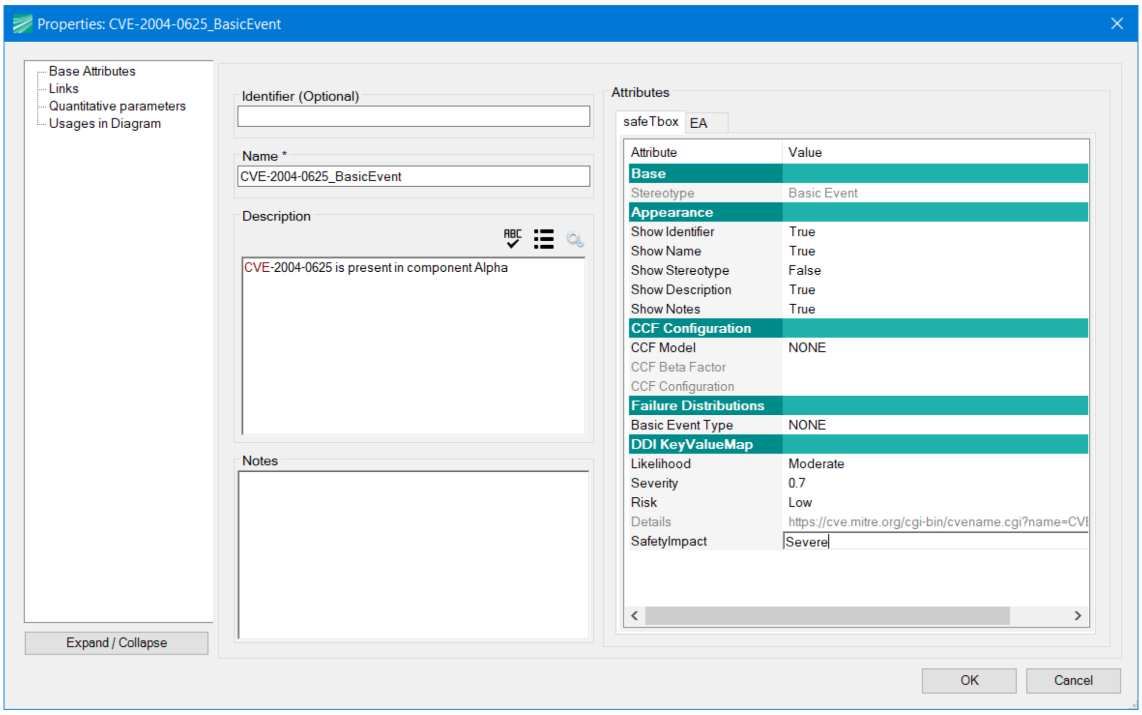

[cft] Alpha_FaultTree [Alpha_FaultTree.cft]
Alpha.P2
«Component Type»
Alpha
Out FM_445
«CFT»
Alpha_FaultTree
Fault/Attack Tree of Alpha
≥1
CVE-2004-0625_BasicEvent
CVE-2004-0625 is present in component Alpha
CWE-400_BasicEvent
CWE-400 is present in component Alpha
CWE-89_BasicEvent
CWE-89 is present in component Alpha
CWE-770_BasicEvent
CWE-770 is present in component Alpha
Alpha.P1

Figure 15 - Example of imported CVE in safeTbox

[6] Deliverable 6.2: "Blockly4SoS User Guide"

Once an analysis has been completed in safeTbox and the impact on safety due to the corresponding vulnerability has been identified, a properties editor can be used to assign a corresponding safety impact rating to the vulnerability in question, as seen in 16, in the bottom-right field listed as ‘SafetyImpact’.


Properties: CVE-2004-0625_BasicEvent

Identifier (Optional):

Name *: CVE-2004-0625_BasicEvent

Description: CVE-2004-0625 is present in component Alpha

Notes:

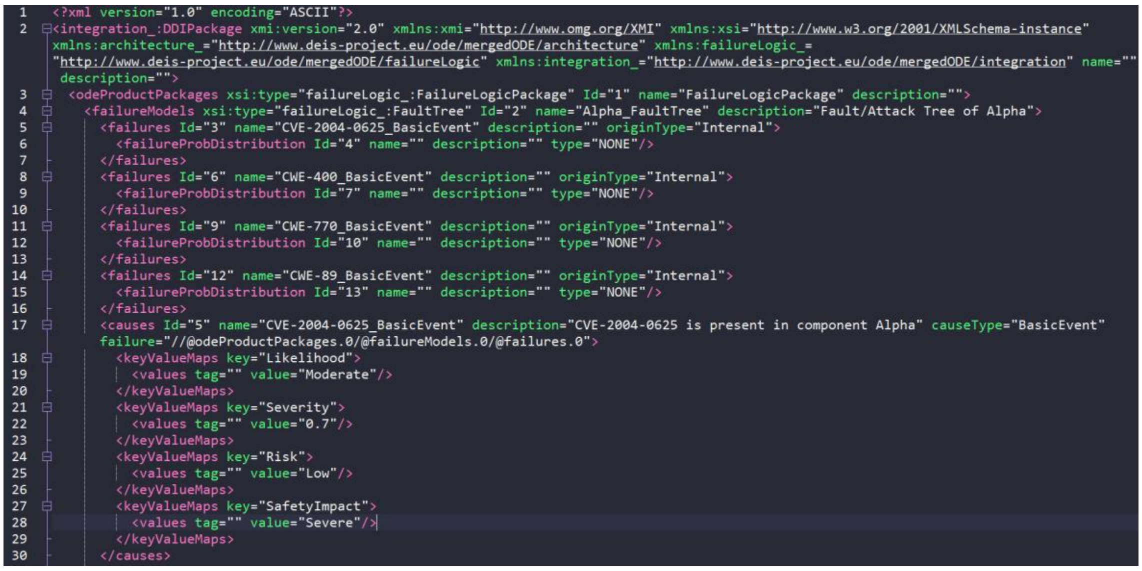
Attributes (safeTbox | EA)

Attribute	Value
Base	
Stereotype	Basic Event
Appearance	
Show Identifier	True
Show Name	True
Show Stereotype	False
Show Description	True
Show Notes	True
CCF Configuration	
CCF Model	NONE
CCF Beta Factor	
CCF Configuration	
Failure Distributions	
Basic Event Type	NONE
DDI KeyValueMap	
Likelihood	Moderate
Severity	0.7
Risk	Low
Details	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVI
SafetyImpact	Severe

Figure 16 - Example of editing CVE properties in safeTbox

The corresponding rating is then stored as a property of the vulnerability, and can be re-exported into other tools (e.g., in ResilBlockly or other tools), using the above-mentioned file format. An example of how this format looks like, for the example model shown previously, can be seen in 17, lines 27 and 28, where the corresponding CVE’s safety impact rating has been recorded. More details on how the above tools interoperate technically are included in BIECO deliverable D6.3[7], section 4.1.

[7] Deliverable 6.3: “Risk Assessment and additional requirements”



```xml
<?xml version="1.0" encoding="ASCII"?>
<integration_:DDIPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:architecture_="http://www.deis-project.eu/ode/mergedODE/architecture" xmlns:failureLogic_=
"http://www.deis-project.eu/ode/mergedODE/failureLogic" xmlns:integration_="http://www.deis-project.eu/ode/mergedODE/integration" name=""
 description="">
 <odeProductPackages xsi:type="failureLogic_:FailureLogicPackage" Id="1" name="FailureLogicPackage" description="">
   <failureModels xsi:type="failureLogic_:FaultTree" Id="2" name="Alpha_FaultTree" description="Fault/Attack Tree of Alpha">
     <failures Id="3" name="CVE-2004-0625_BasicEvent" description="" originType="Internal">
       <failureProbDistribution Id="4" name="" description="" type="NONE"/>
     </failures>
     <failures Id="6" name="CWE-400_BasicEvent" description="" originType="Internal">
       <failureProbDistribution Id="7" name="" description="" type="NONE"/>
     </failures>
     <failures Id="9" name="CWE-770_BasicEvent" description="" originType="Internal">
       <failureProbDistribution Id="10" name="" description="" type="NONE"/>
     </failures>
     <failures Id="12" name="CWE-89_BasicEvent" description="" originType="Internal">
       <failureProbDistribution Id="13" name="" description="" type="NONE"/>
     </failures>
     <causes Id="5" name="CVE-2004-0625_BasicEvent" description="CVE-2004-0625 is present in component Alpha" causeType="BasicEvent"
     failure="//@odeProductPackages.0/@failureModels.0/@failures.0">
       <keyValueMaps key="Likelihood">
         <values tag="" value="Moderate"/>
       </keyValueMaps>
       <keyValueMaps key="Severity">
         <values tag="" value="0.7"/>
       </keyValueMaps>
       <keyValueMaps key="Risk">
         <values tag="" value="Low"/>
       </keyValueMaps>
       <keyValueMaps key="SafetyImpact">
         <values tag="" value="Severe"/>
       </keyValueMaps>
     </causes>
```

Figure 17 - Example of exported file with safety impact rating

3.3. Security Testing

In this Section, testing techniques and tools used in the BIECO project are presented. We define the techniques of testing and how they are employed in the BIECO. Then, we present specific open-source tools that were used to achieve effective automated testing methodologies.

In particular, we integrate 3 different testing techniques (model-based testing, fuzzing testing and combinatorial testing) using three different tools that have been developed or improved within BIECO: Graphwalker, fuzzing tool and GROOT. Even if Resilblockly was considered at the beginning of the W7P work as a candidate for MBT (Model-Based Testing), it was replaced by the open-source tool Graphwalker. The main reason was the automation of the test generation process, which we consider highly important in a MBT tool, and the link with the real system, as Resilblockly can execute the tests only over a simulated system and does not export a test suite nor adapter to link the tests with the real system.

3.3.1. GraphWalker - Model Based Testing

MBT is a software testing technique where the generation of test cases is based on models that describe the behavior of the SUT. Therefore, the activity of designing models consists in represent the target part of the system, being able to use different types of models depending on the tool used. Applying the graph theory to these models, we can automate the generation of multiple test scripts (test suite). The test suite is a path made up of steps through the model until a goal, condition or requirement is met.

Following this technique we give to our tests a better structure, updates on the model to reflect new conditions or requirements make the test suite easy to maintain, improving test coverage and saving time and costs.

GraphWalker[39] is an open-source solution for MBT that reads models in the shape of directed graphs and generates tests from these graphs. The general idea is to model an application as a graph of calls and verifications which in turn can be employed for extensive and automated testing.

The tool provides a GraphWalker Studio, an editor in which models can be created and edited. Models can also be created *by-hand,* but this guide describes GraphWalker Studio for model creation as an intuitive tool requiring little coding knowledge. Studio also has a feature to run test path generation to verify if the models are correct. The file format of the generated models is JSON.

Models are composed of two main elements: vertex and edges. On the one hand, a vertex represents the state of the SUT, and it is the place where the *asserts* take place in the code. On the other hand, an edge is an action that changes the state of the system. For our methodology, requirements are another important actor in this tool when designing models, since they are in charge of defining where the condition or goal vertex that indicates the end of a test is located. Also exists other two remarkable elements: actions, that modifies data from the model context, and guards, that do not allow to walk through an edge until a certain condition is met based on the model context.

Moreover, GraphWalker provides command line tools for generating paths, which can be integrated as a Maven project. It requires only an implementation of vertices and edges, and the tests are run automatically.

3.3.1.1. Using Graphwalker Tool

The first step to use Graphwalker is to start the GraphWalker Studio. It can be executed:

a) Locally:
   i) Prerequisite: java is installed (GraphWalker studio runs on the majority of JRE versions).
   ii) Download the latest GraphWalker Studio from GraphWalker download page[8].
   iii) Launch GraphWalker Studio: *java -jar graphwalker-studio-<LATEST VERSION>.jar*
   iv) GraphWalker studio can be opened in a browser: *localhost:9090/studio.html*

b) Within the BIECO platform: Studio tool will be integrated into the BIECO platform in order to avoid manual installation.

Once started, a model can be created by clicking the '+' button (Figure 18, left). On the new model tab, the user can press the 'v' key and press a left mouse button (LMB) to create a vertex. To create an edge, the user can click on a vertex, press the 'e' key, then press and hold LMB, dragging it to another vertex. This will create a directed edge between two vertices.

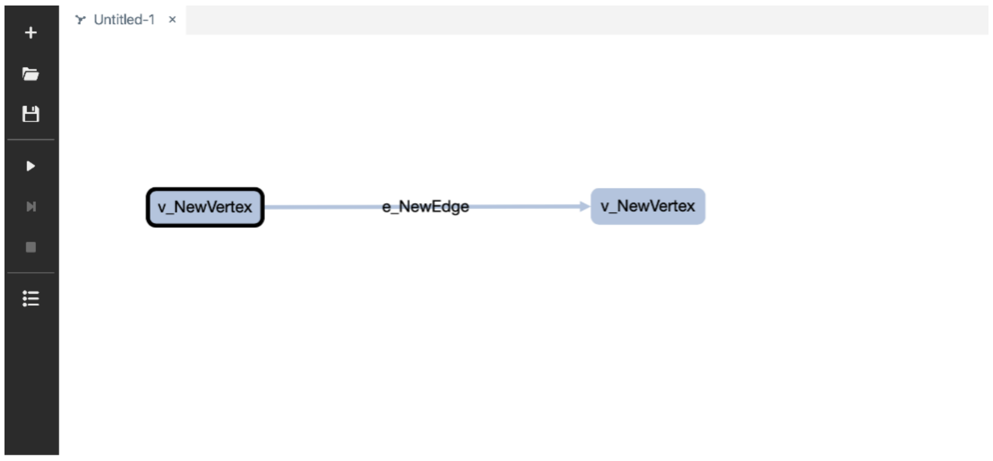


Figure 18 A simple graph in GraphWalker Studio

[8] http://graphwalker.github.io/#download

The name and other properties of each vertex and edge can be modified by using modification button (See Figure 19).

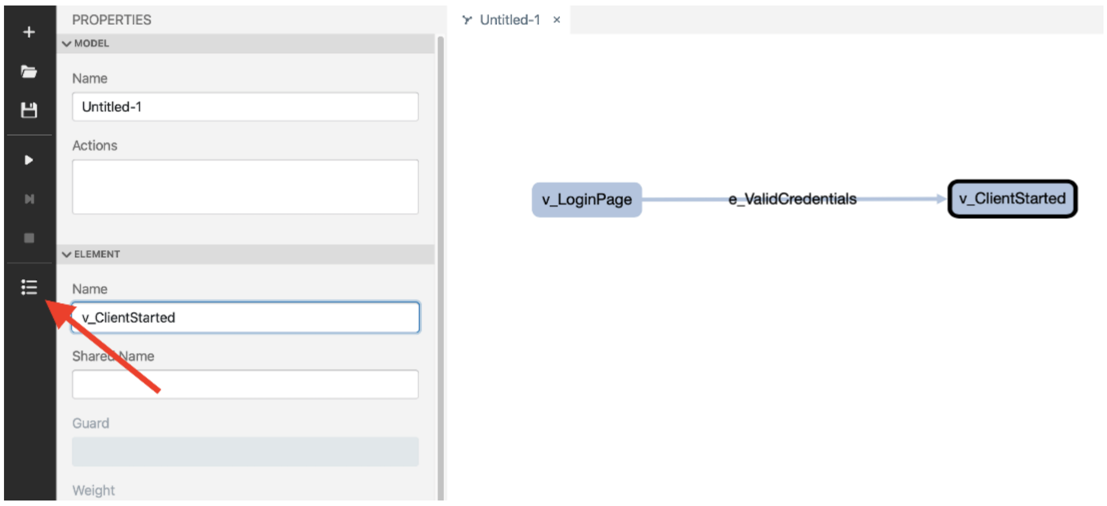


Figure 19 Element names modification in GraphWalker Studio

Finally, the user can set actions and guards to control a state (for example, introduce a variable responsible for click count), as well as add information about requirements related to given verifications.

Graphwalker also allows to import and existing model **by clicking the** 'Open' button or save a created model as a JSON file just click the *Save* button in the GraphWalker Studio and choose the name and directory in the browser pop-up window.

To test the model based on the graph design, guards, conditions, start element, generator and end stop conditions defined, the user needs to click the *Play* button in GraphWalker studio (see Figure 20).

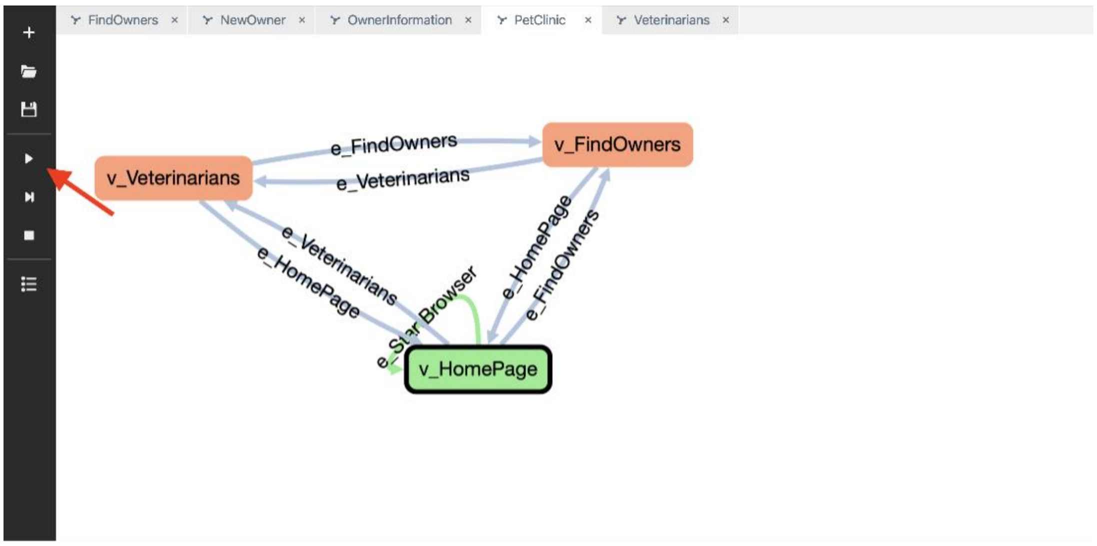


Figure 20 Initiating a graph test in GraphWalker Studio

The test run can be controlled by the three buttons:

i) *Play/Pause* - starts or pauses the run.
ii) *Next* – executes only one step (one transition).
iii) *Stop* – terminates the test run.

After fulfilling the stop condition, the test is finished. The vertices visited at least once are coloured green.

3.3.1.2. Test Generation: Adapter and Test Suite

Once generated the model for our SUT, the user needs to create the adapter that contains the connection between the abstract model and the real system, as well as the test suite that contains the different test cases that will be executed over the system.

To achieve this, BIECO has created on its platform the tool Test Adapter and Suite Generator, which is in charge of automating the creation of both needed files (see Figure 21).

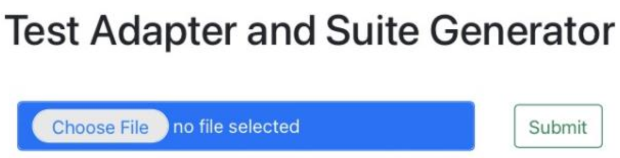
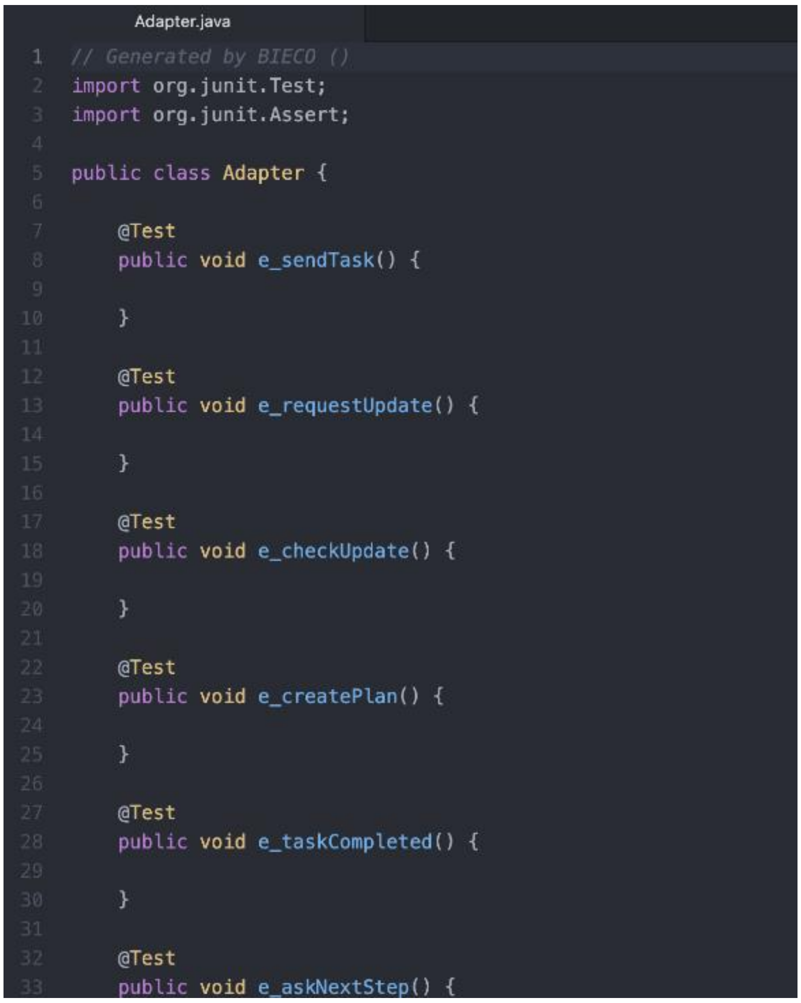


Figure 21 Test Adapter and Suite Generator tool from BIECO platform

As entry, the user should provide the resulting JSON file from GraphWalker Studio by pressing the "Choose File" button. Once submitted, the tool will redirect to the page where the user can download two resulting Java files: 'Adapter.java' and 'TestSuite.java'.

The first one, 'Adapter.java' contains the methods that the user should implement in order to provide the connection of the tests with the real system. In the case of vertex methods, the asserts should be also implemented. An example of adapter file is shown in Figure 22.

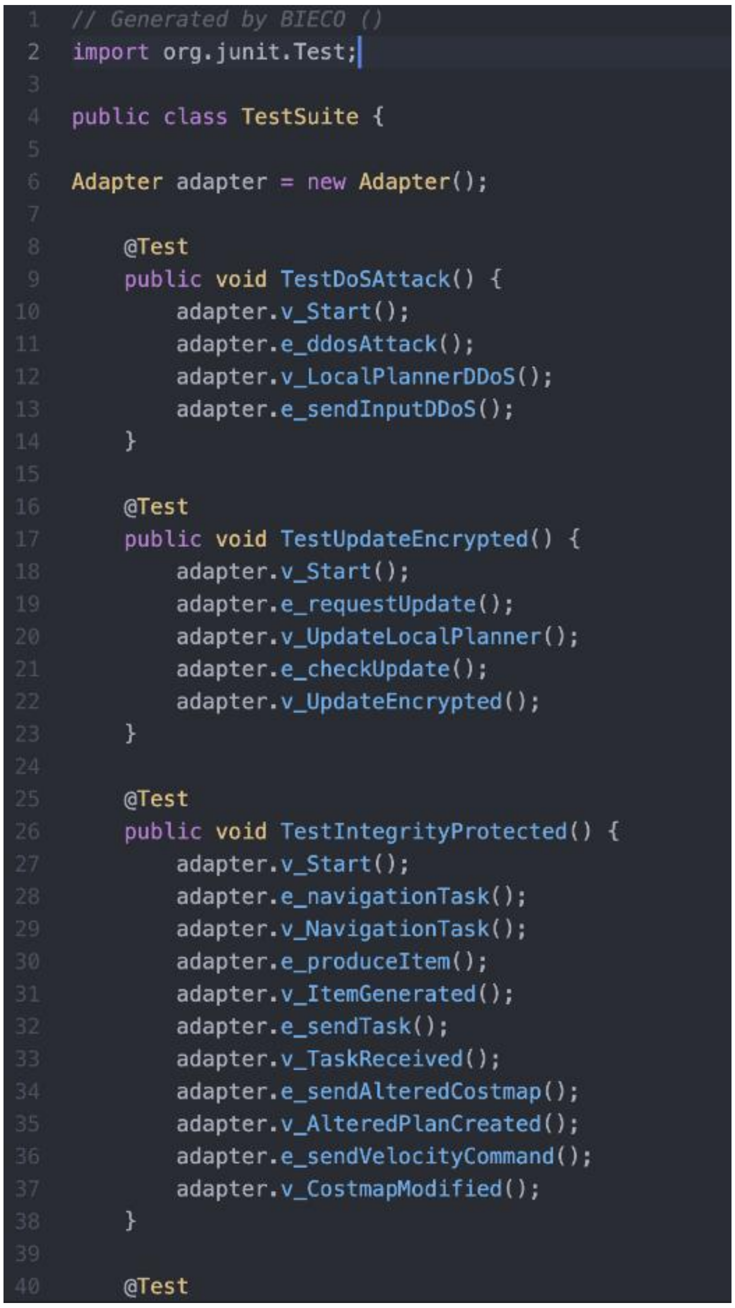


Adapter.java

```java
// Generated by BIECO ()
import org.junit.Test;
import org.junit.Assert;

public class Adapter {

    @Test
    public void e_sendTask() {

    }

    @Test
    public void e_requestUpdate() {

    }

    @Test
    public void e_checkUpdate() {

    }

    @Test
    public void e_createPlan() {

    }

    @Test
    public void e_taskCompleted() {

    }

    @Test
    public void e_askNextStep() {
```

Figure 22 'Adapter.java' example

The second one, 'TestSuite.java' contains the different test cases that will be executed over our system. Each of these tests is a sequence of calls to the Adapter methods. An example of test suite file is shown in Figure 23.



```java
// Generated by BIECO ()
import org.junit.Test;

public class TestSuite {

Adapter adapter = new Adapter();

    @Test
    public void TestDoSAttack() {
        adapter.v_Start();
        adapter.e_ddosAttack();
        adapter.v_LocalPlannerDDoS();
        adapter.e_sendInputDDoS();
    }

    @Test
    public void TestUpdateEncrypted() {
        adapter.v_Start();
        adapter.e_requestUpdate();
        adapter.v_UpdateLocalPlanner();
        adapter.e_checkUpdate();
        adapter.v_UpdateEncrypted();
    }

    @Test
    public void TestIntegrityProtected() {
        adapter.v_Start();
        adapter.e_navigationTask();
        adapter.v_NavigationTask();
        adapter.e_produceItem();
        adapter.v_ItemGenerated();
        adapter.e_sendTask();
        adapter.v_TaskReceived();
        adapter.e_sendAlteredCostmap();
        adapter.v_AlteredPlanCreated();
        adapter.e_sendVelocityCommand();
        adapter.v_CostmapModified();
    }

    @Test
```

Figure 23 'TestSuite.java' example

Having both files and having implemented the Adapter with the required functionality the user is able to execute all the tests by using Maven.

3.3.1.3. Test Implementation and Execution by Using Maven

The software requirement to perform the execution of the tests is to have a Java environment that allows the creation of projects with Maven.

Once created the Maven project, we must add the 'Adapter.java' and 'TestSuite.java' files to the folder *test* of our project structure. The project should be similar to the one shown in Figure 24.

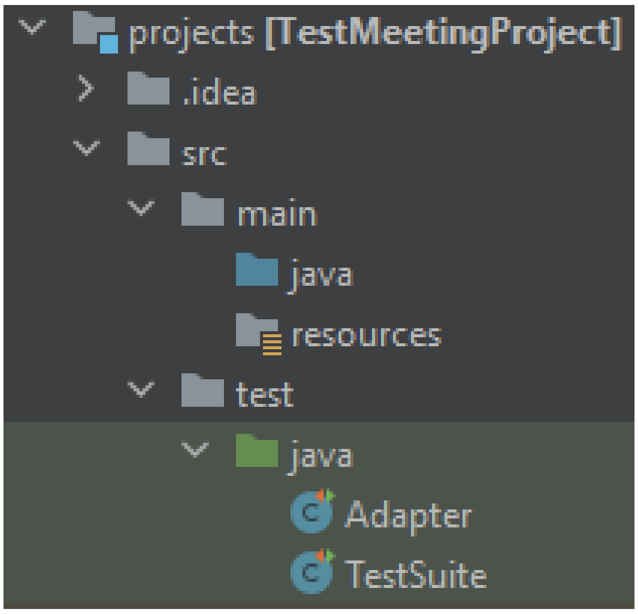

```
projects [TestMeetingProject]
  .idea
  src
    main
      java
      resources
    test
      java
        Adapter
        TestSuite
```

Figure 24 Add both Adapter and TestSuite files into the Maven project

Next step is to implement the Adapter methods. The task of the user is to implement these methods with the part of the SUT functionality to be tested. After implementation is complete, to perform the execution it is enough to use the maven command:

mvn test

As result, it generates a new folder in our project structure containing the test report (see Figure 25):

target/surefire-reports/

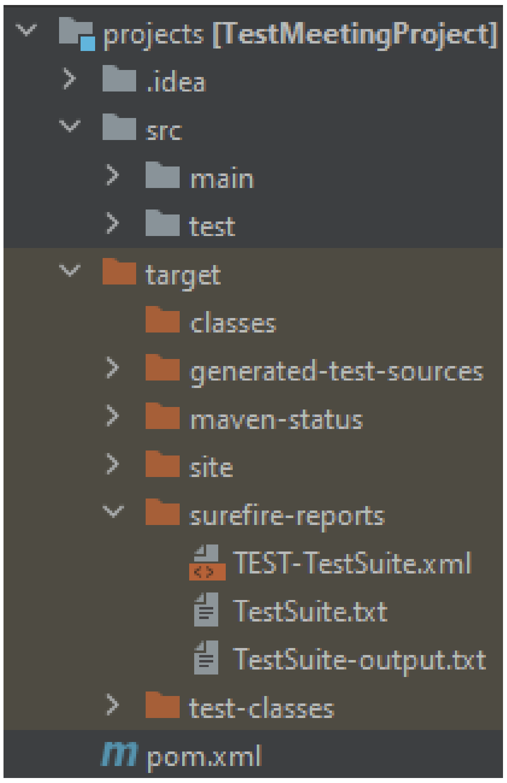

```
projects [TestMeetingProject]
  .idea
  src
    main
    test
  target
    classes
    generated-test-sources
    maven-status
    site
    surefire-reports
      TEST-TestSuite.xml
      TestSuite.txt
      TestSuite-output.txt
    test-classes
  pom.xml
```

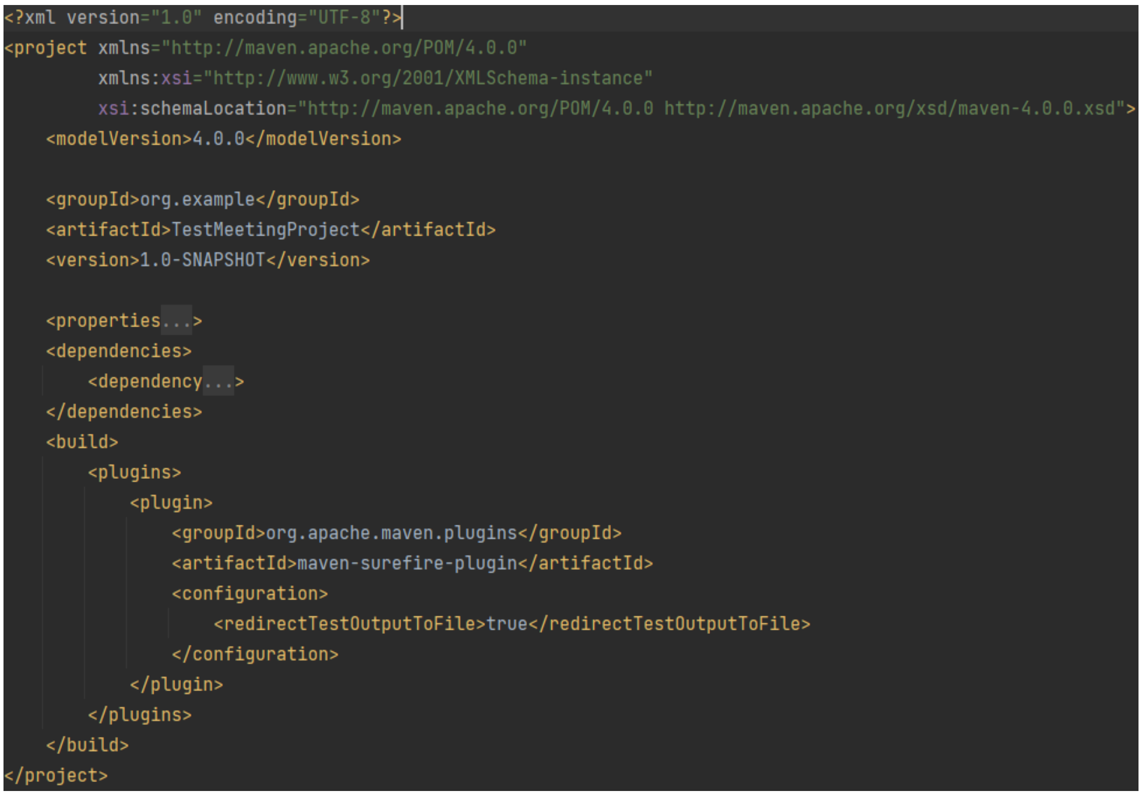
Figure 25 Test report folder

Finally, two results files are obtained:

1. 'TEST-TestSuite.xml': Collects the results of each test (pass/fail), describing the failing reasons and the assertions messages with the relevant information. This is the main output of the tests after executing with Maven.

2. TestSuite-output.json: Collects the set of metrics derived of each test, described in a JSON format.

To generate this second output, it is necessary to modify the 'pom.xml' file from our Maven project adding the plugin which allows you to redirect the output to a new file created during test execution, as shown in Figure 26.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>TestMeetingProject</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties...>
    <dependencies>
        <dependency...>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <configuration>
                    <redirectTestOutputToFile>true</redirectTestOutputToFile>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Figure 26 Plugin added to 'pom.xml' in our Maven project

Besides to complete the information included in this output file, the tool *Test Adapter and Suite Generator* provides the user with a method within the 'Adapter.java' class that allows to indicate the metrics and values information that are wanted to be included as extra information to the test execution. The reason is that some of the Graphwalker tests could be non-binary (not only pass/fail), returning specific values or metrics. The outputs given are later described with details in section 3.4.1.

Both are later analyzed or used by other BIECO tools (SecurityScorer, MUD Updater) to obtain conclusions from the generated results.

3.3.2. Fuzzing Testing

The purpose of the Fuzzing tool is to evaluate the possible requests that a platform may receive, preventing possible errors or vulnerabilities that have not been previously considered. In this way, risks such as vulnerability exploitations or information leaks, among others, are detected and controlled. To perform this assessment, the user runs the Fuzzing tool on a developed platform or endpoint. The tool sends multiple HTTP requests by combining a number of parameters that may lead to errors. The responses of these requests are analysed by identifying the parameters that are not contemplated in the specification file, which may cause some risk to the platform.

The execution process followed by the Fuzzing tool is described in Figure 27.

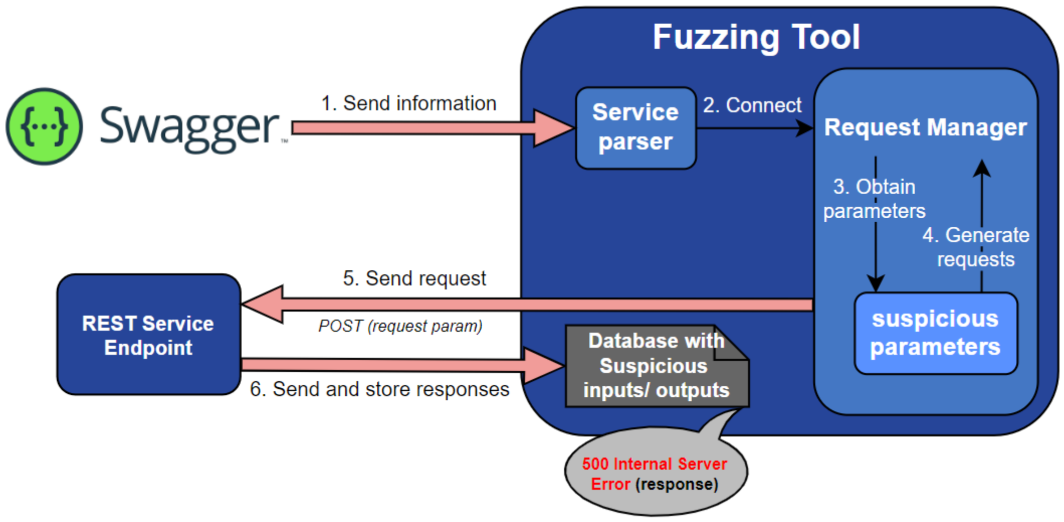


Figure 27 Conceptual design of the fuzzing tool

Before the execution of the tool and as requirements, it is necessary to provide an implemented endpoint and its corresponding swagger file. This swagger file documents the endpoint of the platform where the fuzzing is performed.

Once the necessary information is obtained from the swagger file, a database is created, which will be internal to the tool. This database is overwritten at each execution. Inside this database, 4 tables are created where the following will be stored:

1. **Information obtained from the swagger**, such as URLs, operation, path and responses that each operation may have.
2. **Requests made with the Fuzzing tool,** including its header, body, HTTP operation, parameters entered, and others.
3. **Correct requests** contemplated by the swagger file and that do not represent any risk. This table includes information such as URL, path, parameters entered, or response obtained.
4. **Suspicious requests** not included in the swagger file. The information stored in this table is the URL, path, type of vulnerability it may contain or response length, and others.

The information provided by the swagger file is parsed by the tool to obtain the necessary information for its execution, including the following:

- The **URLs** allowed by the endpoint, such as HTTP and/or HTTPS;
- The **paths**, which are the different pages that each URL can have;

- **HTTP operations**, such as POST, GET, PUT;
- The **parameters** that can contain the requests of each operation;
- The **contemplated responses** of each HTTP operation.

After having the necessary information from the swagger file, the parameters to be introduced in the requests are obtained. These data are internal of the tool and are suspicious in order to check if the different requests give errors, or if exists any vulnerability (e.g., SQL parameter to perform SQL injection). Once the database is created and the necessary information is obtained, the tool proceeds to do the petitions. First, default request is created in order to have a correct response which is used as a reference for the rest of the responses. Then, the corresponding requests are elaborated for each URL, path and specific operation. For each request, the default request is taken and each parameter is changed by the suspicious parameter and sent to the endpoint. The response is stored in the table that store all responses. The response is checked to see if it is suspicious of vulnerability or has not been contemplated in the swagger file. To do this, several analyses are performed, such as the code of the request made is within the responses contemplated by the swagger file, or the comparison between the message length of the request response and the reference response, so that, if there is much difference, the request can be detected as suspicious. The responses that are not suspicious are stored in the table of the correct responses. The suspicious requests are stored in the suspicious requests table of the internal database, notified to the user and sent to the security scorer.

3.3.3. GROOT

GROOT (GdpR-based cOmbinatOrial Testing) is a general combinatorial testing approach, for validating systems managing GDPR's concepts (e.g., Data Subject, Personal Data or Controller)[47]. In the following, we illustrate the GROOT methodology by using the following definitions:

Definition 1 (GDPR-based SUT Model) A GDPR-based SUT Model is a tuple $\mathrm{Model}_{\mathrm{GDPR}}\,(\mathrm{PAR}, \mathrm{V})$, where:

- $PAR \subset \{\mathrm{DS}, \mathrm{PD}, \mathrm{DC}, \mathrm{DP}, \mathrm{C}, \mathrm{P}, \mathrm{PA}, \mathrm{TP}\}$ is the set of parameters that affect the GDPR-based SUT, where DS = Data Subject, PD = Personal Data, DC = Controller, DP = Processor, C = Consent, P = Purpose, PA = Processing Activity, TP = Third Party, and
- $\mathrm{V} = \{\mathrm{V_i} \mid \mathrm{i} \in \mathrm{PAR}$ and $\mathrm{V_i}$ is set of values for the parameter i$\}$ is the set of the sets$\}$

Definition 2 (GDPR-based Test Case) Given a GDPR-based SUT Model $\mathrm{Model}_{\mathrm{GDPR}}\,(\mathrm{PAR}, \mathrm{V})$, a GDPR-based Test Case is a Tuple $\mathrm{TC}_{\mathrm{GDPR}}\,(\mathrm{ATT})$ where: $\mathrm{ATT} = \{\mathrm{ATT_i} \mid \mathrm{ATT_i} \subseteq \mathrm{V_i},\ \mathrm{i} \in \mathrm{PAR}$ and $\mathrm{V_i} \in \mathrm{V}\}$

The GROOT methodology takes as an input a GDPR-based implementation, representing the GDPR in terms of a specification language. GROOT is composed of three main steps (see Figure 28): GDPR-based Model Derivation; Test Cases Generation; and Test Cases Translation.

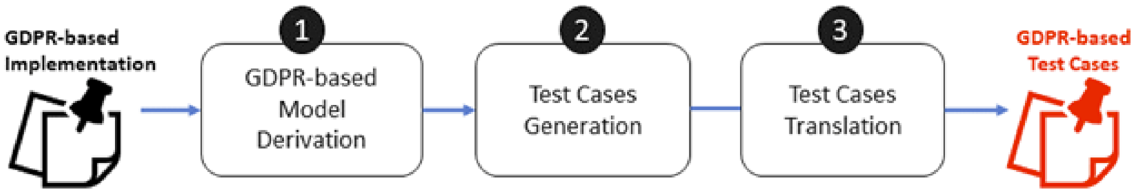


Figure 28 Contextualization of GROOT within BIECO

GDPR-based Model Derivation (Step1). In line with Definition 1, the GDPR-based SUT Model of the GDPR-based implementation is then derived. For this, the GDPR-based implementation is parsed in order to identify the set of parameters P, and the associated set of sets V. More precisely, for each parameter i, the subset V_i, containing the values used in the GDPR-based implementation, is derived.

Test Cases Generation (Step2). In this step, combinatorial testing is performed. Based on the derived parameters' values sets, different combinatorial strategies can be adopted such as all-combinations, pairwise combinations, or t-wise combinations. For instance, in the all-combinations test strategy according to Definition 2, for each parameter *i* and its set of value V_i, the power set of $V_i(P(V_i))$ is derived, i.e., all possible subsets of V_i. Then, the obtained powersets $P(V_i)$ are combined so as to derive the test cases, i.e., the $TC_{GDPR}(ATT)$ tuples. Because combinatorial testing is costly, selecting the best combinatorial strategy that could be adopted may depend on different testing objectives such as coverage, effectiveness, reduction, or prioritization.

Test Cases Translation (Step3). According to the domain-specific language, each of the obtained TCGDPR(ATT) tuples in Step 2 is translated into a specific executable test case. In the context of access control, a test case is represented through an AC request that the access control mechanism can evaluate.

3.3.3.1. Contextualization of GROOT in BIECO

The contextualization of GROOT methodology is depicted in Figure 29, which involves different components and artifacts developed within BIECO.

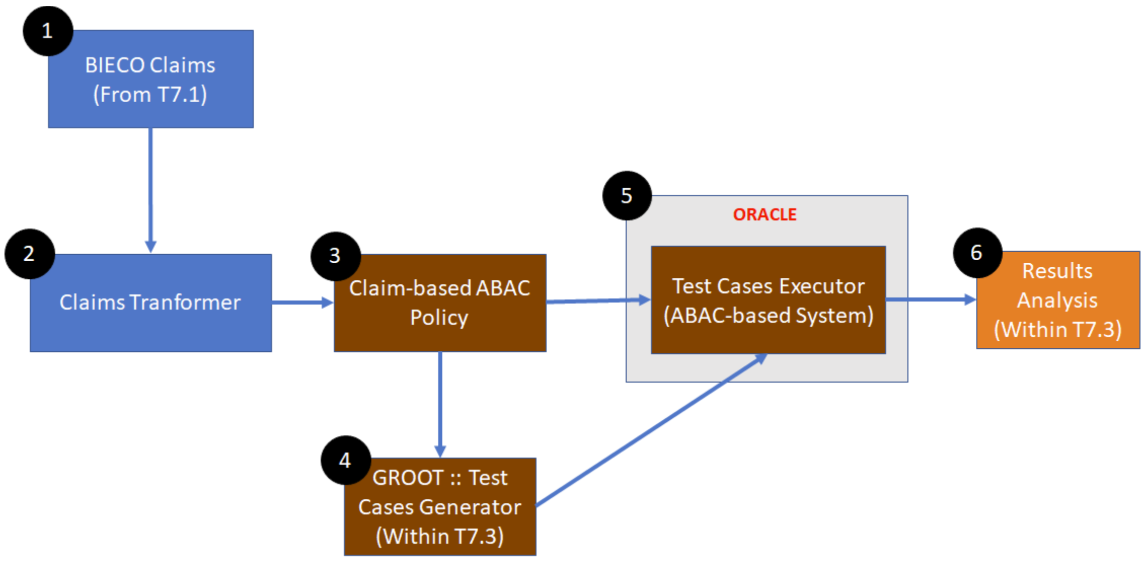

1
BIECO Claims (From T7.1)
2
Claims Tranformer
3
Claim-based ABAC Policy
4
GROOT :: Test Cases Generator (Within T7.3)
5
ORACLE
Test Cases Executor (ABAC-based System)
6
Results Analysis (Within T7.3)

Figure 29 Contextualization of GROOT within BIECO

The basic idea is to start from the BIECO Claims collected in task T7.1, i.e., the privacy claims (component 1 in the figure). These claims are then translated into authorization policies (through the Claim Transformer, component 2); in this context we are referring to access control policies expressed in ABAC (component 3, Claim-based ABAC Policy, Attribute-Based Access Control). Component 2 refers to the application of the GROOT methodology described above. Both the policy and the generated access control quests (in the form of <policy, {requests}>) are then used to by the Oracle component so as to associate to each request the expected result. To this end, component 5 integrate a specific Test Cases Executor able to evaluate each request over the policy by obtaining the expected authorization response. The result of Oracle represented in the form of **<policy, {(request, response)}>** is then used by Results Analysis (component 6) within task T7.3, and in particular by the SecurityScorer.

3.3.3.2. Supporting Framework

GROOT and its partial contextualization are being supported by a reference framework, called GROOT Testing Framework, depicted in Figure 30, and it composed of five components: GROOT Client; three Services, namely GROOT Proxy Service, GROOT Requests Generator, and GROOT Requests Evaluator, and GROOT Testing DB.

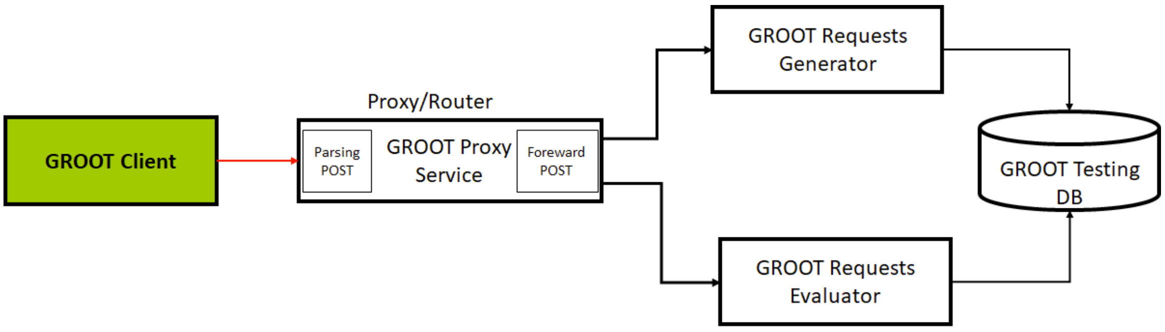


Figure 30 GROOT Reference Architecture

GROOT Client: It allows interacting with the overall framework through a specific GUI, that enables performing four distinct operations. As depicted in the implemented GUI reported in Figure 31, the available operations are:

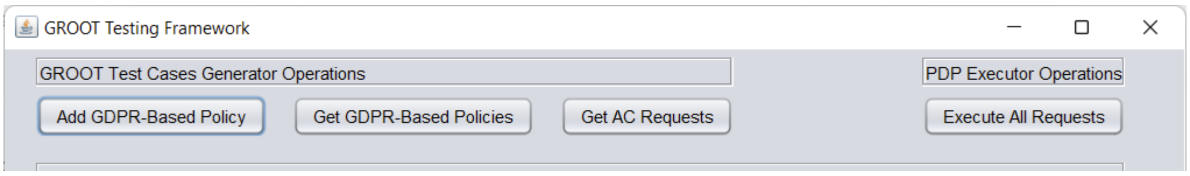


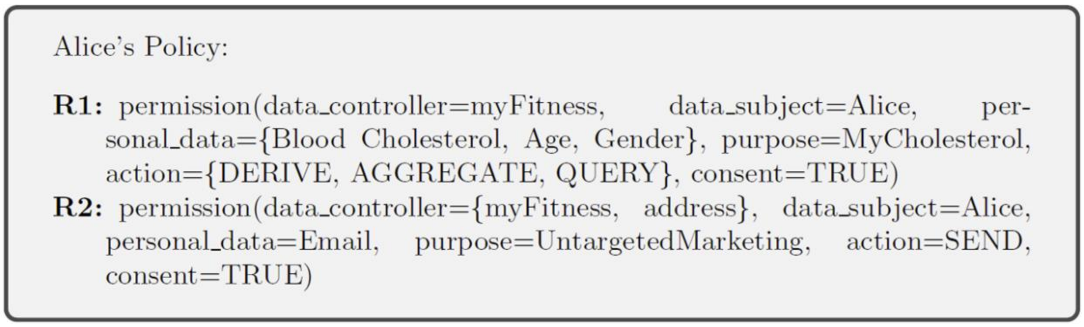
Figure 31 GROOT Client GUI

1. ***Add GDPR-Based Policy (Op1)*** aims at uploading the GDPR-based policy into the system and the contextual access control requests generation.
2. ***Get GDPR-Based Policies (Op2)*** allows to retrieve all the access control policies available into the system.

3. ***Get AC Requests (Op3)*** allows retrieving all the access control request (i.e., the test cases) associated with a given GDPR-based policy.
4. ***Execute All Requests (Op4)*** aims at executing and evaluating a set of access control requests and the visualization of the obtained results, i.e., the authorization decision associated to each request.

GROOT Requests Generator: It is a generator of access control requests, starting from the information contained into the access control policy. In the current implementation, we rely on the XACML standard for expressing both access control policies and access control requests. This component enables performing the first three operations described above (Op1, Op2 and Op3)

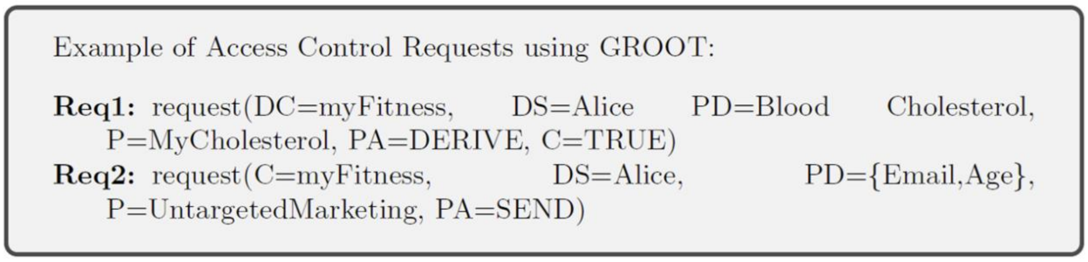
A possible GDPR-based access control policy (called Alice's Policy, see Figure 32) and associated requests (namely, Req1 and Req2, see Figure 33), adopted from [47],are reported below.



Alice's Policy:

R1: permission(data_controller=myFitness, data_subject=Alice, personal_data={Blood Cholesterol, Age, Gender}, purpose=MyCholesterol, action={DERIVE, AGGREGATE, QUERY}, consent=TRUE)
R2: permission(data_controller={myFitness, address}, data_subject=Alice, personal_data=Email, purpose=UntargetedMarketing, action=SEND, consent=TRUE)

Figure 32 GDPR-based Access Control Policy related to Lawful Processing



Example of Access Control Requests using GROOT:

Req1: request(DC=myFitness, DS=Alice PD=Blood Cholesterol, P=MyCholesterol, PA=DERIVE, C=TRUE)
Req2: request(C=myFitness, DS=Alice, PD={Email,Age}, P=UntargetedMarketing, PA=SEND)

Figure 33 Example of GDPR-based Access Control Requests

GROOT Requests Evaluator: This is the component that implement the fourth operation available in GROOT Testing Framework (Op4), and it allows associating an authorization response to each of the selected access control request. This component is used as an Oracle for testing access control systems managing privacy concerns such as the GDPR.

GROOT Proxy Service: This component interacts with the GROOT Client and based on the requested operation, forwards the request to the right service.

GROOT Testing DB: This component allows the persistency of the generated data, both access control policies and requests during the GROOT lifecycle.

3.4. Risk Estimation

The test results are analysed to get a numeric value of the risk, using the results of the tools described in Section 3.3 and an additional tool for risk evaluation. In order to gather, analyse and evaluate the results of the security testing tools used in BIECO, a new tool was created, *SecurityScorer*[9]. This tool parses the results of a given security testing tool, which can be in various formats, like .json, .xml, .csv, etc., and an input file created by the user, describing the threats and their impact to calculate a numeric value of a risk for a given software. The overview is presented in Figure 34.

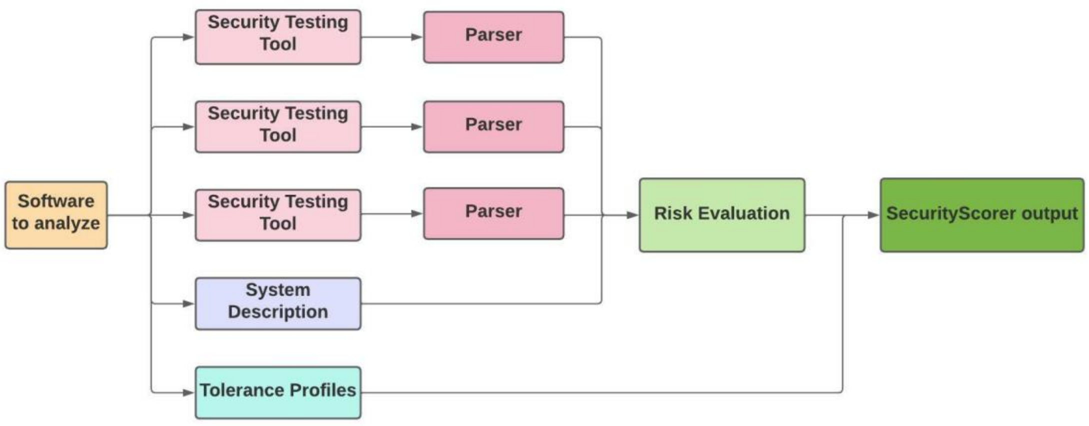


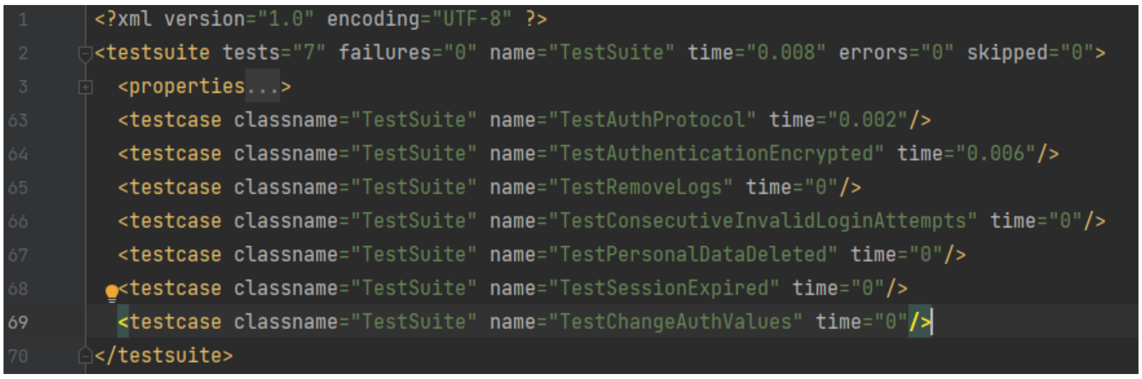
Figure 34 A general idea of security testing and risk evaluation

Next subsections describe the outputs of the three testing tools considered within BIECO that are used by the security scorer to calculate the numerical risk value.

3.4.1. GraphWalker

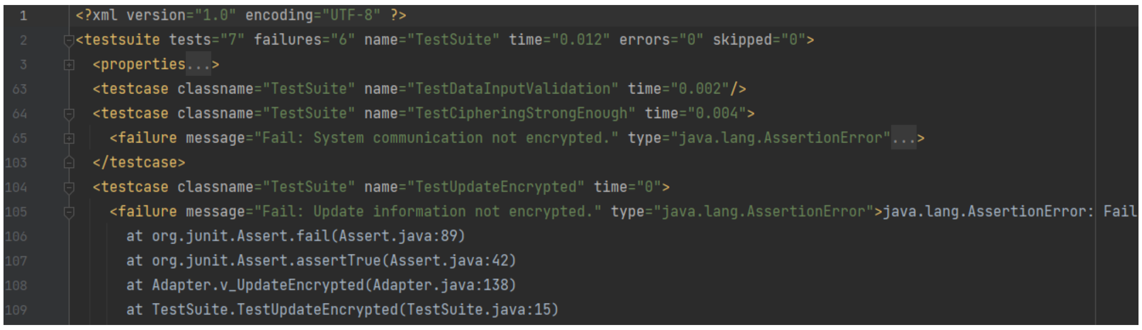
To give an example, this section shows and details a specific output from GraphWalker tool (see Section 3.3.1). After executing all the steps to generate the tests using GraphWalker and execute them, the output is 'TEST-TestSuite.xml' (Figure 35 and Figure 36), which contains the results of each test (pass/fail), describing the failing reasons and the assertions messages with the relevant information.

[9] https://github.com/7bulls/security-scorer-public.git



```xml
<?xml version="1.0" encoding="UTF-8" ?>
<testsuite tests="7" failures="0" name="TestSuite" time="0.008" errors="0" skipped="0">
  <properties...>
  <testcase classname="TestSuite" name="TestAuthProtocol" time="0.002"/>
  <testcase classname="TestSuite" name="TestAuthenticationEncrypted" time="0.006"/>
  <testcase classname="TestSuite" name="TestRemoveLogs" time="0"/>
  <testcase classname="TestSuite" name="TestConsecutiveInvalidLoginAttempts" time="0"/>
  <testcase classname="TestSuite" name="TestPersonalDataDeleted" time="0"/>
  <testcase classname="TestSuite" name="TestSessionExpired" time="0"/>
  <testcase classname="TestSuite" name="TestChangeAuthValues" time="0"/>
</testsuite>
```

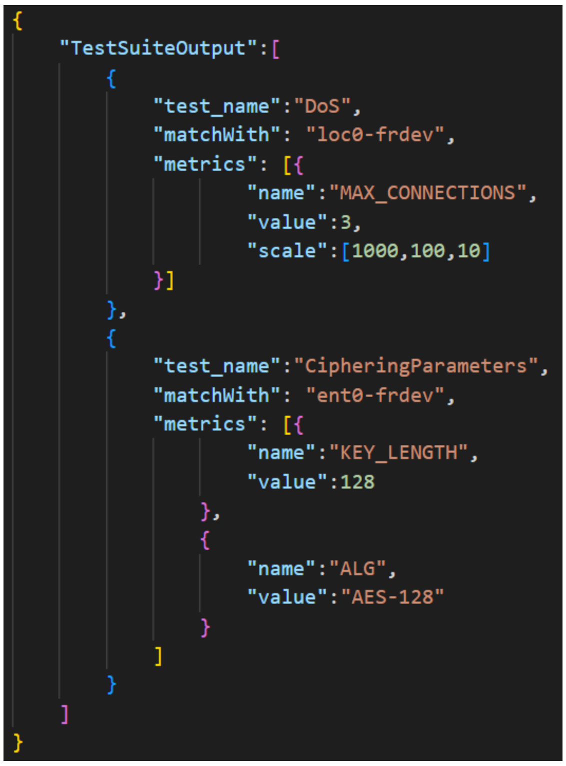
Figure 35 XML output example with all tests passed



```xml
<?xml version="1.0" encoding="UTF-8" ?>
<testsuite tests="7" failures="6" name="TestSuite" time="0.012" errors="0" skipped="0">
  <properties...>
  <testcase classname="TestSuite" name="TestDataInputValidation" time="0.002"/>
  <testcase classname="TestSuite" name="TestCipheringStrongEnough" time="0.004">
    <failure message="Fail: System communication not encrypted." type="java.lang.AssertionError"...>
  </testcase>
  <testcase classname="TestSuite" name="TestUpdateEncrypted" time="0">
    <failure message="Fail: Update information not encrypted." type="java.lang.AssertionError">java.lang.AssertionError: Fail
      at org.junit.Assert.fail(Assert.java:89)
      at org.junit.Assert.assertTrue(Assert.java:42)
      at Adapter.v_UpdateEncrypted(Adapter.java:138)
      at TestSuite.TestUpdateEncrypted(TestSuite.java:15)
```

Figure 36 XML output example with failed tests

As some of the Graphwalker tests could be non-binary, that means that specific values can be returned instead of PASS or FAIL. The second output of the test execution is the file TestSuite-output.json (Figure 37), which collects the set of metrics derived of each test, described in a JSON format. The content found in this file, shown in Figure 37, describes the test name from which the metrics come ('test_name'), the name and value of the metric ('name', 'value') and the connection name ('matchWith') to which these values apply. To calculate the likelihood associated with these tests, the returned value should be mapped to a value between 0 and 1 using a scale, which is also indicated on the file for each metric. This file is later used by the SecurityScorer tool to calculate the likelihood associated.



```
{
    "TestSuiteOutput":[
        {
            "test_name":"DoS",
            "matchWith": "loc0-frdev",
            "metrics": [{
                    "name":"MAX_CONNECTIONS",
                    "value":3,
                    "scale":[1000,100,10]
            }]
        },
        {
            "test_name":"CipheringParameters",
            "matchWith": "ent0-frdev",
            "metrics": [{
                    "name":"KEY_LENGTH",
                    "value":128
                },
                {
                    "name":"ALG",
                    "value":"AES-128"
                }
            ]
        }
    ]
}
```

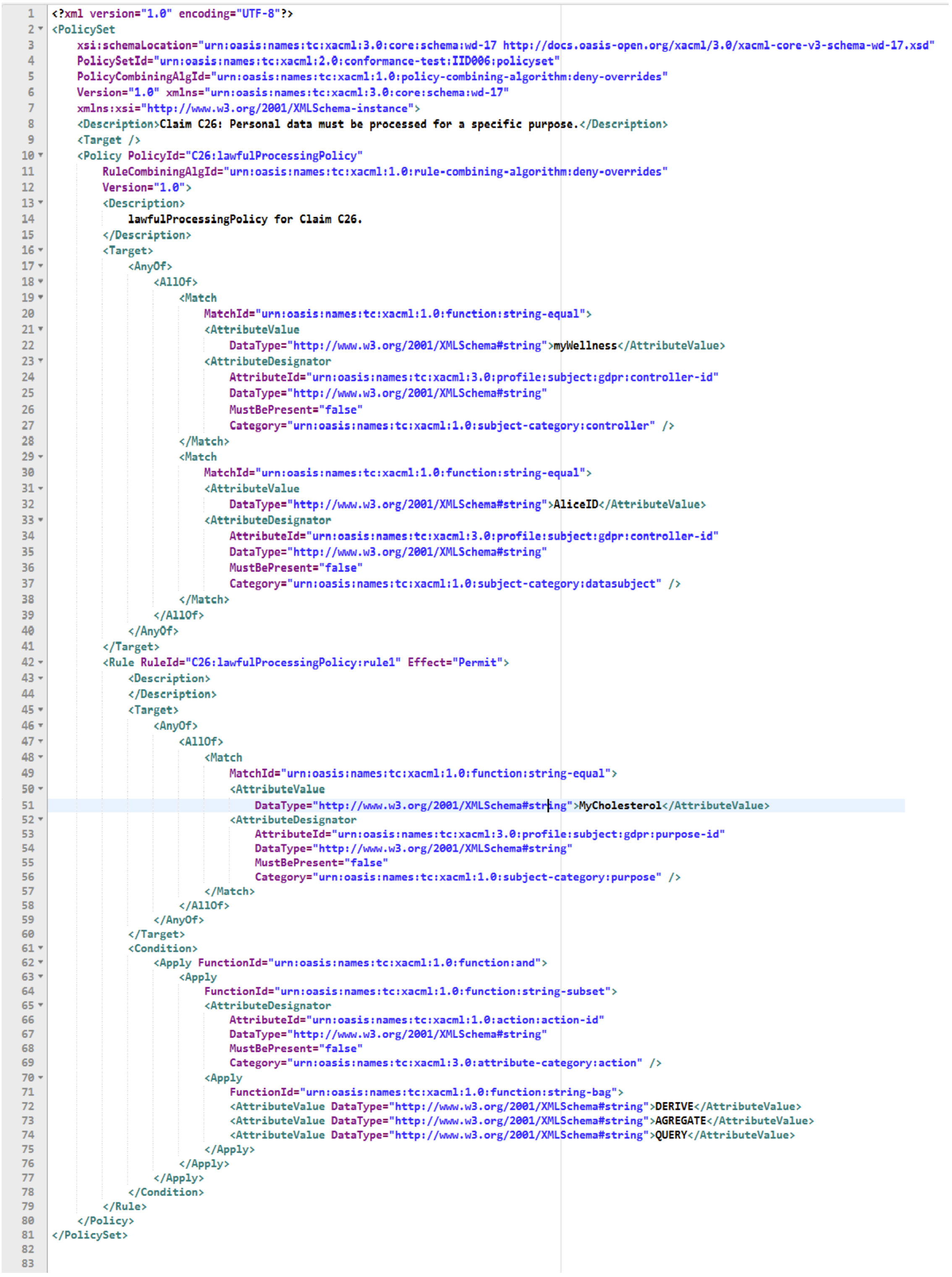
Figure 37 Test metrics and values example 'TestSuite-output.json'

3.4.2. GROOT

In the following, we report an example of an entry of the report produced by GROOT, and more precisely by the Oracle component depicted in Figure 29.

A report entry is composed of:

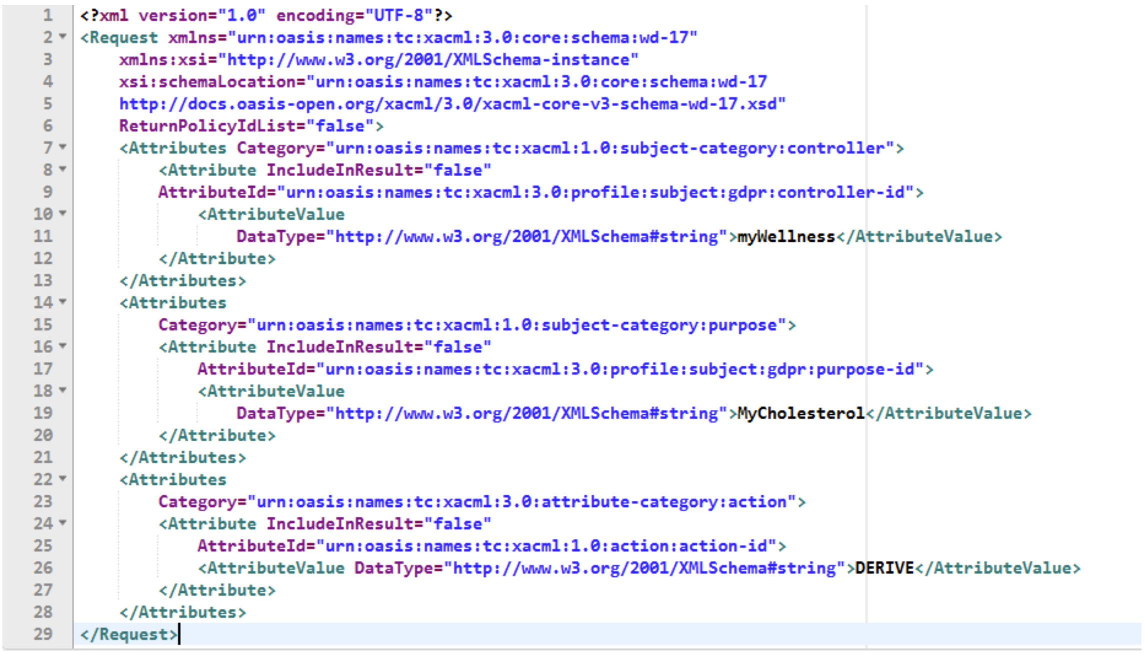
1) A GDPR-based Access Control Policy. Below in Figure 38, we report an extract of the policy reported in Figure 32 expressed in XACML language.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet
    xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17 http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
    PolicySetId="urn:oasis:names:tc:xacml:2.0:conformance-test:IID006:policyset"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides"
    Version="1.0" xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Description>Claim C26: Personal data must be processed for a specific purpose.</Description>
    <Target />
    <Policy PolicyId="C26:lawfulProcessingPolicy"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"
        Version="1.0">
        <Description>
            lawfulProcessingPolicy for Claim C26.
        </Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match
                        MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue
                            DataType="http://www.w3.org/2001/XMLSchema#string">myWellness</AttributeValue>
                        <AttributeDesignator
                            AttributeId="urn:oasis:names:tc:xacml:3.0:profile:subject:gdpr:controller-id"
                            DataType="http://www.w3.org/2001/XMLSchema#string"
                            MustBePresent="false"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:controller" />
                    </Match>
                    <Match
                        MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue
                            DataType="http://www.w3.org/2001/XMLSchema#string">AliceID</AttributeValue>
                        <AttributeDesignator
                            AttributeId="urn:oasis:names:tc:xacml:3.0:profile:subject:gdpr:controller-id"
                            DataType="http://www.w3.org/2001/XMLSchema#string"
                            MustBePresent="false"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:datasubject" />
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
        <Rule RuleId="C26:lawfulProcessingPolicy:rule1" Effect="Permit">
            <Description>
            </Description>
            <Target>
                <AnyOf>
                    <AllOf>
                        <Match
                            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                            <AttributeValue
                                DataType="http://www.w3.org/2001/XMLSchema#string">MyCholesterol</AttributeValue>
                            <AttributeDesignator
                                AttributeId="urn:oasis:names:tc:xacml:3.0:profile:subject:gdpr:purpose-id"
                                DataType="http://www.w3.org/2001/XMLSchema#string"
                                MustBePresent="false"
                                Category="urn:oasis:names:tc:xacml:1.0:subject-category:purpose" />
                        </Match>
                    </AllOf>
                </AnyOf>
            </Target>
            <Condition>
                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
                    <Apply
                        FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-subset">
                        <AttributeDesignator
                            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                            DataType="http://www.w3.org/2001/XMLSchema#string"
                            MustBePresent="false"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action" />
                        <Apply
                            FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
                            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">DERIVE</AttributeValue>
                            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">AGREGATE</AttributeValue>
                            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">QUERY</AttributeValue>
                        </Apply>
                    </Apply>
                </Apply>
            </Condition>
        </Rule>
    </Policy>
</PolicySet>
```

Figure 38 GROOT: A possible policy related to the Lawfulness of Processing Personal Data (Claim C26). It is a possible implementation of the GDPR-based Policy reported in Figure 32

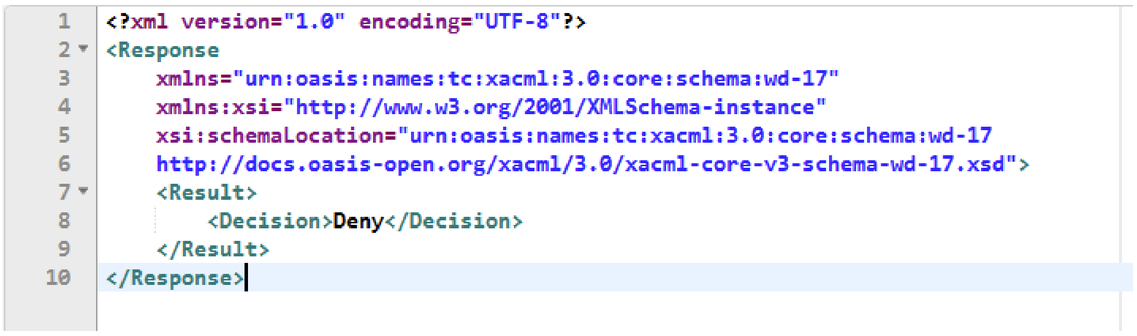
2) An access control request. A possible request generated by GROOT is shown in Figure 39.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
    http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
    ReturnPolicyIdList="false">
    <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:controller">
        <Attribute IncludeInResult="false"
        AttributeId="urn:oasis:names:tc:xacml:3.0:profile:subject:gdpr:controller-id">
            <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">myWellness</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes
        Category="urn:oasis:names:tc:xacml:1.0:subject-category:purpose">
        <Attribute IncludeInResult="false"
            AttributeId="urn:oasis:names:tc:xacml:3.0:profile:subject:gdpr:purpose-id">
            <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">MyCholesterol</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes
        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
        <Attribute IncludeInResult="false"
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">DERIVE</AttributeValue>
        </Attribute>
    </Attributes>
</Request>
```

Figure 39 GROOT: A possible request derived from the policy in Figure 38

3) An authorization response access. An example is reported in Figure 40, where the authorization decision is Deny.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<Response
    xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
    http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd">
    <Result>
        <Decision>Deny</Decision>
    </Result>
</Response>
```

Figure 40 GROOT: A possible decision

3.4.3. Fuzzing Tool

Another example is the one provided by the Fuzzing Tool. This tool provides a report in a JSON format which indicates if the performed tests pass or fail, together with useful information for the user or other tools. The structure the Fuzzing output and the description of each of the parameter is as follow:

{ "passed": [

{

"endpoint": String. *Endpoint tested*,

"operation": String. *HTTP operation type,*

"response_body": String. *Response given by the tested API,*

"response_code": Sring. *Response code given by the tested API*,

"query_headers": String. *Headers used to perform the query,*

"query_body": String. *Body content used to perform the query,*

"expected_code": String. *Expected response code following the specification,*

"default_response": String. *Default response code (given in the specification),*

"expected_len": String. *Expected response length,*

"response_len": String. *Actual response length,*

"input_test": String. *Input given to the test,*

"vulnerability_type": String. *Tested vulnerability type,*

},

...

],

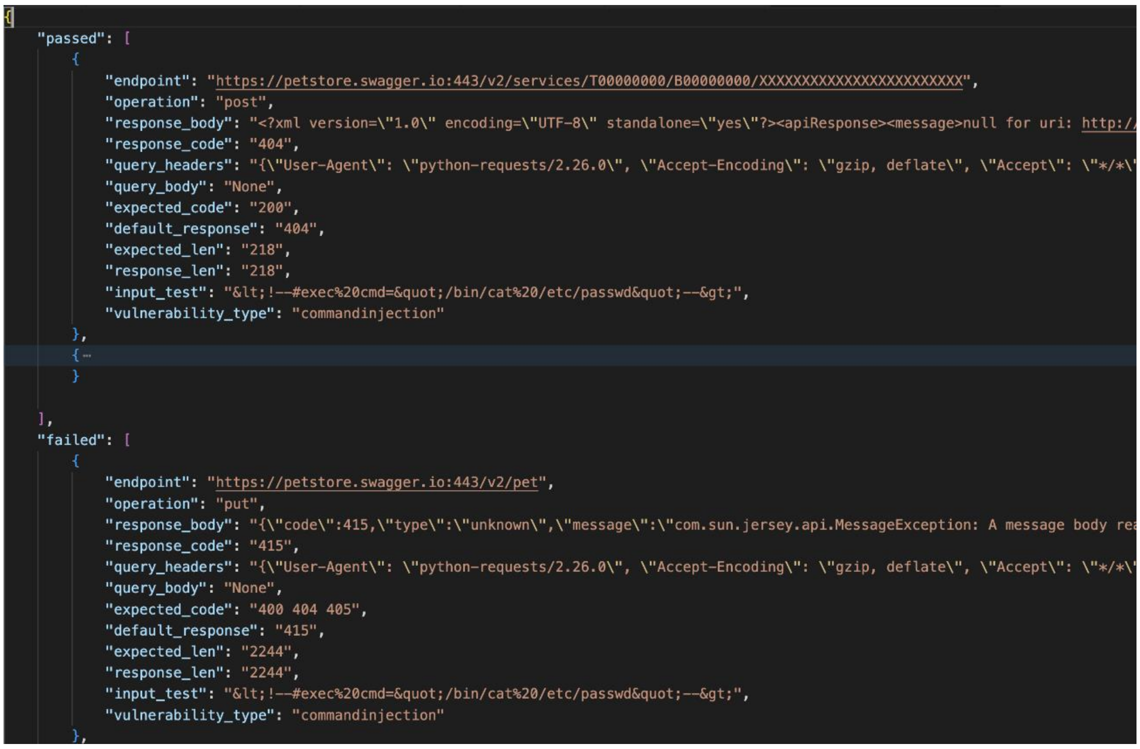
"failed": [

...

]

}

For a more complete vision, in Figure 41 is presented a short example of output.



```
{
    "passed": [
        {
            "endpoint": "https://petstore.swagger.io:443/v2/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX",
            "operation": "post",
            "response_body": "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?><apiResponse><message>null for uri: http:/
            "response_code": "404",
            "query_headers": "{\"User-Agent\": \"python-requests/2.26.0\", \"Accept-Encoding\": \"gzip, deflate\", \"Accept\": \"*/*\"
            "query_body": "None",
            "expected_code": "200",
            "default_response": "404",
            "expected_len": "218",
            "response_len": "218",
            "input_test": "&lt;!--#exec%20cmd=&quot;/bin/cat%20/etc/passwd&quot;--&gt;",
            "vulnerability_type": "commandinjection"
        },
        {…
        }

    ],
    "failed": [
        {
            "endpoint": "https://petstore.swagger.io:443/v2/pet",
            "operation": "put",
            "response_body": "{\"code\":415,\"type\":\"unknown\",\"message\":\"com.sun.jersey.api.MessageException: A message body rea
            "response_code": "415",
            "query_headers": "{\"User-Agent\": \"python-requests/2.26.0\", \"Accept-Encoding\": \"gzip, deflate\", \"Accept\": \"*/*\"
            "query_body": "None",
            "expected_code": "400 404 405",
            "default_response": "415",
            "expected_len": "2244",
            "response_len": "2244",
            "input_test": "&lt;!--#exec%20cmd=&quot;/bin/cat%20/etc/passwd&quot;--&gt;",
            "vulnerability_type": "commandinjection"
        },
```

Figure 41 A shortened version of the Fuzzing output

3.5. Risk Evaluation

This section describes the process of the risk evaluation in the BIECO methodology. As was explained in the previous sections and visually presented in Figure 1, this phase takes place after security testing phase, using as input the data collected from the testing tools and the system description (YAML file) created in the Risk identification phase. All these stages provide an essential set of data to finally evaluate the risk.

The *SecurityScorer* tool was created to realize the functions of the risk evaluation phase. These include five main areas:

1. Parsing the system description file (system decomposition and tolerance profiles);
2. Parsing the outputs of the security testing phase tools ;
3. Using both sources of information to evaluate the risk value for each component;
4. Combine the risks of the components to calculate the overall risk value of the whole system;
5. Use the tolerance profiles to certify the system properties with an appropriate label (A, B, C, D, or not certified).

The in-depth description of the SecurityScorer, including the installation and usage guide, are in the SecurityScorer Anex II at the end of the deliverable. This section intends to present a general view of the functionality implemented for the risk evaluation phase to provide an understanding of the whole methodology.

1. The system description: The scheme was presented in Section **Error! Reference source not found.** of this deliverable and introduced in Deliverable 7.2. SecurityScorer parses the file to build an internal scheme of the system.
2. The outputs of the other tools: SecurityScorer provides a module for each of the risk identification and estimation phase tools (GraphWalker, Groot, Fuzzing Tool) to parse their outputs. These outputs are then internally linked to specific claims or vulnerabilities in the system scheme.
3. Evaluating the risk of the component: Evaluation of the risk for each component follows the procedure described in Deliverable 7.2, Section 6.4 Risk Estimation, paragraphs: From tests to claims, and From claims to components.

3.5.1. Likelihood Calculation

The first phase is to calculate the likelihoods. For the standard tests, it is straightforward: each test is either passed or failed. Therefore, the likelihood is either equal to 0 or 1.

For the non-binary tests, the procedure is more complicated. Section 3.4.1 presents the additional file required to evaluate the risk. It contains a metric and scale for each of the tests with a non-binary result. The scale scheme is [v_1, $v_{2, \dots}$,v_n] (ascending or descending order), meaning that the value range of test results should be divided into n+1 sections:

$$(-\infty, v_1), [v_1, v_2), \dots , [v_{n,}+\infty)$$

(-inf and +inf should be swapped for the descending scale). If the test result is in the first section, the resulting probability is $\frac{1}{n+1}$, if it is in the second section, the probability is $\frac{2}{n+1}$, etc. For the result in the last *(n+1)*-th section, the probability is: $\frac{n+1}{n+1} = 1$.

For example, let us analyze the metric from Figure 37 example: MAX_CONNECTIONS metric with scale [1000, 100, 10] and test result value: 2. The sections of the MAX_CONNECTIONS value space are as follows:

(-inf, 1000), [1000, 100), [100, 10), [10, -inf)

Obviously, the minimal value of the number of connections is 0, but for mathematical consistency we use the established scheme.

The probabilities assigned to each section are respectively:

¼, ½, ¾, 1

Since the test result value 2 belongs to the last section [10, -inf), the likelihood used in the rest of the evaluation for this test is 1.

3.5.2. Risk Calculation

The risk associated with each "pure" claim is calculated as an arithmetic mean of likelihoods from *m* tests scaled by the impact:

$$Risk_{Claim} = Impact_{Claim} \sum \frac{Likelihood_{Test}}{m}.$$

The risk associated with each claim associated with *n* vulnerabilities is calculated as a maximum of likelihoods scaled by the impacts:

$$Risk_{Claim} = \max_{1 \leq i \leq n} \{Impact_i \cdot Likelihood_i\},$$

Where the likelihood of the vulnerability *i*, *Likelihood$_i$*, is calculated as an arithmetic m of likelihoods from *d* tests associated with the vulnerability:

$$Risk_{Claim} = \frac{\sum_d Likelihood_{Test_d}}{d}.$$

Then, the risk of the whole component is just a maximum of the risks of the *r* associated claims:

$$Risk_{Component} = \max_{1 \leq i \leq r} \{Risk_{Claim_i}\}.$$

To evaluate the most high-level risk, i.e., the system property risk, the maximum of the *c* system components weighted by their sensitivities is calculated:

$$Risk_{System\ property} = \max_{1 \leq i \leq c} \left\{Risk_{Component_i} Sensitivity_{Component_i}\right\}.$$

All system properties are calculated this way, generating a set of values in range 0-10, denoting the overall risk evaluated for each system property.

After the whole process of identifying and estimating the risk, parsing the output of the risk estimation tools, evaluating the risk values for claims, components, and the entire system, numerical values are obtained. These are in the range [0,10] and denote the calculated risk for each system property, i.e., Confidentiality, Integrity, Availability, Authorization, Authentication, Non-Repudiation.

3.5.3. Risk Evaluation Against the Tolerance Profiles

By using tolerance profiles, SecurityScorer assigns a label for each category. Possible levels are (from highest): A, B, C, D, and not certified. For example, if the tolerance profile for confidentiality was [2, 4, 6, 7] and the obtained value is 3, then confidentiality is certified as B. The reason is that the tolerance profile implies that all values in the range [2, 4) should be classified as B (a more thorough explanation was presented in Section3.1.2). As a result, the six levels are returned as a result of the risk evaluation phase.

3.6. Labelling

The obtained security levels are represented as a spider chart diagram. Figure 42 shows an example of security label. The higher the green area inside the chart, the more secure the system is, which is an easy concept to understand for a non-expert user.

The label also includes a public QR code to deal with the security changes on the label that may happen due to a recertification. This way, the end user can scan the QR code and access to the updated label. It is worth noting that the label should be complementary to the notion of a certificate, which may include additional details of the evaluation and claims. However, the definition of the certificate is outside the scope of BIECO.

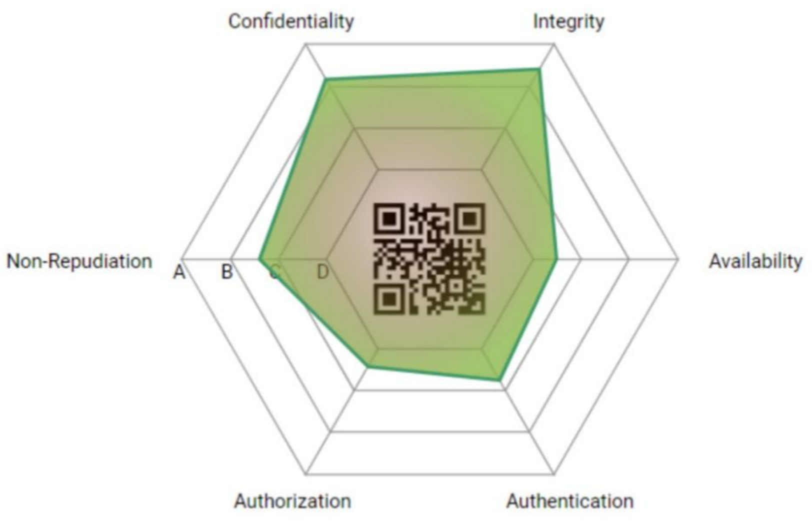


Figure 42 Labelled results of the risk evaluation phase

3.7. Treatment

Based on the results obtained by the BIECO tools, the information contained in the MUD file could be completed or updated. The collected values from the Graphwalker tests output, explained in section 3.4.1, can be used to improve the initial version of the extended MUD, which was originally created after an initial assessment phase in the Resilblockly tool and stored in the Data Collection tool (DCT). Table 2 shows the claims and test that may imply an update of the extended MUD file. The first column is the related claim and the second one the associated test. The third column lists the test values that are used as input for the MUD update, and the last column indicates which field of the extended MUD is updated and under which conditions.

Table 2 Mapping between testing outputs and extended MUD fields

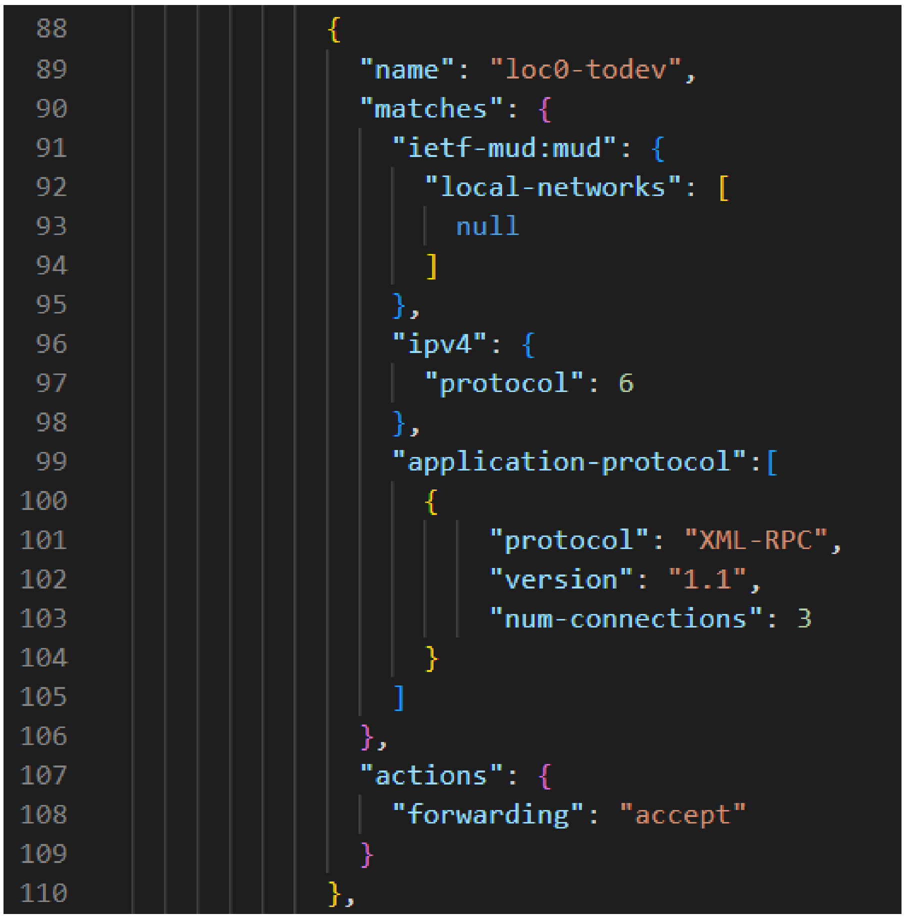
Claim	Test	Test values	Field
C9	1. Confidentiality: strength of Confidentiality parameters in communications	• **ALG** • **KEY_LENGTH**	**Keys/alg, keys/length** Where keys/purpose=conf and Where keys/key_ops=derive key
C19	1. Privacy: strength of confidentiality parameters for private data	1. **ALG** 2. **KEY_LENGTH**	**Keys/alg, keys/length** Where keys/purpose=conf and Where keys/key_ops=encry pt
C10	1. Authentication: strength of authn parameters in communications	• **ALG** • **KEY_LENGTH**	**Keys/alg, keys/length** Where keys/purpose=authn and Where keys/key_ops=encry pt
C20	2. Authentication: strength of authn parameters in authn process	• **ALG** • **KEY_LENGTH**	**Keys/alg, keys/length** Where keys/purpose=authn and Where keys/key_ops=derive key
C14, C17	1. Ciphering: strength of ciphering parameters in communications	• **ALG** • **KEY_LENGTH**	**Keys/alg, keys/length**

	2. Ciphering: secure encryption of sensitive parameters		Where keys/purpose=ciph and Where keys/key_ops=encry pt
C21	3. Integrity: strength of ciphering parameters in communications	**1. ALG** **2. LENGTH**	**Keys/alg, keys/length** Where keys/purpose=sign or verify and Where keys/key_ops=Integri ty
C22	1. Resistance DoS attacks	**2. MAX_CONNECTION S**	**num-connections**
C32, C43 (vulne rability claim)	1. Tests to verify if the vulnerabilities identified in Resilblockly or WP3 tools are present or not	**2. PASS/FAIL** **3. Additional info may be required (e.g., from the test name) to identify the vulnerability being tested**	**Vulnerabilities, weaknesses** Where ID coincides remove if test passes
C40, C18	1. Check compliance of REST interfaces with swagger file (fuzzing tool)	**2. PASS/FAIL** **3. Additional info may be required (e.g., from the test name) to identify the resource url** **4. Other info: version, method, resource**	**Application-protocol (HTTP)/resource** Where URL coincides Remove method if test fails (Whole resource if method will be empty) Add new block if test passes and it is not in the MUD file.
-	-	**-**	**last-update** Current date of MUD update
-	-	-	**mud-signature** Update MUD signature with Last version of MUD file

In order to reach this update, BIECO has developed the tool "MUD Updater" that uses the metrics derived from the tests execution, which are collected in the 'TestSuite-output.json' file (generated as explained in section 3.4.1). The description of the connections contained in the MUD file is updated or completed based on this output, being able to locate the connection corresponding to each metric thanks to the field 'matchWith'.

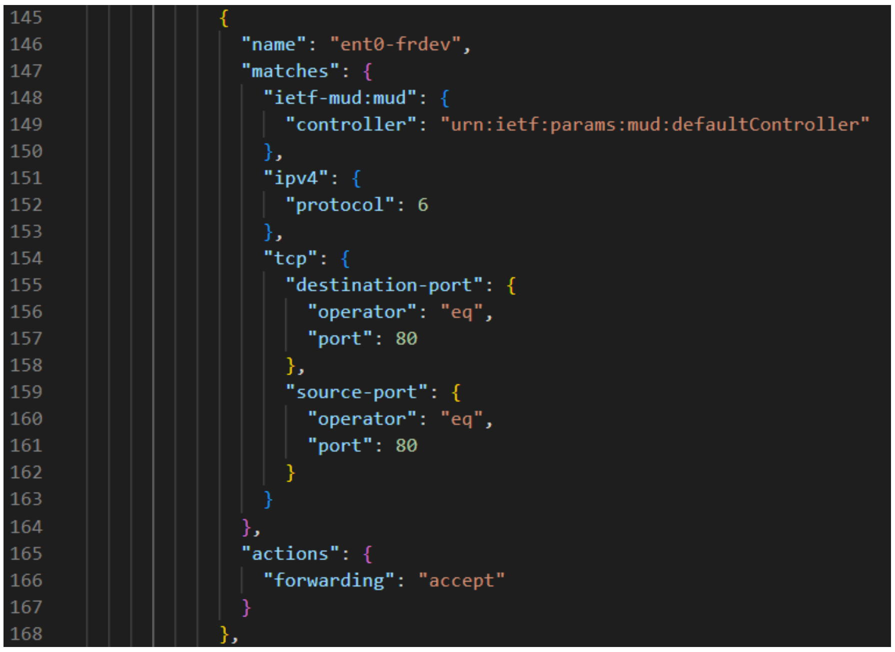
To illustrate the operation of this tool the following example is shown:

1. On the one hand we have the extended MUD file that describes the SUT connections:



```
{
  "name": "loc0-todev",
  "matches": {
    "ietf-mud:mud": {
      "local-networks": [
        null
      ]
    },
    "ipv4": {
      "protocol": 6
    },
    "application-protocol":[
      {
        "protocol": "XML-RPC",
        "version": "1.1",
        "num-connections": 3
      }
    ]
  },
  "actions": {
    "forwarding": "accept"
  }
},
```

Figure 43 MUD file fragments of interest for this example (1)

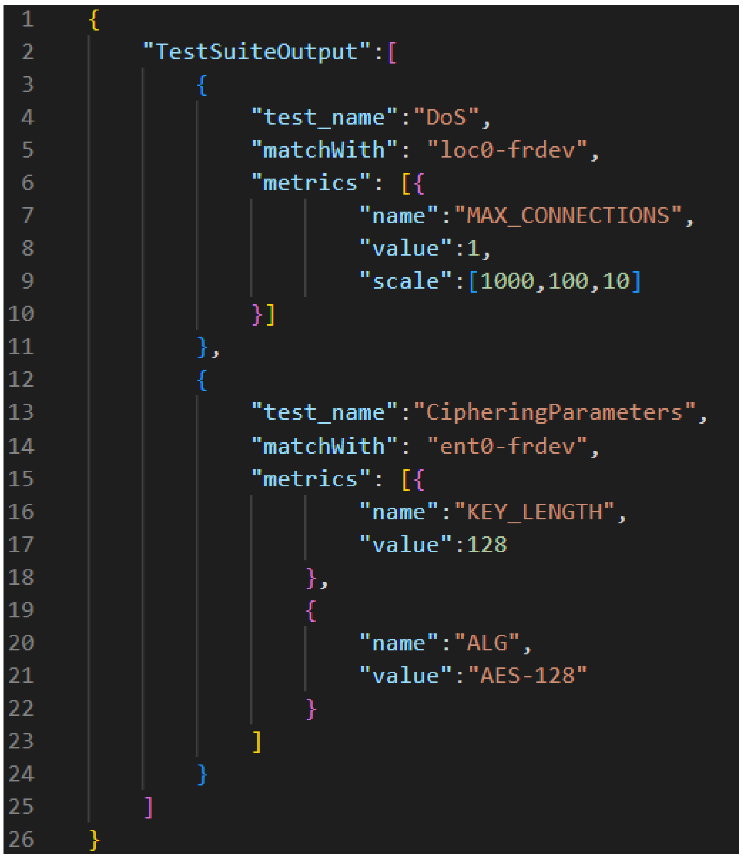


```
        {
          "name": "ent0-frdev",
          "matches": {
            "ietf-mud:mud": {
              "controller": "urn:ietf:params:mud:defaultController"
            },
            "ipv4": {
              "protocol": 6
            },
            "tcp": {
              "destination-port": {
                "operator": "eq",
                "port": 80
              },
              "source-port": {
                "operator": "eq",
                "port": 80
              }
            }
          },
          "actions": {
            "forwarding": "accept"
          }
        },
```

Figure 44 MUD file fragments of interest for this example (2)

In this case, connection names are 'loc0-todev' and 'ent0-ftdev'.

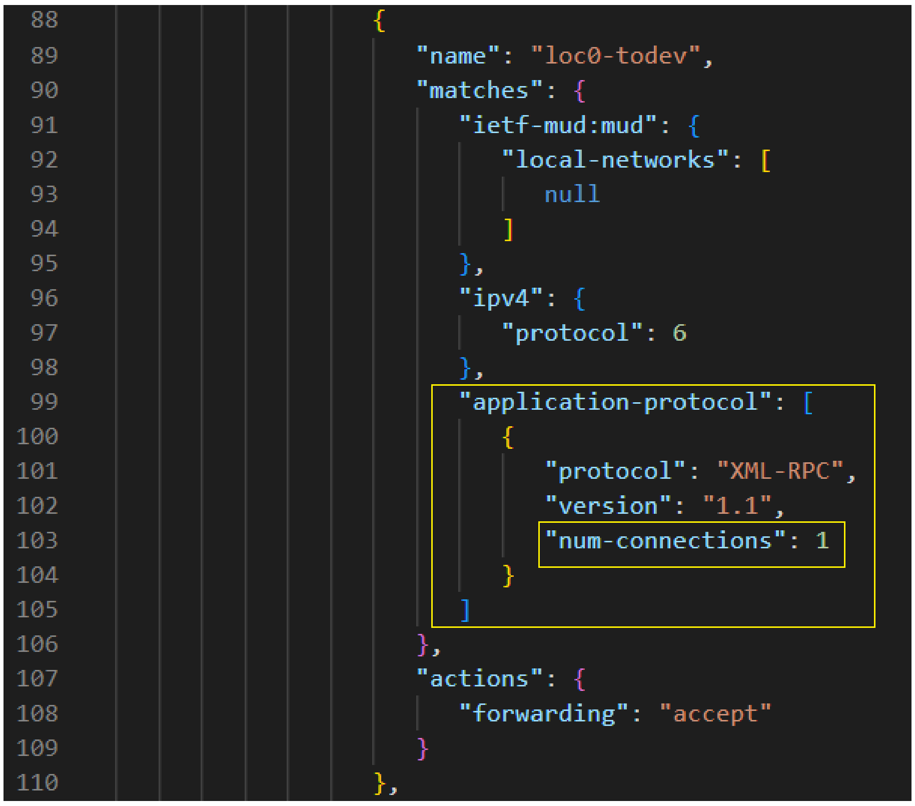
2. On the other hand, the output file containing the metrics values information:



```
{
    "TestSuiteOutput":[
        {
            "test_name":"DoS",
            "matchWith": "loc0-frdev",
            "metrics": [{
                    "name":"MAX_CONNECTIONS",
                    "value":1,
                    "scale":[1000,100,10]
            }]
        },
        {
            "test_name":"CipheringParameters",
            "matchWith": "ent0-frdev",
            "metrics": [{
                    "name":"KEY_LENGTH",
                    "value":128
                },
                {
                    "name":"ALG",
                    "value":"AES-128"
                }
            ]
        }
    ]
}
```

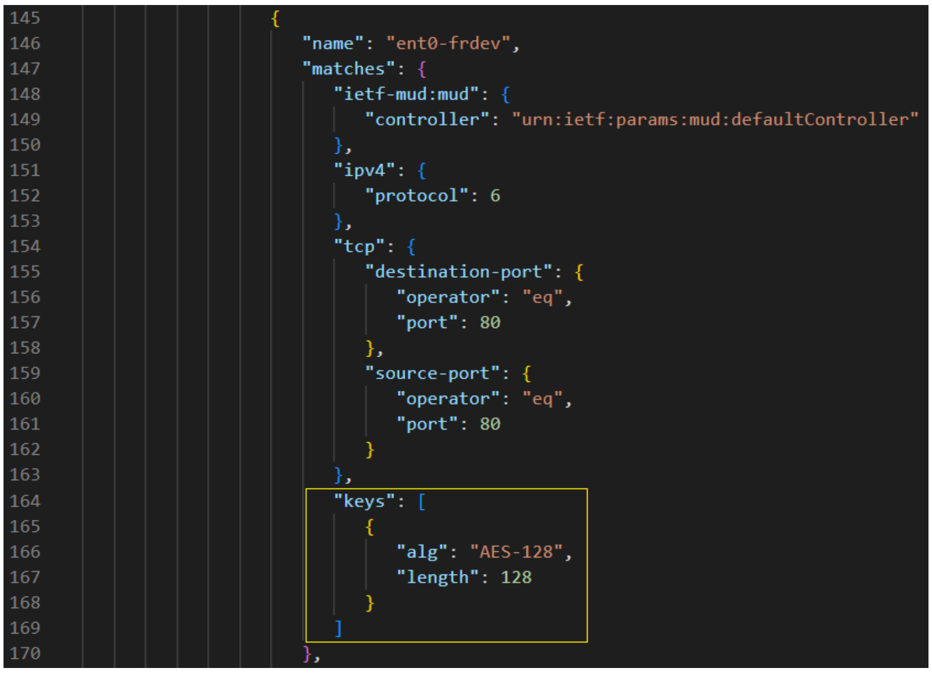
Figure 45 Metrics file, 'TestSuite-output.json'

3. Based on the information of these metrics ('MAX_CONNECTIONS', 'KEY_LENGTH' and 'ALG') the MUD file fields are updated with the new information derived from the tests.



```
{
    "name": "loc0-todev",
    "matches": {
        "ietf-mud:mud": {
            "local-networks": [
                null
            ]
        },
        "ipv4": {
            "protocol": 6
        },
        "application-protocol": [
            {
                "protocol": "XML-RPC",
                "version": "1.1",
                "num-connections": 1
            }
        ]
    },
    "actions": {
        "forwarding": "accept"
    }
},
```

Figure 46 Connection 'loc0-todev' information updated



```
{
    "name": "ent0-frdev",
    "matches": {
        "ietf-mud:mud": {
            "controller": "urn:ietf:params:mud:defaultController"
        },
        "ipv4": {
            "protocol": 6
        },
        "tcp": {
            "destination-port": {
                "operator": "eq",
                "port": 80
            },
            "source-port": {
                "operator": "eq",
                "port": 80
            }
        },
        "keys": [
            {
                "alg": "AES-128",
                "length": 128
            }
        ]
    },
```

Figure 47 Connection 'ent0-frdev' completed with extra information

Hence, for the first case Figure 46 the number of simultaneous connections allowed is updated from 3 to 1. In the second case, Figure 47, the field 'keys' is added, completing the previous information in the MUD. This tool will be also integrated inside the BIECO platform within WP8 and connected with the DCT for the retrieval of the extended MUD and the storage of the updated version.

3.8. Communication and Auditing

Communication and auditing deals with the security changes that may happen during the lifecycle of the system. The methodology can be supported in this phase by the Auditing Framework developed in WP5, which is intended to detect security issues based on a set of blueprints coming from the design phase and manually specified by the user, see Deliverable D5.2[10]. One of the blueprints used in the auditing framework is the updated MUD generated from the security evaluation methodology (treatment phase, section 3.7), which contains specific configuration that should be controlled to keep the system in a secure state according to the test results, as shown in Figure 48.

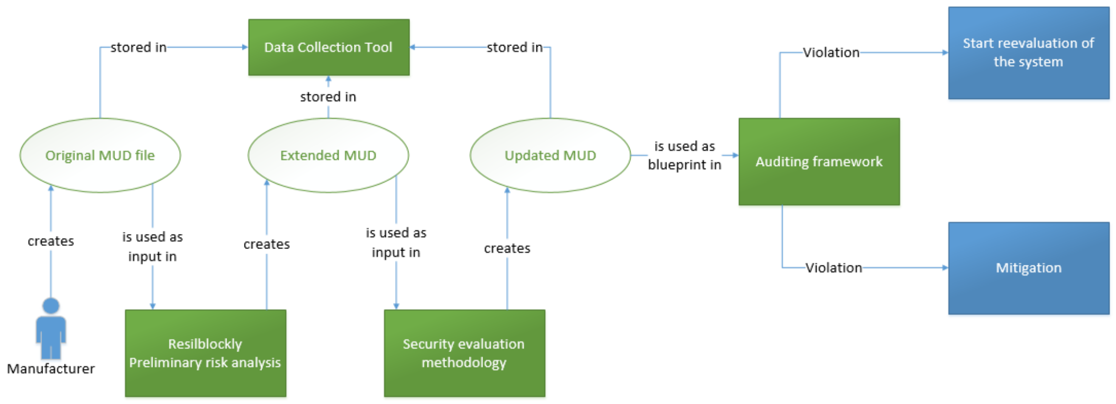


Figure 48 Usage of the extended MUD file for auditing

The used blueprints are converted into rules to be monitored, as shown in Figure 49.

[10] Deliverable 5.2:"First version of the simulation environment and monitoring"

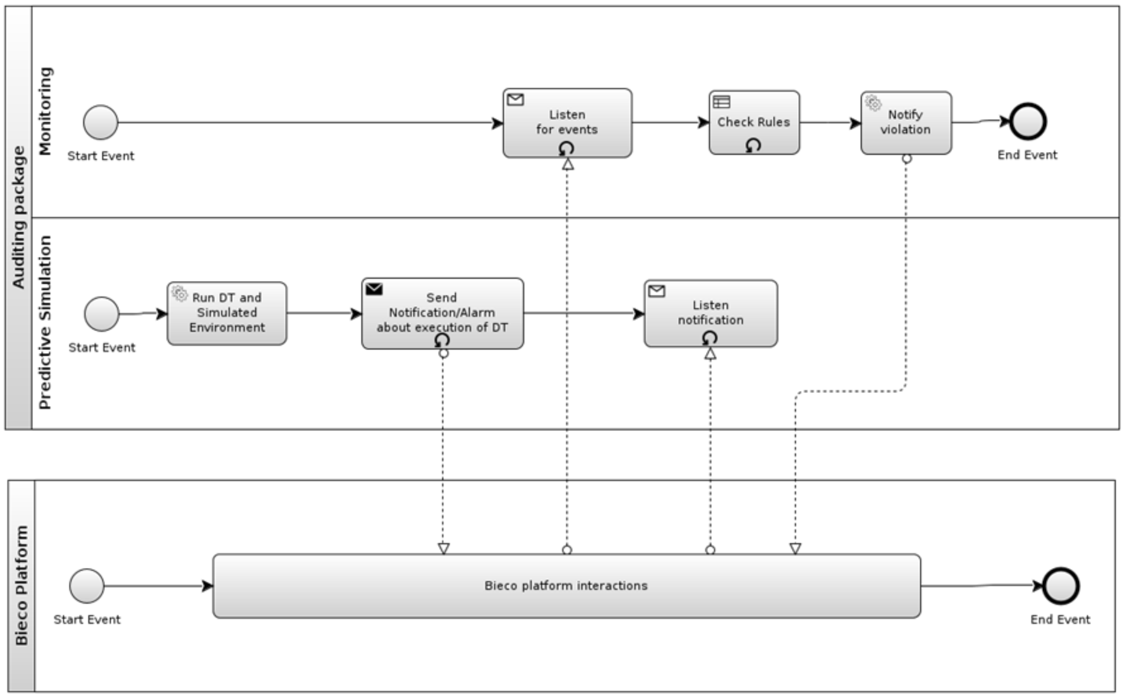


Figure 49 Auditing Violation and Alarm Notification (Adopted from D5.1)

For each event notified to the monitoring platform, the Complex Event Processor (CEP) check if one or more rules will be matched or a new pattern of interest is generated. If a rule is matched, a notification containing the kind of violation and all the available information for debugging, will be provided to the BIECO Platform. In particular, as described in D5.2, Section 3.2.1 (Notification Alarm by the Runtime Monitoring) the Runtime Monitor component in charge of the notification management is the Notification Manager, that manages the notification of failure sent by the CEP and forwards the notification of failure to the specific channel gathering the correct information (channels details) from the ChannelRegistry component.

Although countermeasures could be provided as soon as a violation is detected and notified to the monitored system (outside the scope of BIECO), some security issues may require a revaluation of the system, if they affect the compliance of a specific claim (see Figure 48).

4. Proof of Concept – Application Over UC4

Figure 50 shows the toolchain used to instantiate part of the methodology within the UC4 developed by UNINOVA. We focused on the security testing and evaluation part, as the majority of the tools used for risk identification are still under development. A more complete validation will be performed in WP8.

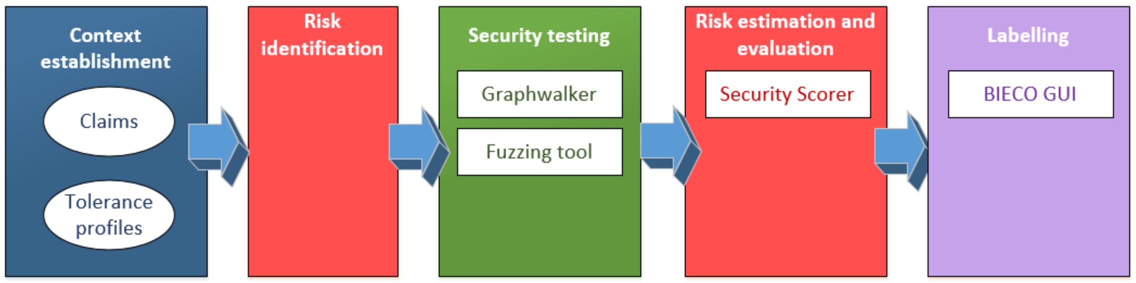


Figure 50 Toolchain used for the UC4 validation

The SUT considered for the evaluation is the local planner component (Figure 51). This component is in charge of producing the velocity commands for the robots according to a specific global plan.

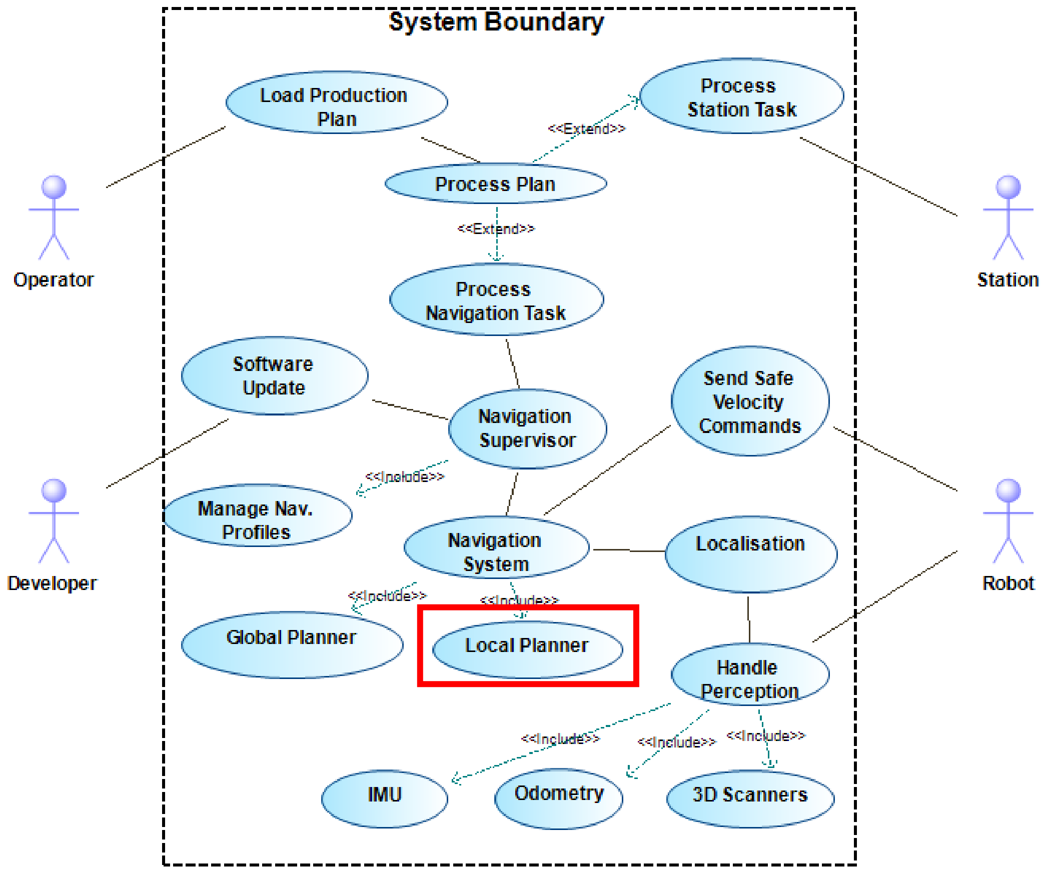


Figure 51 SUT in UC4

4.1 Context establishment: claims selection

We selected a set of claims from D7.1 to evaluate the security of the SUT, and we associated a set of tests to verify the compliance of the claim. Each claim has an impact vector taking into account the 4 dimensions of the methodology (safety, financial, operational and privacy). These values were established manually.

- **C0: Update software files should be encrypted and be transmitted using encryption – Impact[S0,F0,O1,P0]=1**
  - Test1 – Confidentiality1: Update LocalPlanner component in order to check if updates are encrypted or not.
  - Test2 – Confidentiality2 (depends on Test6): Update LocalPlanner component in order to check if encryption used in updates is strong enough.
- **C5: The exchanged messages in the communication should be integrity protected – Impact[S0,F10,O0,P0]=2**
  - Test3 – Integrity1: Create an item and tasks for the Navigator, generating the plan and velocity commands needed to reach a new position for the robot. Send modified command and analyse if system is capable of detecting the modification (MITM, Man In The Middle).
- **C14: Ciphered communications should use strong algorithms – Impact[S0,F0,O0,P0]=0**
  - Test4 – Confidentiality3 (depends on test1): Create an item and tasks for the Navigator, generating the plan and velocity commands needed to reach a new position for the robot. Send correct command and analyse if ciphering used is strong enough.
- **C22: Resistance to DoS attacks - Impact[S0,F10,O100,P0]=7**
  - Test5 – Availability1: DDoS attack based on send/request new velocity commands to the robot. Calculate how many simultaneous requests is capable to process before crashing.
- **C23: Data input validation - Impact[S10,F0,O100,P0]=7**
  - Test6 – Availability2: Create an item and tasks for the Navigator, generating the plan and velocity commands needed to reach a new position for the robot. Send non valid command and analyse if system continues working and manages properly the error.
  - Test1b – Availability1b: Send requests to the endpoint with special parameters. Check if the endpoint crash.
  - Test2b – Availability2b: Send requests to the endpoint with special parameters. Check if the endpoint crash.
  - …
  - TestNb – AvailabilityNb: Send requests to the endpoint with special parameters. Check if the endpoint crash.
- **C24: Data Communications should be ciphered - Impact[S0,F0,O0,P0]=0**
  - Test7 – Confidentiality4: Create an item and tasks for the Navigator, generating the plan and velocity commands needed to reach a new position for the robot. Send correct command and analyze if communications are ciphered between the different components.

4.2 Risk Identification

Figure 52 shows the system description mapping the tests with the claims, the claims with the system components (as the SUT is a single component, all the claims are mapped to it), and the system components to the security properties evaluated, in this case Integrity, Availability and Confidentiality.

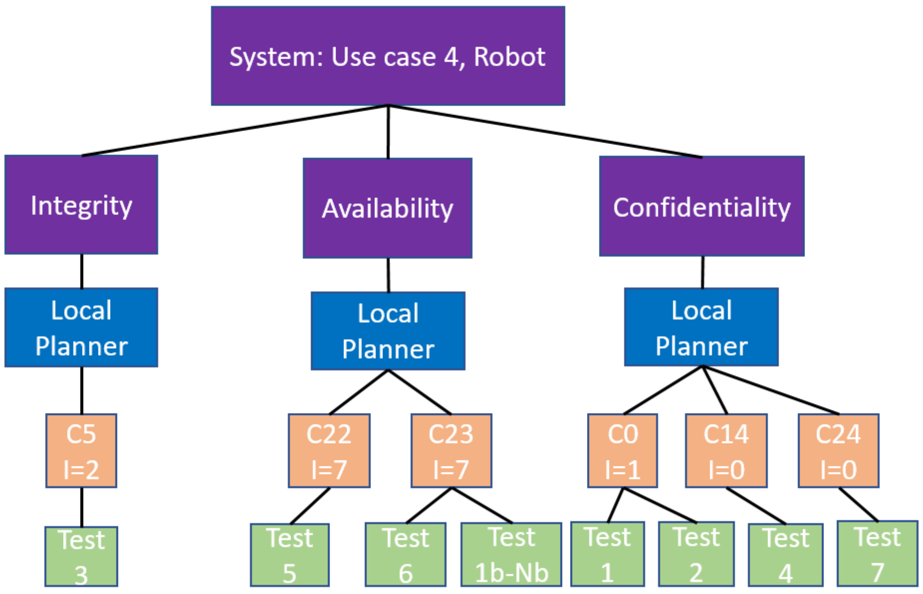


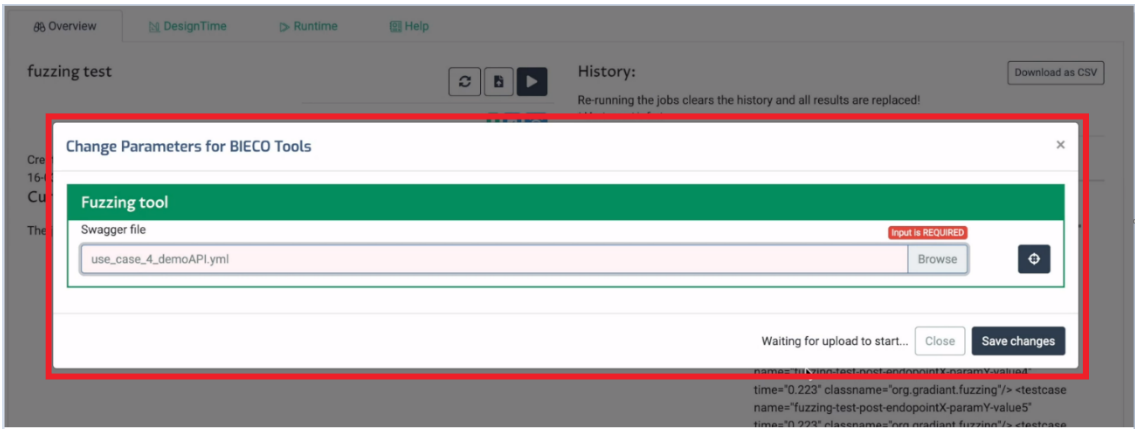
Figure 52 System decomposition for UC4

4.3 Security testing

The tests were implemented using the Graphwalker and fuzzing tool. Next subsections detail the implementation and execution of the tests.

4.3.1. Fuzzing Tool

Once the platform to be analysed has been identified, the tool is put into process. To do this, the platform must be active and ready to receive requests. As previous mentioned, the tool requires for its operation the swagger file that describes the platform to be analysed (Figure 53).



fuzzing test

History:

Re-running the jobs clears the history and all results are replaced!

Change Parameters for BIECO Tools

Fuzzing tool

Swagger file

Input is REQUIRED

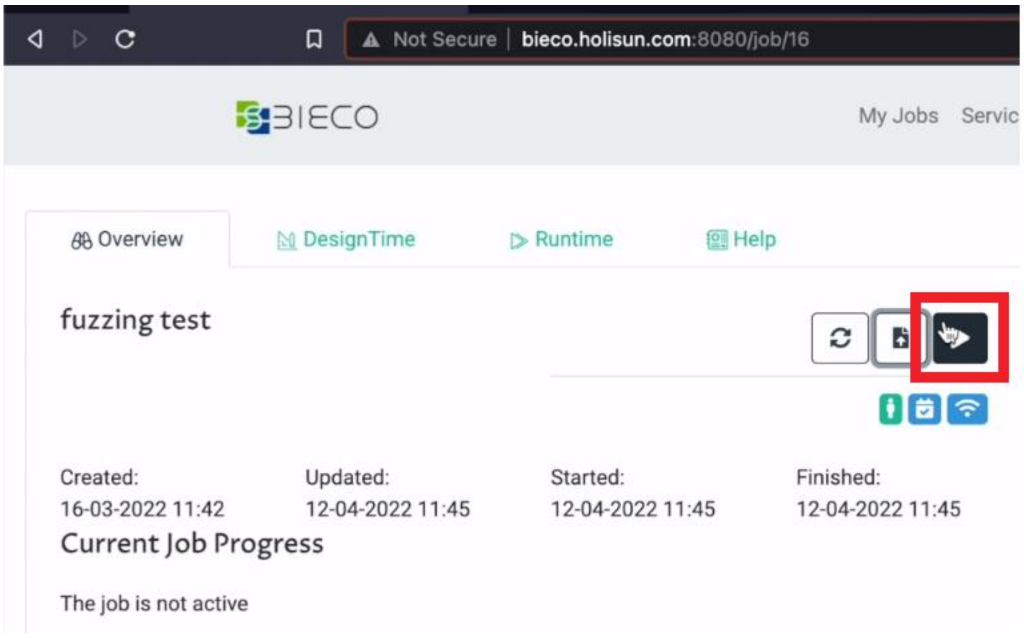
use_case_4_demoAPI.yml

Waiting for upload to start...

```
time="0.223" classname="org.gradiant.fuzzing"/> <testcase
name="fuzzing-test-post-endopointX-paramY-value5"
```

Figure 53 Uploading of the swagger file in the Fuzzing tool

Once the swagger file is uploaded to the platform, the Fuzzing tool can proceed to its execution (Figure 54).



fuzzing test

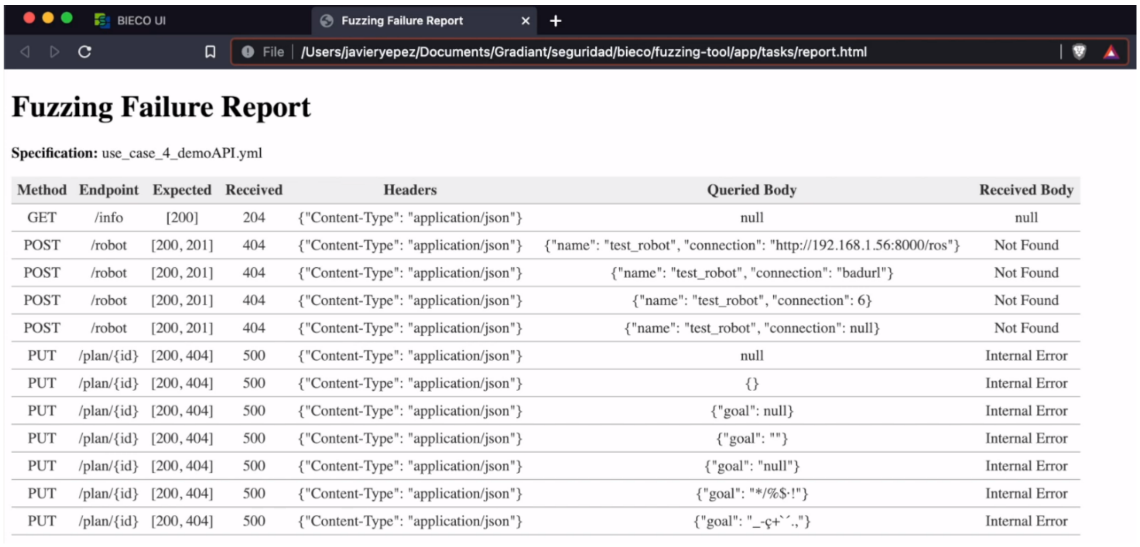
Created: 16-03-2022 11:42
Updated: 12-04-2022 11:45
Started: 12-04-2022 11:45
Finished: 12-04-2022 11:45

Current Job Progress

The job is not active

Figure 54 Execution of the Fuzzing tool

At this point, the tool sends a multitude of requests to the API under test combining the parameters of the same in order to obtain those that has not been contemplated previously or cause an error. This process may take time depending on the complexity of the analysed API. Once the process ends, the tool generates two reports: one that is delivered to the BIECO platform to be used by other tools such as the security scorer, and another for the user (Figure 55) with the suspicious inputs that can affect the platform and their corresponding outputs.


BIECO UI

Fuzzing Failure Report

File | /Users/javieryepez/Documents/Gradiant/seguridad/bieco/fuzzing-tool/app/tasks/report.html

Fuzzing Failure Report

Specification: use_case_4_demoAPI.yml

Method	Endpoint	Expected	Received	Headers	Queried Body	Received Body
GET	/info	[200]	204	{"Content-Type": "application/json"}	null	null
POST	/robot	[200, 201]	404	{"Content-Type": "application/json"}	{"name": "test_robot", "connection": "http://192.168.1.56:8000/ros"}	Not Found
POST	/robot	[200, 201]	404	{"Content-Type": "application/json"}	{"name": "test_robot", "connection": "badurl"}	Not Found
POST	/robot	[200, 201]	404	{"Content-Type": "application/json"}	{"name": "test_robot", "connection": 6}	Not Found
POST	/robot	[200, 201]	404	{"Content-Type": "application/json"}	{"name": "test_robot", "connection": null}	Not Found
PUT	/plan/{id}	[200, 404]	500	{"Content-Type": "application/json"}	null	Internal Error
PUT	/plan/{id}	[200, 404]	500	{"Content-Type": "application/json"}	{}	Internal Error
PUT	/plan/{id}	[200, 404]	500	{"Content-Type": "application/json"}	{"goal": null}	Internal Error
PUT	/plan/{id}	[200, 404]	500	{"Content-Type": "application/json"}	{"goal": ""}	Internal Error
PUT	/plan/{id}	[200, 404]	500	{"Content-Type": "application/json"}	{"goal": "null"}	Internal Error
PUT	/plan/{id}	[200, 404]	500	{"Content-Type": "application/json"}	{"goal": "*/%$·!"}	Internal Error
PUT	/plan/{id}	[200, 404]	500	{"Content-Type": "application/json"}	{"goal": "_-ç+`´.,"}	Internal Error

Figure 55 Output database information

The different information provided by the report is:

- Method: HTTP operation used to send the requests to the platform.
- Endpoint: the different paths to which the platform requests are sent.
- Expected: Represents the response codes of the requests expected to be received as indicated in the swagger file for a specific operation and path.
- Received: The code obtained in the request sent to a specific path and operation with each combination of parameters.
- Headers: Parameters sent in the request header.
- Queried Body: Parameters sent in the request body.
- Received Body: Message obtained in the response to the request made.

4.3.1. Graphwalker

Figure 56 shows the model of the local planner component of the Use Case 4 necessary to generate 7 tests for confidentiality, integrity and availability.

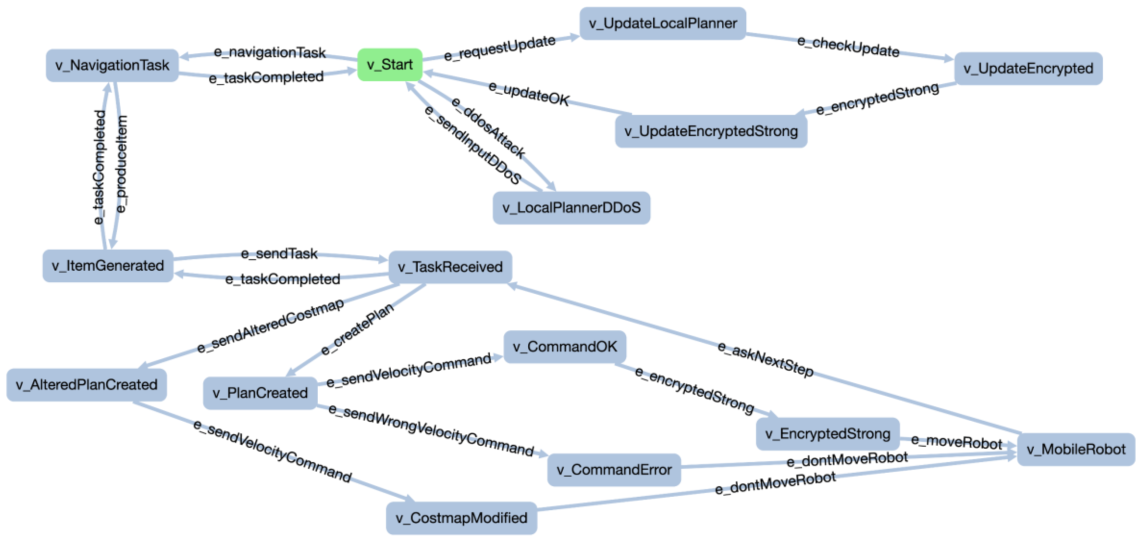
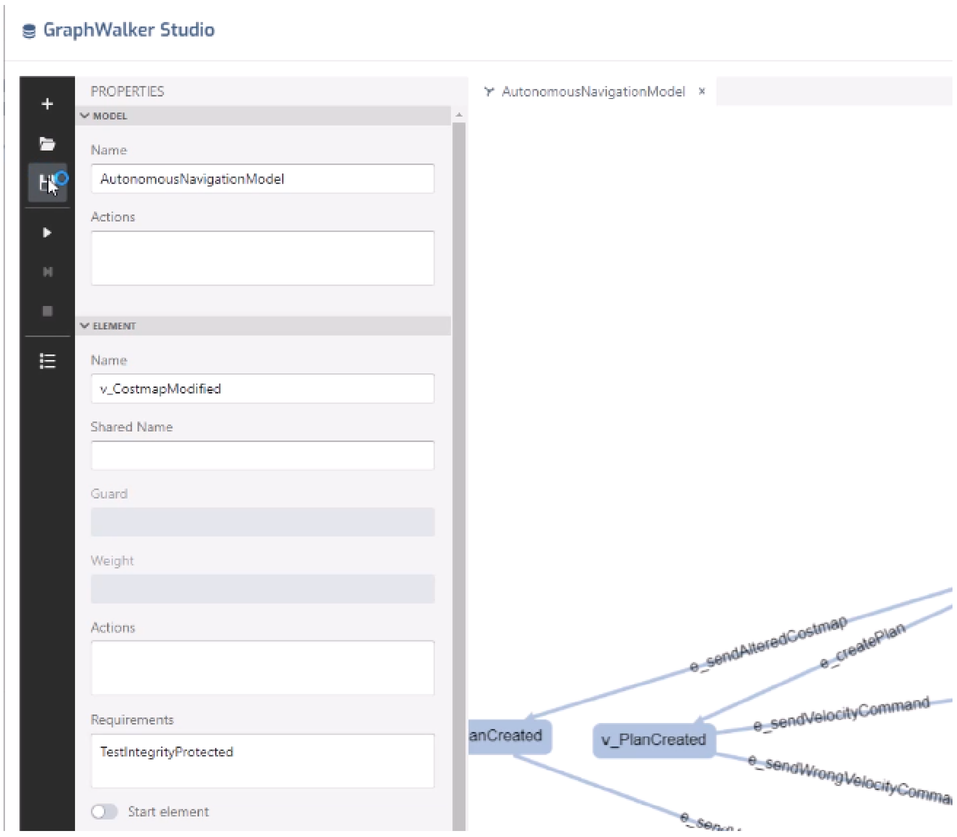


Figure 56 Localplanner modeled with Graphwalker Studio

While in general the tests will be binary (PASS or Fail), some tests will measure certain aspects, for example the maximum number of simultaneous connections before the system crashes (Test 5).

The tests are labelled using the tag requirements, which is used as a coverage condition for the automated test generation. Figure 57 shows the tag for the integrity test (TestIntegrityProtected).


GraphWalker Studio

PROPERTIES

MODEL

Name: AutonomousNavigationModel

Actions:

ELEMENT

Name: v_CostmapModified

Shared Name:

Guard:

Weight:

Actions:

Requirements: TestIntegrityProtected

Start element

Figure 57 Save SUT model

The user can export the model as JSON file and use it as input for the test suite generator tool (Figure 58).

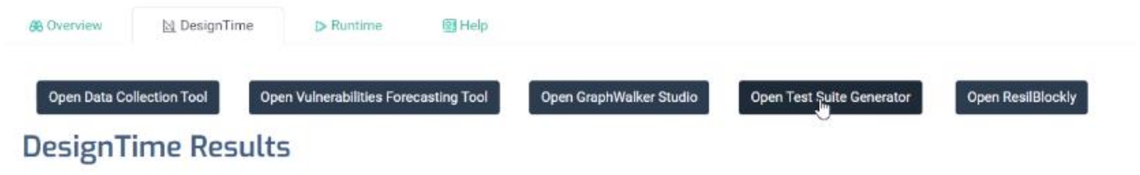
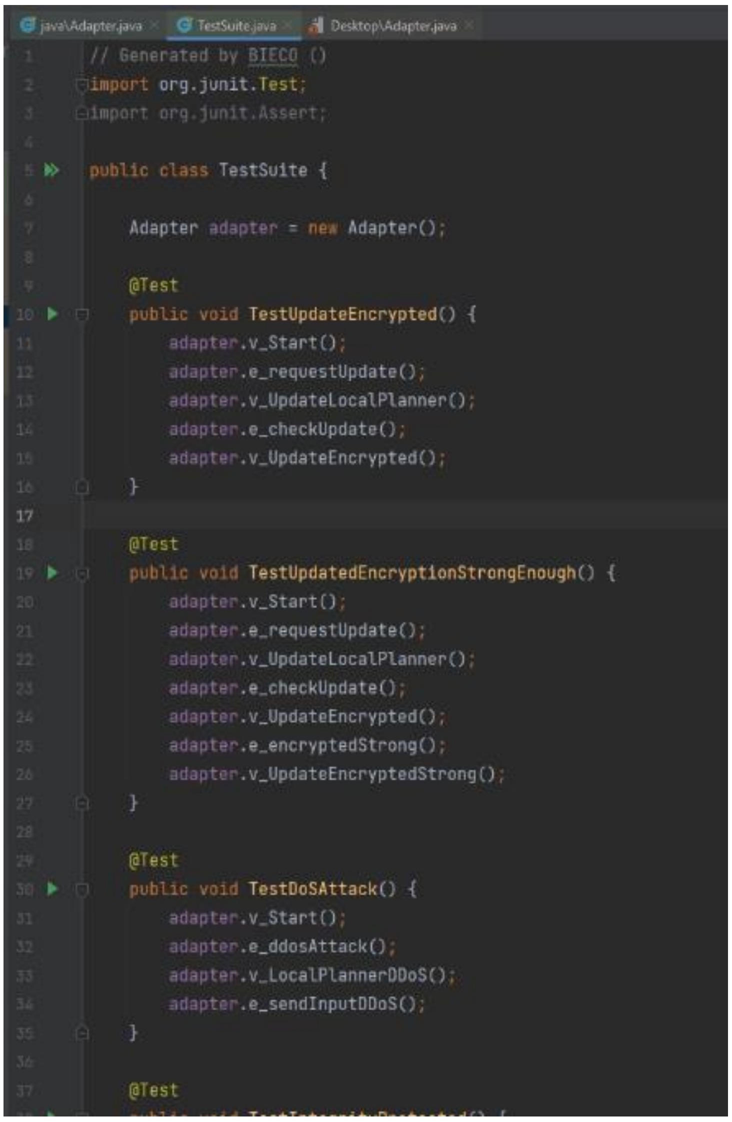

Overview
DesignTime
Runtime
Help
Open Data Collection Tool
Open Vulnerabilities Forecasting Tool
Open GraphWalker Studio
Open Test Suite Generator
Open ResilBlockly
DesignTime Results

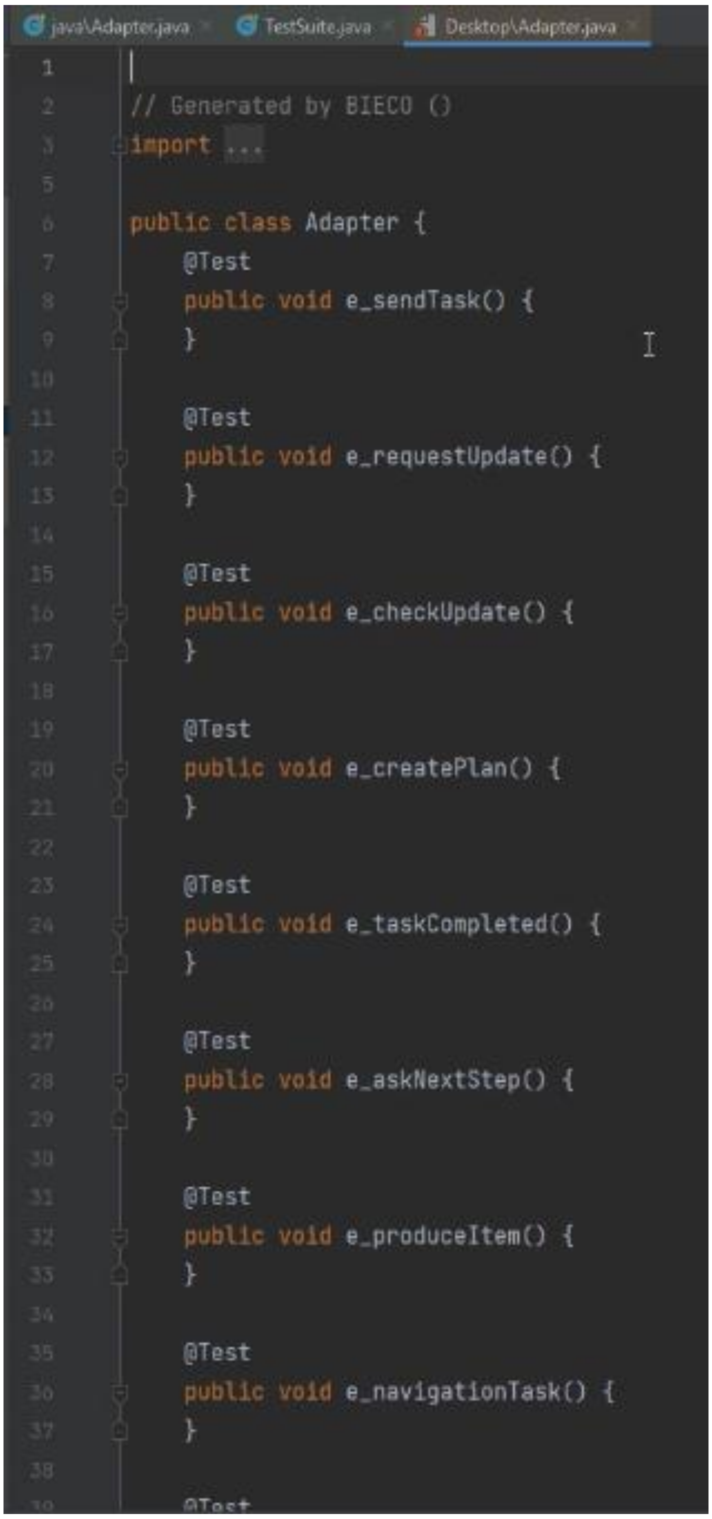
Figure 58 Test Suite Generator Tool from BIECO platform

This tool automatically generates two files. The first one (Figure 59) is the JUnit test suite ('TestSuite.java') with the 7 generated tests and the second one ('Adapter.java', Figure 60) is an interface called adapter to link the high-level test operations with the real system. The adapter must be implemented by the user to run the test suite over the real system and obtain the test report.



```java
// Generated by BIECO ()
import org.junit.Test;
import org.junit.Assert;

public class TestSuite {

    Adapter adapter = new Adapter();

    @Test
    public void TestUpdateEncrypted() {
        adapter.v_Start();
        adapter.e_requestUpdate();
        adapter.v_UpdateLocalPlanner();
        adapter.e_checkUpdate();
        adapter.v_UpdateEncrypted();
    }

    @Test
    public void TestUpdatedEncryptionStrongEnough() {
        adapter.v_Start();
        adapter.e_requestUpdate();
        adapter.v_UpdateLocalPlanner();
        adapter.e_checkUpdate();
        adapter.v_UpdateEncrypted();
        adapter.e_encryptedStrong();
        adapter.v_UpdateEncryptedStrong();
    }

    @Test
    public void TestDoSAttack() {
        adapter.v_Start();
        adapter.e_ddosAttack();
        adapter.v_LocalPlannerDDoS();
        adapter.e_sendInputDDoS();
    }

    @Test
```

Figure 59 Generated 'Adapter.java' and 'TestSuite.java' classes (1)



```java

// Generated by BIECO ()
import ...

public class Adapter {
    @Test
    public void e_sendTask() {
    }

    @Test
    public void e_requestUpdate() {
    }

    @Test
    public void e_checkUpdate() {
    }

    @Test
    public void e_createPlan() {
    }

    @Test
    public void e_taskCompleted() {
    }

    @Test
    public void e_askNextStep() {
    }

    @Test
    public void e_produceItem() {
    }

    @Test
    public void e_navigationTask() {
    }

    @Test
```

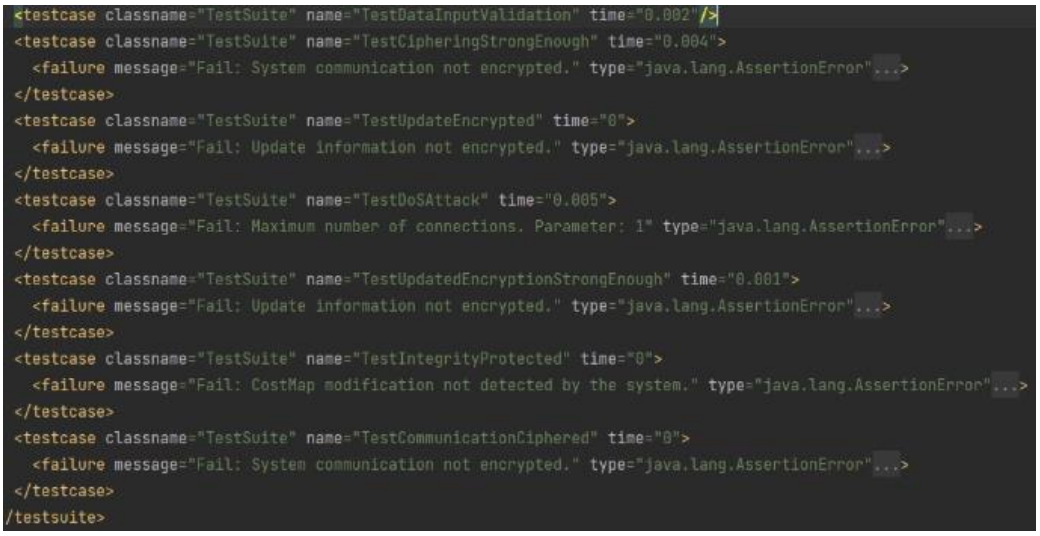
Figure 60 Generated 'Adapter.java' and 'TestSuite.java' classes (2)

Both classes must be integrated into a Maven project and the 'Adapter.java' must be implemented with the required functionality to achieve the execution of the tests. Also the rest of the SUT classes necessary for the execution must be imported into the project.

Once implemented, the execution is done by executing the maven command: *mvn test*.

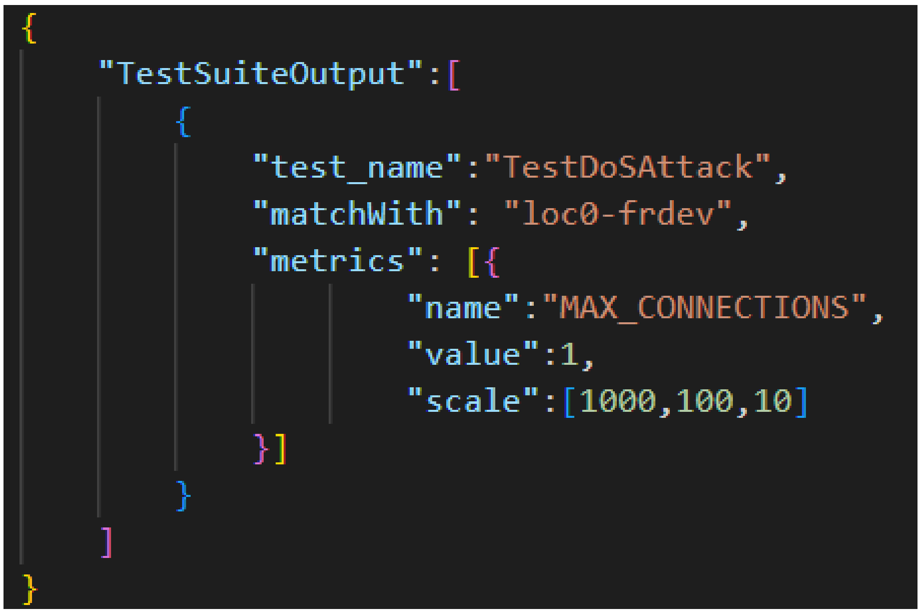
4.4. Risk estimation and evaluation

Next figure shows the test report generated by Graphwalker. In this case, 7 tests fail, 1 test passes (Figure 61) and the non-binary test obtains a metric of 1 for the number of simultaneous connections (Figure 62).



```
<testcase classname="TestSuite" name="TestDataInputValidation" time="0.002"/>
<testcase classname="TestSuite" name="TestCipheringStrongEnough" time="0.004">
  <failure message="Fail: System communication not encrypted." type="java.lang.AssertionError"...>
</testcase>
<testcase classname="TestSuite" name="TestUpdateEncrypted" time="0">
  <failure message="Fail: Update information not encrypted." type="java.lang.AssertionError"...>
</testcase>
<testcase classname="TestSuite" name="TestDoSAttack" time="0.005">
  <failure message="Fail: Maximum number of connections. Parameter: 1" type="java.lang.AssertionError"...>
</testcase>
<testcase classname="TestSuite" name="TestUpdatedEncryptionStrongEnough" time="0.001">
  <failure message="Fail: Update information not encrypted." type="java.lang.AssertionError"...>
</testcase>
<testcase classname="TestSuite" name="TestIntegrityProtected" time="0">
  <failure message="Fail: CostMap modification not detected by the system." type="java.lang.AssertionError"...>
</testcase>
<testcase classname="TestSuite" name="TestCommunicationCiphered" time="0">
  <failure message="Fail: System communication not encrypted." type="java.lang.AssertionError"...>
</testcase>
/testsuite>
```

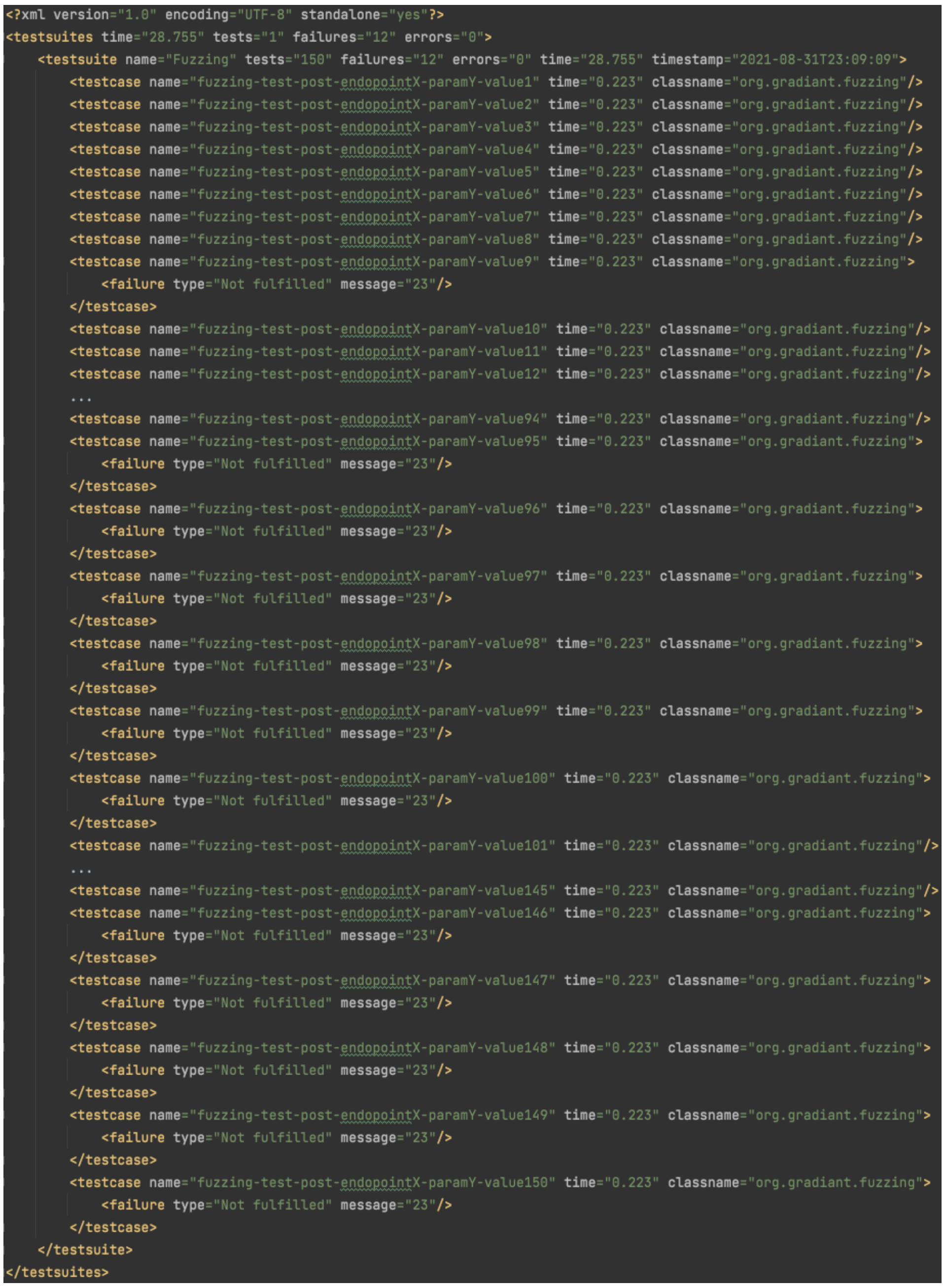
Figure 61 Test report output



```
{
    "TestSuiteOutput":[
        {
            "test_name":"TestDoSAttack",
            "matchWith": "loc0-frdev",
            "metrics": [{
                    "name":"MAX_CONNECTIONS",
                    "value":1,
                    "scale":[1000,100,10]
            }]
        }
    ]
}
```

Figure 62 Non-binary test metrics obtained

Figure 63 shows the internal report generated by the Fuzzing tool for Local Planner API in XML format. In this case, the tool executed 150 tests where none of them had an error and 12 failed. The tests made by the Fuzzing Tool always provides a binary output, and contains the tested claim.



```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<testsuites time="28.755" tests="1" failures="12" errors="0">
    <testsuite name="Fuzzing" tests="150" failures="12" errors="0" time="28.755" timestamp="2021-08-31T23:09:09">
        <testcase name="fuzzing-test-post-endopointX-paramY-value1" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value2" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value3" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value4" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value5" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value6" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value7" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value8" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value9" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value10" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value11" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value12" time="0.223" classname="org.gradiant.fuzzing"/>
        ...
        <testcase name="fuzzing-test-post-endopointX-paramY-value94" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value95" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value96" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value97" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value98" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value99" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value100" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value101" time="0.223" classname="org.gradiant.fuzzing"/>
        ...
        <testcase name="fuzzing-test-post-endopointX-paramY-value145" time="0.223" classname="org.gradiant.fuzzing"/>
        <testcase name="fuzzing-test-post-endopointX-paramY-value146" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value147" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value148" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value149" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
        <testcase name="fuzzing-test-post-endopointX-paramY-value150" time="0.223" classname="org.gradiant.fuzzing">
            <failure type="Not fulfilled" message="23"/>
        </testcase>
    </testsuite>
</testsuites>
```

Figure 63 Fuzzing tool report for LocalPlanner

4.5 Evaluation and labelling

The test report generated from the Graphwalker and fuzzing tool are then used as input in the ecurity scorer tool, which based on the methodology and on the tolerance profile selected, calculates for each one of the 6 security properties, the security level.

The output of the Security Scorer tool is visualised in the BIECO GUI by generating the security label associated to the security levels of the 6 security properties. Figure 64 shows the label obtained for UC4 in the BIECO GUI.

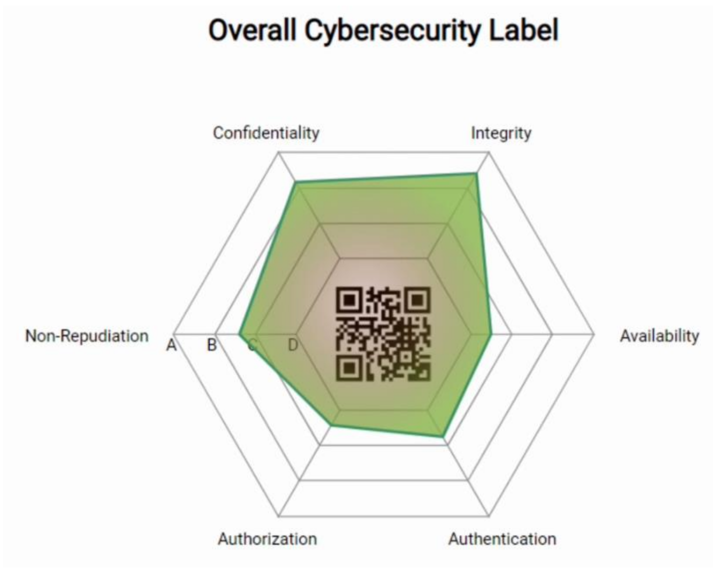
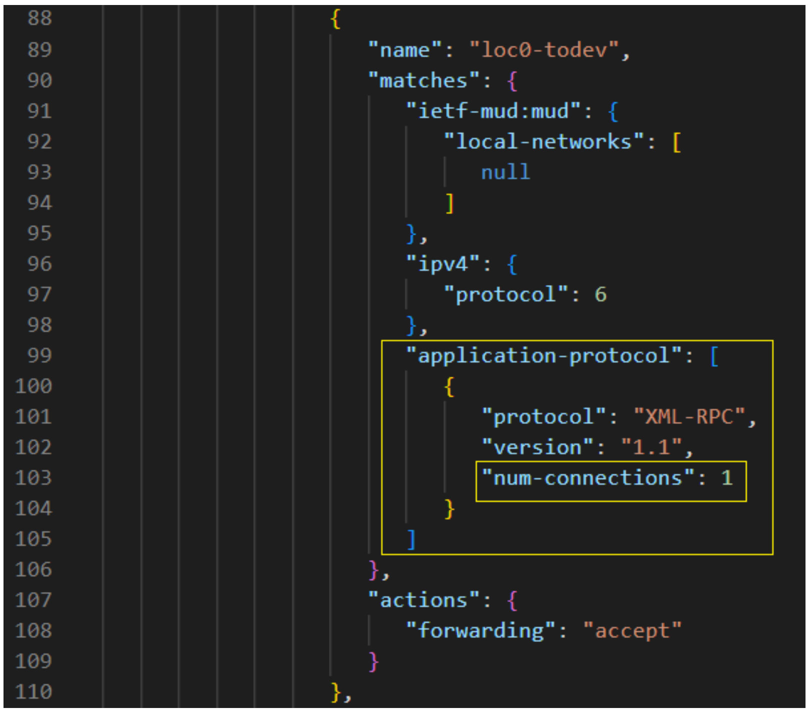


Figure 64 Security label of UC4

4.7. Treatment: Updating the extended MUD file

Based on the tests results from Graphwalker (described Section 3.4.1) and the updating process of the MUD Updater tool (Section 3.7) the information contained in the MUD file of our SUT is completed and updated with de derived metrics and values from the tests.

Taking into account the metrics file for this use case after the execution of the test, shown in Figure 62, the number of maximum simultaneous connections allowed is updated for the new value "1". The following figure illustrates:



```
{
  "name": "loc0-todev",
  "matches": {
    "ietf-mud:mud": {
      "local-networks": [
        null
      ]
    },
    "ipv4": {
      "protocol": 6
    },
    "application-protocol": [
      {
        "protocol": "XML-RPC",
        "version": "1.1",
        "num-connections": 1
      }
    ]
  },
  "actions": {
    "forwarding": "accept"
  }
},
```

Figure 65 MUD file of UC4 component localplanner is updated

5. Certificate Composition

Although until recently the certification focused on a specific type of product or process, the growing complexity of scenarios such as vehicles, IoT, 5G, etc. makes it necessary to search for more intelligent solutions. In a scenario such as the supply chain, a system can be made up of components that have been manufactured by different entities specialized in a specific type of product. In this scenario, a global system certification can only occur by assembling the certified components, commonly called composition.

In this case, each certified component provides its own evaluation results, creating a base that facilitates the reuse of this data for global certification. The objective is to reuse as much as possible the evidence that comes from another certification process, speeding up and reducing the complexity of the global certification of the system. This information would allow an evaluator to assess the security of the product in the context of a security evaluation methodology that defines what to assess, as well as the steps and techniques that should be applied during the process.

In this section, we analyze different scenarios that may arise in the context of the supply chain, and how the proposed methodology could be adapted to deal with them. For this, we take into account the guidelines for product certification composition of the CSA[11] and ECSO (European Cyber Security Organisation)[12].

5.1. Product Composition

A product can be a single component, or a system composed by different components. Even if the methodology seems designed to evaluate a **multicomponent** system, if the product is a **single component**, the certification process is the usual one and the evaluation methodology can be adapted by considering the SUT as a single component system during the risk identification phase (system description).

However, when the SUT is a complex system, the overall security depends on the security of its components and the security of their **interactions**. In this case, it is not enough just to reuse the information that the components could provide from previous evaluations, but we also need an analysis of the interactions between the component and the system. Additionally, integration testing and retesting of critical parts of the component may be necessary. The instantiation of the proposed methodology contemplates support tools and methods for the identification and measurement of dependencies between components like the one developed in WP3 (T3.4) to measure the **degree of dependency**, and the identification of cascade effects and propagation between possible attacks using the **attack paths** functionality of WP6 Resilblockly tool. The information obtained from this analysis can be used to identify additional tests necessary to perform a successful composition.

Although composition can help to reduce costs and time in the certification process by reusing evidence, it is not always an easy task. It is therefore necessary to identify what can be reused and what cannot. According to ECSO indications, one of the parameters that most influences this decision is the level of assurance (EAL), since a component should be evaluated at least at the same EAL as the system in order to reuse the evidence. If this is not possible, the previously evaluated component should be reevaluated as part of the SUT, with **white box** testing techniques, if the code is available, or **black box** techniques if the third party does not allow access to the code.

[11] https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX%3A32019R0881
[12] https://ecs-org.eu/documents/uploads/product-composition-document-november-2020.pdf

Furthermore, the composition process will be influenced by the relationship between the component and the system. If the component is an independent product, its functionality can be easily separated from the system and therefore the available evidence could be almost completely reused, requiring few **integration tests**. However, when the component is highly integrated into the system (e.g., it implements security functions of a larger product) the separation of component-system functions is more complex. In this scenario, the evidence that could be reused is reduced and it may be necessary to retest some aspects to guarantee that the security claims are still maintained. In any case, the **claims** that the SUT must fulfill **should drive the testing process**.

5.2. Scheme composition

If any of the system components has been **previously evaluated following the BIECO methodology**, the evidence can be easily reused, since the risk estimation phase combines the individual risks of each system component.

In this case, the risk associated with that component can be obtained from the previous evaluation and combined in the risk estimation phase together with the risks of the other components of the system to obtain the final security level, as shown in Figure 66. It is important to mention that, although this value can be reused, the combination of individual risks is modulated by the sensitivity factor, so a detailed analysis of the role of said component within the system and its dependencies with other components is necessary to the final calculation of the security level. In addition, as mentioned above, the integration of the component in the system can lead to additional claims and integration tests that allow verifying the joint security of the system. In this case, the results of the new tests should be combined with the previously certified security of the component.

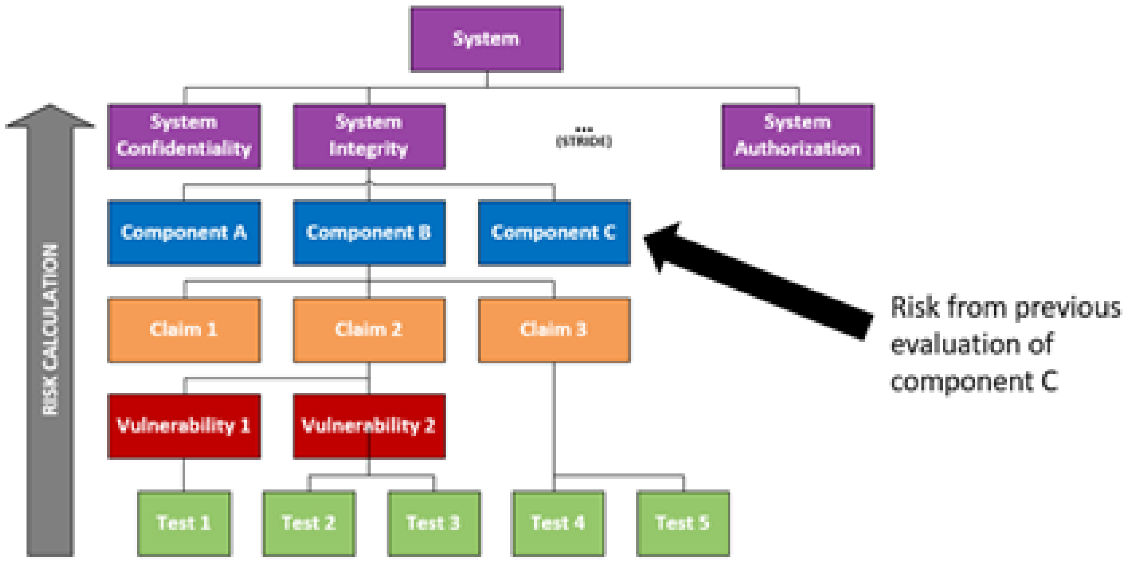


Figure 66 Composition within BIECO methodology

If any of the components of the system has been **evaluated under another evaluation scheme different from the one followed by BIECO**, the reuse of evidence requires a more fine-grained analysis. To allow the reuse, it is necessary to know which claims have been evaluated in the component, mapping the claims already evaluated with the claims required for the SUT to be certified.

Those claims that have been previously evaluated can be combined in the risk estimation phase (component risk), obtaining the probability of their previous result (0: the component met the claim, 1: the component did not meet the claim) and the impact through manual analysis in the 4 dimensions considered in the methodology. The claims that have not been evaluated on the component and affect the system must be evaluated using black box techniques to ensure compliance with said claim. If this is not possible, the composition could not be carried out. Finally, as mentioned before, the integration of the component in the system can lead to claims and additional integration tests that allow verifying the joint security of the system.

5.3 Supply chain cybersecurity interface agreement

Revealing data from a component's prior security assessment to use in certification composition is not a trivial process. It is necessary that both parties (the owner of the already certified component and the security evaluator of the SUT to be certified) agree on the type and amount of information needed to facilitate the composition and confidentiality of said information.

Such guidance for establishing agreements can be found in the automotive industry’s cybersecurity standard ISO 21434. In clause 7 of the standard, the subject of distributing cybersecurity activities (including e.g., assessment) across supply chain stakeholders is discussed. Specifically, as seen in Figure 67, the standard explains how an Original Equipment Manufacturer (OEM) can distribute requirements, responsibilities and overall activities with its direct and indirect suppliers (aka Tier-N suppliers). In the automotive domain, and other domains which involve similar types of stakeholders e.g., railway, OEMs are typically responsible with integrating systems and components provided by direct suppliers (aka ‘Tier-1’) and indirect suppliers (e.g., Tier-2 suppliers supply Tier-1, and so on). Distribution of cybersecurity activities is achieved bilateral cybersecurity interface agreements, which should address, among other things:

- Cybersecurity-related requirements of the technical elements being supplied
- Relationship and responsibilities across the customer-supplier
- Applicable lifecycle phases (which may extend into the post-deployment phase)

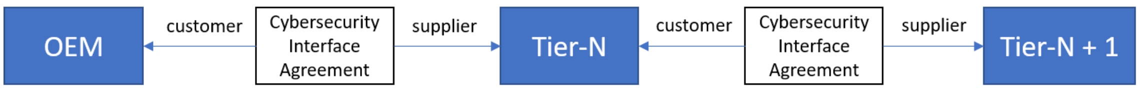

OEM
customer
Cybersecurity Interface Agreement
supplier
Tier-N
customer
Cybersecurity Interface Agreement
supplier
Tier-N + 1

Figure 67 - OEM-Tier-N Cybersecurity Interface Agreement Abstract Example (reproduced from ISO 21434:2021, p.20)

Through these bilateral agreements, the timing and responsibility of conducting relevant cybersecurity activities can be agreed-upon beforehand. Provisions in such agreements can include, for example, that the cybersecurity validation of the integrated system should be conducted by the OEM, whereas the supplied component’s cybersecurity assessment should be assigned to a third party. Distributing activities in such ways allows proprietary information to remain protected, while still providing cybersecurity assurance to an acceptable degree.

Moreover, as mentioned previously, the distribution of activities can also extend into the post-deployment phase of the system. For instance, the parties could agree that the supplier should be responsible for monitoring cybersecurity vulnerability information channels for new vulnerabilities e.g., by periodically searching the NIST NVD CVE dataset [34]. When a relevant vulnerability for the supplied component is brought to the attention of the supplier, they could then be responsible for assessing its risk themselves, and deciding on its treatment, which includes (per ISO 21434:2021:15.9.2):

- Avoiding the risk, e.g., by halting system operation;
- Reducing the risk, e.g., by restricting or degrading the system's performance;
- Sharing the risk, e.g., based on contracts and/or insurance;
- Retaining the risk.

Naturally, in cases where the decision impacts the customer (e.g., OEM), it would be expected that the agreement stipulates that the customer should be made aware of the decision. In cases of sharing or retaining the risk, the standard explicitly states that corresponding cybersecurity statement about a risk should be established (or updated) to reflect the rationale leading to the respective risk treatment decision.

The BIECO methods and corresponding tools support such distribution of activities across the supply chain, as indicated in the previous subsections of Section 5.

6. Conclusions

This deliverable has culminated the work developed during WP7, which started by the definition of a basic set of security and privacy claims in T7.1, the design of a flexible security evaluation methodology in T7.2 and a specific proposal instantiation of the methodology in T7.3.

Each of the methodology phases, including establishing the context, risk identification, risk estimation, security testing, risk evaluation, treatment, and labelling were described, identifying which tools from the BIECO platform could be used and how to support the user in each of them. Each phase has its own methods and tools, which were also described in this document.

The goal was to bring the methodology defined theoretically in deliverable D7.2 to real use, integrating tools developed within BIECO. Therefore, each phase was instantiated and in Chapter 4 a proof-of-concept application was provided. A more complete validation of the methodology will be provided within WP8, based on the claims selected in the Annex I. Finally, Chapter 5 included a discussion on certificate composition, a scenario very common y the supply chain scenario, and how the methodology could be adapted to fit in.

7. Artifacts

Table 3 shows the artifacts produced in this deliverable.

Table 3 Artifactts produced in T7.3

Name	Description
SecurityScorer	A tool responsible for parsing the system description and risk estimation tools' outputs and evaluating the numerical risk value and system properties' labels.
Graphwalker	Improvement of the Graphwalker open-source tool to integrate it in the methodology, allowing automated test suite generation and non-binary values.
GROOT	GROOT stands for *GdpR-based cOmbinatOrial Testing*. It is a general combinatorial testing approach, for validating systems (including access control) managing GDPR's concepts (e.g., Data Subject, Personal Data or Controller).

8. Annex I: Use Case Selected Claims

8.1 UC1: ICT GW

ID	Description	STRIDE category
C0	Update software files should be encrypted and be transmitted using encryption	Confidentiality
C1	Update software files should be integrity protected	Integrity
C2	Update software files should be encrypted using strong keys and algorithms	Confidentiality
C3	Update software files should be authenticated	Authentication
C4	The update mechanism shall prevent downgrade	Availability
C5	The exchanged messages in the communication should be integrity protected	Integrity
C6	Automatically generated passwords should be unique	Authentication, Confidentiality
C7	Passwords should avoid common patterns	Authentication, Confidentiality
C8	Passwords are not obviously linked to public information	Authentication, Confidentiality
C9	Passwords should be strong in terms of complexity	Authentication, Confidentiality
C11	Sensitive security parameters exchanged during the communication for the establishment of a secure association should be integrity protected	Integrity
C12	Stored sensitive security parameters should be integrity protected	Integrity
C13	Stored critical security parameters should be ciphered	Confidentiality
C14	Ciphered communications should use strong algorithms	Confidentiality
C15	Access to device functionality via a network interface in the initialized state should only be possible after authentication on that interface.	Authentication
C16	The system should have a mechanism available which makes brute-force attacks on authorization mechanisms via network interfaces impracticable.	Authentication
C17	Sensitive security parameters should be encrypted in transit, with such encryption appropriate	Confidentiality
C18	All unused network interfaces shall be disabled.	Authorization
C19	The confidentiality of personal data transiting between a device and a service, especially associated services, should be protected, with best practice cryptography	Confidentiality
C20	Authentication mechanisms must use strong passwords	Authentication
C21	Integrity mechanisms must be strong	Integrity
C22	Resistance to DoS attacks	Availability
C23	Data input validation	Availability
C24	Data communications should be ciphered	Confidentiality
C28	The source code must not contain SQL injection vulnerabilities	Integrity, Availability, Confidentiality

C29	The source code must not contain command injection vulnerabilities	Integrity, Availability, Confidentiality
C30	The source code must not contain code injection vulnerabilities	Integrity, Availability, Confidentiality
C31	The source code must not contain path traversal vulnerabilities	Integrity, Availability, Confidentiality
C32	The source code must not use components with known vulnerabilities	Integrity, Availability, Confidentiality
C36	Warning must be issued in case of potentially reduced functionality	Integrity, Availability
C37	Warning must be followed by triggering fail-over behaviour	Integrity, Availability
C38	Safety Risk Management has been applied	Integrity, Availability
C39	Automatic updates should not change the network protocol interfaces in any way that is incompatible with previous versions	Integrity, Availability
C42	Connections to remote services, interfaces, and end-points should be cryptographically authenticated	Authentication
C43	The software should not use unsafe libraries that contain vulnerabilities	All
C44	Device should remain operating and locally functional in the case of a lost network connection	Availability
C45	Protocols and libraries used by the system are updated	All
C46	Authentication protocols should be secure, using recommended algorithms.	Authentication
C47	Authenticated sessions should expire, and a new re-authentication required.	Authentication
C49	Authentication algorithms should avoid channel side attack	Authentication
C50	System should work in case of power outage	Availability
C53	The system should allow data subject to withdraw its given consent	Authorization, Confidentiality
C54	The system shall implement mechanisms of protection from malicious code manipulation	Integrity, Availability
C55	The system shall update protection mechanisms whenever new releases are available	Integrity, Availability
C56	The system shall prevent anyone from circumventing malicious code protection mechanisms.	Integrity, Availability
C57	The system shall enforce assigned authorizations for controlling the flow of information within the system and from interconnected systems	Authorization, Confidentiality
C58	The system shall enforce a limit of consecutive invalid login attempts during a time period.	Authentication
C59	The system shall notify, upon successful logon, of the date and time of the last logon and the number of unsuccessful logon attempts since the last successful logon.	Authentication

C60	The system shall execute a fail-safe procedure upon the loss of communications with other systems.	Integrity, Availability
C61	The system shall uniquely identify and authenticate users.	Authentication, Integrity, Non-repudiation
C62	The system shall uniquely identify and authenticate a defined list of devices before establishing a connection	Authentication, Non-repudiation
C63	The system shall isolate security functions from non-security functions.	Integrity
C64	The system shall separate user functionalities from management functionalities.	Integrity
C65	The system shall monitor events to detect attacks, unauthorized activities or conditions, and non-malicious errors	Authorization, Integrity, Non-repudiation
C66	The system shall lock the session after a configurable time period of inactivity.	Authentication, Non-repudiation
C67	The system shall set outputs to a predetermined state if normal operation cannot be maintained as a result of an attack.	Availability
C68	The system shall prevent messages from being received from external users or systems.	Integrity, Confidentiality
C69	The system shall operate in a degraded mode during a DoS event.	Integrity, Availability
C70	The system shall limit the use of resources by security functions to prevent resource exhaustion.	Integrity, Availability
C71	The system shall terminate a remote session at the end of the session or after a period of inactivity.	Authentication, Authorization
C74	The system should ensure that only authorised users may gain access to the information under the circumstances specified in the access control policy	Authorization
C75	The system shall monitor events to detect attacks, unauthorized activities or conditions, and non-malicious errors.	Integrity

8.2 UC2: AI INVESTMENT

ID	Description	STRIDE category
C11	Sensitive security parameters exchanged during the communication for the establishment of a secure association should be integrity protected	Integrity
C12	Stored sensitive security parameters should be integrity protected	Integrity
C13	Stored critical security parameters should be ciphered	Confidentiality
C14	Ciphered communications should use strong algorithms	Confidentiality
C17	Sensitive security parameters should be encrypted in transit, with such encryption appropriate	Confidentiality
C18	All unused network interfaces shall be disabled.	Authorization
C20	Authentication mechanisms must use strong passwords	Authentication
C21	Integrity mechanisms must be strong	Integrity
C23	Data input validation	Availability
C24	Data communications should be ciphered	Confidentiality
C28	The source code must not contain SQL injection vulnerabilities	Integrity, Availability, Confidentiality
C29	The source code must not contain command injection vulnerabilities	Integrity, Availability, Confidentiality
C30	The source code must not contain code injection vulnerabilities	Integrity, Availability, Confidentiality
C31	The source code must not contain path traversal vulnerabilities	Integrity, Availability, Confidentiality
C32	The source code must not use components with known vulnerabilities	Integrity, Availability, Confidentiality
C42	Connections to remote services, interfaces, and end-points should be cryptographically authenticated	Authentication
C43	The software should not use unsafe libraries that contain vulnerabilities	All
C45	Protocols and libraries used by the system are updated	All
C46	Authentication protocols should be secure, using recommended algorithms.	Authentication
C47	Authenticated sessions should expire, and a new re-authentication required.	Authentication
C48	Random bit generators should be strong enough	All
C49	Authentication algorithms should avoid channel side attack	Authentication
C54	The system shall implement mechanisms of protection from malicious code manipulation	Integrity, Availability
C56	The system shall prevent anyone from circumventing malicious code protection mechanisms.	Integrity, Availability

C58	The system shall enforce a limit of consecutive invalid login attempts during a time period.	Authentication
C59	The system shall notify, upon successful logon, of the date and time of the last logon and the number of unsuccessful logons attempts since the last successful logon.	Authentication
C60	The system shall execute a fail-safe procedure upon the loss of communications with other systems.	Integrity, Availability
C61	The system shall uniquely identify and authenticate users.	Authentication, Integrity, Non-repudiation
C65	The system shall monitor events to detect attacks, unauthorized activities or conditions, and non-malicious errors	Authorization, Integrity, Non-repudiation
C68	The system shall prevent messages from being received from external users or systems.	Integrity, Confidentiality
C72	Logs should be protected against removal	Non-repudiation
C74	The system should ensure that only authorised users may gain access to the information under the circumstances specified in the access control policy	Authorization
C75	The system shall monitor events to detect attacks, unauthorized activities or conditions, and non-malicious errors.	Integrity

8.3 UC3: MICROFACTORY – SOFTWARE UPDATES

ID	Description	STRIDE category
C0	Update software files should be encrypted and be transmitted using encryption	Confidentiality
C1	Update software files should be integrity protected	Integrity
C2	Update software files should be encrypted using strong keys and algorithms	Confidentiality
C3	Update software files should be authenticated	Authentication
C4	The update mechanism shall prevent downgrade	Availability
C5	The exchanged messages in the communication should be integrity protected	Integrity
C6	Automatically generated passwords should be unique	Authentication, Confidentiality
C7	Passwords should avoid common patterns	Authentication, Confidentiality
C9	Passwords should be strong in terms of complexity	Authentication, Confidentiality
C11	Sensitive security parameters exchanged during the communication for the establishment of a secure association should be integrity protected	Integrity
C12	Stored sensitive security parameters should be integrity protected	Integrity
C13	Stored critical security parameters should be ciphered	Confidentiality
C14	Ciphered communications should use strong algorithms	Confidentiality
C15	Access to device functionality via a network interface in the initialized state should only be possible after authentication on that interface.	Authentication
C16	The system should have a mechanism available which makes brute-force attacks on authorization mechanisms via network interfaces impracticable.	Authentication
C17	Sensitive security parameters should be encrypted in transit, with such encryption appropriate	Confidentiality
C18	All unused network interfaces shall be disabled.	Authorization
C20	Authentication mechanisms must use strong passwords	Authentication
C21	Integrity mechanisms must be strong	Integrity
C22	Resistance to DoS attacks	Availability
C23	Data input validation	Availability
C24	Data communications should be ciphered	Confidentiality
C29	The source code must not contain command injection vulnerabilities	Integrity, Availability, Confidentiality
C30	The source code must not contain code injection vulnerabilities	Integrity, Availability, Confidentiality
C31	The source code must not contain path traversal vulnerabilities	Integrity, Availability, Confidentiality

C32	The source code must not use components with known vulnerabilities	Integrity, Availability, Confidentiality
C36	Warning must be issued in case of potentially reduced functionality	Integrity, Availability
C38	Safety Risk Management has been applied	Integrity, Availability
C39	Automatic updates should not change the network protocol interfaces in any way that is incompatible with previous versions	Integrity, Availability
C42	Connections to remote services, interfaces, and end-points should be cryptographically authenticated	Authentication
C43	The software should not use unsafe libraries that contain vulnerabilities	All
C44	Device should remain operating and locally functional in the case of a lost network connection	Availability
C45	Protocols and libraries used by the system are updated	All
C46	Authentication protocols should be secure, using recommended algorithms.	Authentication
C47	Authenticated sessions should expire, and a new re-authentication required.	Authentication
C48	Random bit generators should be strong enough	All
C49	Authentication algorithms should avoid channel side attack	Authentication
C54	The system shall implement mechanisms of protection from malicious code manipulation	Integrity, Availability
C55	The system shall update protection mechanisms whenever new releases are available	Integrity, Availability
C56	The system shall prevent anyone from circumventing malicious code protection mechanisms.	Integrity, Availability
C57	The system shall enforce assigned authorizations for controlling the flow of information within the system and from interconnected systems	Authorization, Confidentiality
C58	The system shall enforce a limit of consecutive invalid login attempts during a time period.	Authentication
C59	The system shall notify, upon successful logon, of the date and time of the last logon and the number of unsuccessful logons attempts since the last successful logon.	Authentication
C61	The system shall uniquely identify and authenticate users.	Authentication, Integrity, Non-repudiation
C62	The system shall uniquely identify and authenticate a defined list of devices before establishing a connection	Authentication, Non-repudiation
C63	The system shall isolate security functions from non-security functions.	Integrity
C65	The system shall monitor events to detect attacks, unauthorized activities or conditions, and non-malicious errors	Authorization, Integrity, Non-repudiation
C66	The system shall lock the session after a configurable time period of inactivity.	Authentication, Non-repudiation

C67	The system shall set outputs to a predetermined state if normal operation cannot be maintained as a result of an attack.	Availability
C68	The system shall prevent messages from being received from external users or systems.	Integrity, Confidentiality
C69	The system shall operate in a degraded mode during a DoS event.	Integrity, Availability
C70	The system shall limit the use of resources by security functions to prevent resource exhaustion.	Integrity, Availability
C71	The system shall terminate a remote session at the end of the session or after a period of inactivity.	Authentication, Authorization
C72	Logs should be protected against removal	Non-repudiation
C74	The system should ensure that only authorised users may gain access to the information under the circumstances specified in the access control policy	Authorization

9. Annex II: SecurityScorer – Technical Annex

SecurityScorer is a tool developed for the BIECO methodology. It is responsible for the entire risk evaluation phase. To be precise, the result of the SecurityScorer (so the result of the risk identification phase) should be a numerical value representing risk for each of the six STRIDE properties of the system under evaluation. Alternatively, the output can be understood as a certification of each of the system properties with labels inferred from the tolerance profile of the system.

To achieve this, SecurityScorer needs the **YAML file with the system description** and tolerance profiles in order to build an internal system scheme and labelling scheme. Moreover, it needs the **outputs from the testing tools** used in the previous phases of the BIECO methodology: risk identification and estimation, i.e.: Graphwalker, Fuzzing Tool, and Groot. SecurityScorer is responsible for parsing the output of each of these tools and, combining all the results with the whole system scheme, evaluating the results. SecurityScorer implements a separate module for each of the tools, but the modules and the implementation of the calculation of the risk is out of scope of this description. A general analytical derivation was presented in Section 3.

SecurityScorer is a standalone tool, therefore it can be independently as a Python application with FastAPI interface, although in practice, its main use case is to be called by other actors in the BIECO methodology. To be able to evaluate the results, it is using local YAML and output files, which are provided through API endpoints.

9.1.1. Installation Guide

Prerequisites:

- pipenv
- tox

Steps to install and run:

1. Clone the repository:

```
git clone https://github.com/7bulls/security-scorer-public.git
```

2. Install:

```
pipenv install --keep-outdated
```

3. Run as a uvicorn server:

```
pipenv run uvicorn security_scorer:app
```

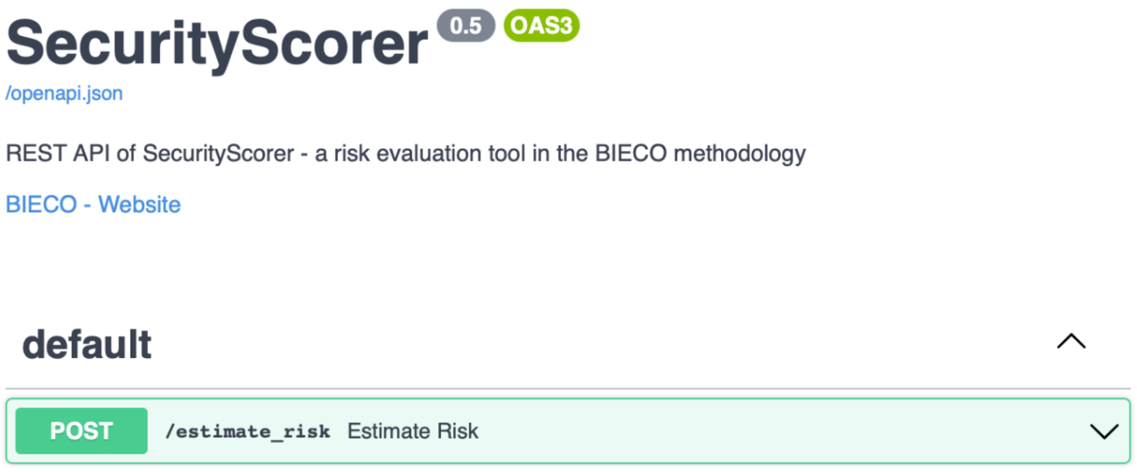
4. (optional) To run tests, run in the main project directory:

```
tox
```

9.1.2. Usage Guide

After installing and running the uvicorn sever using the default settings, the API is accessible from:

```
http://127.0.0.1:8000/docs#/
```


SecurityScorer 0.5 OAS3
/openapi.json

REST API of SecurityScorer - a risk evaluation tool in the BIECO methodology

BIECO - Website

default

POST /estimate_risk Estimate Risk

Figure 68 SecurityScorer REST API graphical interface

Figure 68 presents the graphical version of the documentation, where the SecurityScorer endpoint can be easily executed.

Alternatively, user can send the POST request manually to /estimate_risk, with request body as JSON:

```
{
 "metadata": {
  "blob": "..."
 },
 "tool_outputs": {
  "graphwalker": "...",
  "fuzzing": "...",
  "groot": "..."
 }
}
```

From the set of *graphwalker*, *fuzzing*, *groot* tools, user should use only these tools that were used to test his system. Ellipsis denote a base64-encoded contents of the system description YAML file (*metadata*), or tools outputs.

The default result is a list of values for each system property and a list of labels:

```
{
 "scores": {
  "confidentiality": 3.0,
  "integrity": 1.5,
  "availability": 2.7,
  "authorization": 1.1,
  "authentication": 4.0,
  "non_repudiation": 2.0
 },
 "labels": {
  "confidentiality": B,
  "integrity": A,
  "availability": B,
  "authorization": A,
  "authentication": C,
  "non_repudiation": B
 }
}
```

The labels are evaluated using the tolerance profiles, passed in the YAML file with the system description. A more detailed description of the labelling was presented in Section 3.6 of this document.

After returning these results, SecurityScorer's API waits for new requests. Because the tool is stateless, all the data from previous execution is not persisted.

10. References

[1] MELL, Peter, et al. *CAESARS Framework Extension: An Enterprise Continuous Monitoring Technical Reference Architecture*. National Institute of Standards and Technology, 2012.

[2] *DIGITALEUROPE's views on cybersecurity certification and labelling schemes*. DIGITALEUROPE, 02-Mar-2017. [Online] Available: https://www.digitaleurope.org/resources/digitaleuropes-views-on-cybersecurity-certification-and-labelling-schemes/

[3] *WG1: Standardisation, Certification and Supply Chain Management*. ECSO. [Online] Available: https://ecs-org.eu/working-groups/wg1-standardisation-certification-and-supply-chain-management

[4] *Methods for Testing & Specification; Risk-based Security Assessment and Testing Methodologies.* ETSI, Nov-2015 [Online] Available: https://www.etsi.org/deliver/etsi_eg/203200_203299/203251/01.01.01_50/eg_203251v010101m.pdf

[5] *Methods for Testing & Specification; Risk-based Security Assessment and Testing Methodologies.* ETSI, Nov-2015 [Online] Available: https://www.etsi.org/deliver/etsi_eg/203200_203299/203251/01.01.01_50/eg_203251v010101m.pdf

[6] ARMOUR. [Online] Available: https://www.sciencedirect.com/science/article/abs/pii/S0920548918301375

[7] *Common Criteria for Information Technology Security Evaluation. Part 1: Introduction and general model.* Common Criteria, 2017.

[8] *Arrangement on the Recognition of Common Criteria Certificates In the field of Information Technology Security*. Common Criteria, 2014. [Online] Available: https://www.commoncriteriaportal.org/files/operatingprocedures/cc-recarrange.pdf

[9] C. Zhou, S. Ramacciotti, *Common Criteria: Its Limitations and Advice on Improvement*, *ISSA J.*, 2011. [Online] Available: https://www.difesa.it/SMD_/Staff/Reparti/II/CeVa/Pubblicazioni/Estere/Documents/CommonCriteria_ISSA%20Journal_0411.pdf

[10] S. P. Kaluvuri, M. Bezzi, Y. Roudier, *A Quantitative Analysis of Common Criteria Certification Practice, in Trust, Privacy, and Security in Digital Business*, vol. 8647, Cham: Springer International Publishing, 2014, pp. 132-143.

[11] F. Keblawi, D. Sullivan, *Applying the common criteria in systems engineering*, *IEEE Secur. Priv. Mag.*, vol. 4, n.º 2, pp. 50-55, 2006, doi: 10.1109/MSP.2006.35.

[12] *CyberSecurity Certification: EUCC Candidate Scheme*. ENISA. [Online] Available: https://www.enisa.europa.eu/publications/cybersecurity-certification-eucc-candidate-scheme-v1-1.1

[13] *Cloud Service Scheme*. EUCS, 22-Dec-2020, [Online] Available: https://www.enisa.europa.eu/topics/publications/eucs-cloud-service-scheme

[14] *Securing EU's Vision on 5G: Cybersecurity Certification*. EUCS, 03-Feb-2021, [Online] Available: https://www.enisa.europa.eu/news/enisa-news/securing_eu_vision_on_5g_cybersecurity_certification

[15] *Certification de sécurité de premier niveau (CSPN)*, *ANSSI*, 2008. [Online] Available: https://www.ssi.gouv.fr/administration/produits-certifies/cspn/

[16] *Certification de securite de premier niveau*. CryptoExperts [Online] Available: https://www.cryptoexperts.com/services/cspn/

[17] G. Baldini, G. Giannopoulos, A. Lazari, *Annex 8: JRC Analysis and recommendations for a European certification and labelling framework for cybersecurity in Europe*, European Commission, 2017.

[18] *UL 2900 Standards Process*. Underwriters Laboratories [Online] Available: https://industries.ul.com/cybersecurity/ul-2900-standards-process

[19] *The Commercial Product Assurance (CPA) build standard*. CESG, 2014, [Online] Available: https://www.nccgroup.trust/uk/our-services/cyber-security/compliance-and-accreditations/cpa-and-cc/

[20] *Foundation Grade explained*. National Cybersecurity Center of United Kingdom, 2017, [Online] Available: https://www.ncsc.gov.uk/articles/foundation-grade-explained
[21] Jungmayr, S. (2002). Testability measurement and software dependencies. undefined. Retrieved from https://www.semanticscholar.org/paper/Testability-measurement-and-software-dependencies-Jungmayr/f3a04f41fb801bda8de726b284bdc3dae1c5ea50
[22] *The Commercial Product Assurance (CPA) build standard*. CESG, 2014. [Online] Available: https://www.nccgroup.trust/uk/our-services/cyber-security/compliance-and-accreditations/cpa-and-cc/
[23] *STATE OF THE ART SYLLABUS*. ECSO, Jun-2017, [Online] Available: http://www.ecs-org.eu/documents/uploads/state-of-the-art-syllabus-v1.pdf
[24] *IEC 62443-4-1:2018*. IEC Webstore, 15-Jan-2018, [Online] Available: https://webstore.iec.ch/publication/33615
[25] *Security of Industrial Automation and Control Systems*. International Society of Automation (ISA), Oct-2020, [Online] Available: https://www.isasecure.org/en-US/Documents/Articles-and-Technical-Papers/ISAGCA-Security-Lifecycles-whitepaper
[26] *CNSSI 4009 Committee on National Security Systems (CNSS) Glossary*, 2015. [Online] Available: https://www.serdp-estcp.org/Tools-and-Training/Installation-Energy-and-Water/Cybersecurity/Resources-Tools-and-Publications/Resources-and-Tools-Files/CNSSI-4009-Committee-on-National-Security-Systems-CNSS-Glossary
[27] *Common Weakness Scoring System (CWSS)*. MITRE, 2014. [Online] Available: https://cwe.mitre.org/cwss/cwss_v1.0.1.html
[28] J. R. C. Nurse, S. Creese, and D. D. Roure, *Security Risk Assessment inInternet of Things Systems*, IEEE Computer Society, IT Pro, 2017.
[29] Eclipse Modelling Framework. [Online] Available: https://www.eclipse.org/modeling/emf/
[30] Common Weakness Enumeration (CWE), [Online] Available: https://cwe.mitre.org/
[31] OWASP Top Ten, [Online] Available: https://owasp.org/www-project-top-ten/
[32] *Common Vulnerability Score System (CVSS) v3*. IRST, 2015, [Online] Available: https://www.first.org/cvss/v3.0/cvss-v30-specification_v1.9.pdf
[33] Common Vulnerabilities and Exposures (CVE), [Online] Available: https://cve.mitre.org/
[34] National Vulnerability Database (NVD), [Online] Available: https://nvd.nist.gov/
[35] Common Attack Pattern Enumeration (CAPEC), [Online] Available: https://capec.mitre.org/
[36] R. A. Caralli, J. F. Stevens, L. R. Young, y W. R. Wilson, *Introducing OCTAVE Allegro: Improving the Information Security Risk Assessment Process*, CERT, 2007
[37] C. J. Alberts, A. J. Dorofee, J. F. Stevens, y C. Woody, *OCTAVE-S Implementation Guide, Version 1*, 2005
[38] *DREAD scheme*. Microsoft, 2010, [Online] Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648644(v=pandp.10)#dread
[39] *Graphwalker, an open*-source model-based testing tool. [Online] Available: https://graphwalker.github.io/
[40] A. B. Garcia, R. F. Babiceanu, y R. Seker, *Trustworthiness requirements and models for aviation and aerospace systems*, 2018 Integrated Communications, Navigation, Surveillance Conference (ICNS), Herndon, VA, 2018, pp. 1-16, doi: 10.1109/ICNSURV.2018.8384911
[41] *Threat prioritisation: DREAD is dead, baby?*. NCCgroup, 2016, [Online] Available: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2016/march/threat-prioritisation-dread-is-dead-baby/
[42] *VerAfied Methodology*. VERACODE [Online] Available: https://www.veracode.com/verified
[43] *Veracode Detailed Report.* Nextcloud, 2016, [Online] Available: https://nextcloud.com/wp-content/themes/next/assets/files/veracode_report.pdf?x53054
[44] *The Cenzic HARM (Hailstorm Application Risk Metric) Score*. Cenzic, [Online] Available: https://owasp.org/www-pdf-archive//OWASP_Cloudy_with_a_chance_of_hack_Nov_2010.pdf
[45] *OWASP Application Security Verification Standard (ASVS) Project*. OWASP, [Online] Available: https://owasp.org/www-project-application-security-verification-standard/

[46] R. M. R. K, *Security risk assessment of Geospatial Weather Information System (GWIS) using integrated CVSS approach*, Int. J. Adv. Comput. Sci. Appl., vol. 1, n.o 3, 2010

[47] Daoudagh, S., Marchetti, E. (2022). GROOT: A GDPR-Based Combinatorial Testing Approach. In: Clark, D., Menendez, H., Cavalli, A.R. (eds) Testing Software and Systems. ICTSS 2021. Lecture Notes in Computer Science, vol 13045. Springer, Cham. https://doi.org/10.1007/978-3-031-04673-5_17

[48] A. R. Ruddle, H. Mira, and S. Information, *Security requirements for automotive on-board networks based on dark-side scenarios. E-safety vehicle intrusion protected applications*. EVITA project, Tech. Rep. February 2016, 2009, [Online] Available: https://www.researchgate.net/publication/46307752_Security_requirements_for_automotive_on-board_networks_based_on_dark-side_scenarios_Deliverable_D23_EVITA_E-safety_vehicle_intrusion_protected_applications

[49] L. Aljoscha and M. Islam, *HEAling Vulnerabilities to ENhance Software Security and Safety - Project Proposal HEAVENS*. 2016.

[50] *ETSI TS 102 165-1 Methods and protocols; Part 1: Method and pro forma for Threat, Vulnerability, Risk Analysis (TVRA)*. ETSI, 2017.

[51] Macher, G., Armengaud, E., Brenner, E., Kreiner, C.: *A review of threat analysis and risk assessment methods in the automotive context*. In International Conference on Computer Safety, Reliability, and Security. pp. 130–141. Springer (2016)

[52] Palin, R., Ward, D., Habli, I., Rivett, R.: *Iso 26262 safety cases: Compliance and assurance*. In 6th IET International Conference on System Safety 2011. pp. 1–6.IET (2011)

[53] *ISO: ISO/SAE FDIS 21434 Road vehicles*. Cybersecurity engineering, [Online] Available: https://www.iso.org/standard/70918.html

[54] *SESAMO: Security and Safety Modelling*. 2015, [Online] Available: http://www.sesamo-project.eu/

[55] *SESAMO: D4.2 Integrated Design and Evaluation Methodology*. 2014, [Online] Available: http://sesamo-project.eu/content/d42-integrated-design-and-evaluation-methodology

[56] Ramaiah, B.S.M.P.S., Gokhale, A.A.: *Fmea and fault tree based software safety analysis of a railroad crossing critical system*. Global Journal of Computer Science and Technology (2011)

[57] Schmittner, C., Gruber, T., Puschner, P., Schoitsch, E.: *Security application of failure mode and effect analysis (fmea)*. In International Conference on Computer Safety, Reliability, and Security. pp. 310–325. Springer (2014)

[58] Ref: Stolte, T., Bagschik, G., Reschka, A., Maurer, M.: *Hazard analysis and risk assess-ment for an automated unmanned protective vehicle*. In 2017 IEEE Intelligent Vehicles Symposium (IV). pp. 1848–1855. IEEE (2017)

[59] Macher, G., Sporer, H., Berlach, R., Armengaud, E., Kreiner, C.: Sahara: *A security-aware hazard and risk analysis method*. In 2015 Design, Au-tomation Test in Europe Conference Exhibition (DATE). pp. 621–624 (2015). [Online] Available: https://doi.org/10.7873/DATE.2015.0622

[60] Ishimatsu, T., Leveson, N.G., Thomas, J.P., Fleming, C.H., Katahira, M., Miyamoto,Y., Ujiie, R., Nakao, H., Hoshino, N.: *Hazard analysis of complex spacecraft usingsystems-theoretic process analysis.* Journal of Spacecraft and Rockets51(2), 509–522(2014)

[61] Young, W., Porada, R.: *System-theoretic process analysis for security (stpa-sec): Cyber security and stpa*. In 2017 STAMP Conference (2017)

[62] Raspotnig, C., Katta, V., Karpati, P., Opdahl, A.L.: *Enhancing chassis: a methodfor combining safety and security.* In 2013 International Conference on Availability,Reliability and Security. pp. 766–773. IEEE (2013)

[63] Pentti, H., Atte, H.: *Failure mode and effects analysis of software-based automationsystems*. VTT Industrial Systems, STUK-YTO-TR190, 190 (2002)

[64] Macher, G., Armengaud, E., Brenner, E., Kreiner, C.: *Threat and risk assessment methodologies in the automotive domain*. Procedia computer science 83, 1288–1294(2016)

[65] G¨oßling-Reisemann, S.: *Resilience–preparing energy systems for the unexpected*. An edited collection of authored pieces comparing, contrasting, and integrating riskand resilience with an emphasis on ways to measure resilience p. 73 (2016)

[66] Cioroaica E., Kar S.R., Sorokos I. (2022) Comparison of Safety and Security Analysis Techniques. In: Gude Prego J.J., de la Puerta J.G., García Bringas P., Quintián H., Corchado E. (eds) 14th International Conference on Computational Intelligence in Security for Information Systems and 12th International Conference on European Transnational Educational (CISIS 2021 and ICEUTE 2021). CISIS - ICEUTE 2021. Advances in Intelligent Systems and Computing, vol 1400. Springer, Cham. [Online] Available: https://doi.org/10.1007/978-3-030-87872-6_23

[67] F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, andM. Utting, *A Subset of Precise UML for Model-Based Testing,* in Proceedings of the 3rd International Workshop on Advances in Model-Based Testing - A-MOST '07. London, United Kingdom: ACM Press, 2007, pp.95–104.

[68] M. Felderer, B. Agreiter, P. Zech, and R. Breu, *A Classification for Model-Based Security Testing*, in VALID 2011, The Third International Conference on Advances in System Testing and Validation Lifecycle, 2011, pp. 109–114.

[69] D. Xu, M. Tu, M. Sanford, L. Thomas, D. Woodraska, and W. Xu, *Automated Security Test Generation with Formal Threat Models*, IEEE Transactions on Dependable and Secure Computing, vol. 9, no. 4, pp. 526–540, 2012.

[70] A. Cretin, B. Legeard, F. Peureux, and A. Vernotte, *Increasing the Resilienceof ATC systems against False Data Injection Attacks using DSLbased Testing*, in Doctoral Symposium ICRAT, 2018.

[71] W. Li, F. Le Gall, and N. Spaseski, *A Survey on Model-Based Testing Tools for Test Case Generation*, in Tools and Methods of Program Analysis, Itsykson, A. Scedrov, and V. Zakharov, Eds., vol. 779. Cham: Springer International Publishing, 2018, pp. 77–89.

[72] B. Legeard and A. Bouzy, *Smartesting CertifyIt: Model-Based Testing for Enterprise IT*, in 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation. Luxembourg, Luxembourg: IEEE, 2013, pp. 391–397.

[73] *GraphWalker, an open-source model-based testomg tool*. [Online] Available: https://graphwalker.github.io/

[74] MISTA. [Online] Available: http://cs.boisestate.edu/~dxu/research/MBT.html

[75] A. Vernotte, *Research Questions for Model-Based Vulnerability Testing of Web Applications*, in IEEE International Conference on Software Testing, Verification, and Validation Workshops, 2013.

[76] J. Bozic and F. Wotawa, *Security Testing Based on Attack Patterns*, in IEEE International Conference on Software Testing, Verification, and Validation Workshops, 2014.

[77] M. Felderer and E. Fourneret, *A Systematic Classification of Security Regression Testing Approaches*, International Journal on Software Tools for Technology Transfer, vol. 17, no. 3, pp. 305–319, 2015.

[78] M. Bishop, *About Penetration Testing*, IEEE Security & Privacy Maga-zine, vol. 5, no. 6, pp. 84–87, 2007

[79] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, *State of the Art: Auto-mated Black-Box Web Application Vulnerability Testing*, in 2010 IEEESymposium on Security and Privacy. Oakland, CA, USA: IEEE, 2010, pp. 332–345

[80] ISECOM, *The Open-Source Security Testing Methodology Manual (OS-STMMv3*), 2010

[81] M. Sutton, A. Greene, and P. Aminir, *Fuzzing: Brute Force VulnerabilityDiscovery*. Pearson Education, 2007, pp. 1–51.

[82] C. Chen, B. Cui, J. Ma, R. Wu, J. Guo, and W. Liu, *A Systematic Reviewof Fuzzing Techniques,* Computers & Security, vol. 75, pp. 118–137, 2018

[83] M. Schneider, J. Grossmann, I. Schieferdecker, and A. Pietschker, *OnlineModel-Based Behavioral Fuzzing*, in IEEE Sixth International Conferenceon Software Testing, Verification and Validation Workshops, 2013.

[84] P. Tsankov, M. T. Dashti, and D. Basin, *SECFUZZ: Fuzz-testing SecurityProtocols*, in Proc. of the 7th International Workshop on Automation ofSoftware Test, 2012

[85] C. Miller and Z. Peterson, *Analysis of Mutation and Generation-BasedFuzzing*, 2007

[86] W. Krenn, R. Schlick, S. Tiran, B. Aichernig, E. Jöbstl, and H. Brandl, *MoMut::UML model-based mutation testing for UML*, in2015 IEEE 8thInternational Conference on Software Testing, Verification and Validation,ICST 2015 - Proceedings, 2015.

[87] F. Duchene, *Detection of Web Vulnerabilities via Model Inferenceassisted Evolutionary Fuzzing*, Ph.D. dissertation, Grenoble University,2014. [Online] Available: https://hal.archives-ouvertes.fr/tel-01102325/document

[88] J. Bozic and F. Wotawa, *Model-based Testing - From Safety to Security*, STV Bozic, Wotawa, 2012

[89] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, andA. Pretschner, "Chapter One - Security Testing: A Survey," inAdvancesin Computers. Elsevier, 2015, vol. 101, pp. 1–51

[90] S. Bekrar, C. Bekrar, R. Groz, and L. Mounier, *Finding Software Vulner-abilities by Smart Fuzzing,* inFourth IEEE International Conference onSoftware Testing, Verification and Validation, 2011

[91] Yoo y M. Harman, *Regression testing minimization, selection and prioritization: a survey*, Softw. Test. Verification Reliab., vol. 22, n.o 2, pp. 67-120, mar. 2012, doi: 10.1002/stv.430

[92] E. Fourneret, F. Bouquet, Frederic Dadeau, y Stephane Debricon, *Selective Test Generation Method for Evolving Critical Systems*, in 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, Berlin, Germany, 2011, pp. 125-134, doi: 10.1109/ICSTW.2011.95

[93] L. Cseppento y Z. Micskei, *Evaluating code-based test input generator tools*, Softw. Test. Verification Reliab., vol. 27, n.o 6, p. e1627, sep. 2017, doi: 10.1002/stvr.1627

[94] B. Chess y J. West, *Secure programming with static analysis*. Gary McGraw, 2007

[95] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, y W. Pugh, *Using Static Analysis to Find Bugs*, IEEE Softw., vol. 25, n.o 5, pp. 22-29, sep. 2008, doi: 10.1109/MS.2008.130

[96] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. *Chapter six - mutation testing advances: An analysis and survey*. Volume 112 of Advances in Computers, pages 275 – 378. Elsevier, 2019

[97] H. Baars, R. Lassche, R. Massink, y H. Pille, «Smart grid security certification in Europe. Challenges and recommendations». ENISA, 2014

[98] *A Meta-Scheme Approach*. ECSO, Dec-2017 [Online] Available: https://www.ecs-org.eu/documents/uploads/european-cyber-security-certification-a-meta-scheme-approach.pdf

[99] S. Murdoch, M. Bond, R. J. Anderson, *How Certification Systems Fail: Lessons from the Ware Report*, *IEEE Secur. Priv. Mag.*, vol. 10, n.º 6, pp. 1-1, 2012, doi: 10.1109/MSP.2012.89

[100] S. P. Kaluvuri, M. Bezzi, Y. Roudier, *A Quantitative Analysis of Common Criteria Certification Practice*, in *Trust, Privacy, and Security in Digital Business*, vol. 8647, Cham: Springer International Publishing, 2014, pp. 132-143.

[101] AIOTI, *Report on Workshop on Security and Privacy in the Hyper-Connected World*. 2016.

[102] J. Hubner, M. Lastovka, *BOSCH Political Viewpoint. Security in IoT*. 2017.

[103] *Iot Security & Privacy Label* [Online] Available: https://iotsecurityprivacy.org/labels

[104] *CYBERSECURITY MADE IN EUROPE*. ECSO, [Online] Available: https://ecs-org.eu/initiatives/cybersecurity-made-in-europe

[105] M. Bartoletti, P. Degano, G. L. Ferrari, *Security Issues in Service Composition*, in *Formal Methods for Open Object-Based Distributed Systems*, Berlin, Heidelberg, 2006, vol. 4037, pp. 1-16, doi: 10.1007/11768869_1.

[106] Regulation (EU) 2019/881 of the European Parliament and of the Council on ENISA (the European Union Agency for Cybersecurity) and on information and communications technology cybersecurity certification and repealing Regulation (EU) No 526/2013 [Online] Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32019R0881&from=EN

[107] *Methods for Testing & Specification; Risk-based Security Assesment and Testing Methodologies*. ETSI, Nov-2015. [Online] Available: https://www.etsi.org/deliver/etsi_eg/203200_203299/203251/01.01.01_50/eg_203251v010101m.pdf

[108] S. N. Matheu, J. L. Hernandez-Ramos, and A. F. Skarmeta, *Toward a Cybersecurity Certification Framework for the Internet of Things*, IEEE Security Privacy, vol. 17, no. 3, pp. 66–76, 5 2019

[109] Lorrain, E. Fourneret, F. Dadeau, and B. Legeard, *MBeeTle - un outil pour la génération de tests à-la-volée à l'aide de modèles*, in Groupement De Recherche CNRS du Génie de la Programmation et du Logiciel, 2016, [Online] Available: https://hal.archives-ouvertes.fr/hal-02472608

[110] *IoT-LAB : The Very Large Scale IoT Testbed*. FIT (Future Internet Testing Facility) [Online] Available: https://www.iot-lab.info/

[111] *HEAVENS: HEAling Vulnerabilities to ENhance Software Security and Safety* – Project Proposal, 2012.

[112] *NIST SP 800-30 Rev. 1 Guide for Conducting Risk Assessments*. NIST, Sep-2012, [Online] Available: https://csrc.nist.gov/publications/detail/sp/800-30/rev-1/final

[113] HEAling Vulnerabilities to ENhance Software Security and Safety (HEAVENS). D2: Security models (2016)

[114] *Overview of ICT certification laboratories*. ENISA, Jan-2018. [Online] Available: https://european-accreditation.org/wp-content/uploads/2018/10/document-ict-certification-laboratories.pdf

[115] *Directive 2010/30/EU on the indication by labelling and standard product information of the consumption of energy and other resources by energy-related products*. European Commission, 19-May-2010, [Online] Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32010L0030

[116] Katie Boeckl, Michael Fagan, William Fisher, Naomi Lefkovitz, Katerina N. Megas, Ellen Nadeau, Danna Gabel O Rourke, Ben Piccarreta, and Karen Scarfone. 25-Jun-2019. *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks*. [Online] Available: https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8228-draft.pdf

[117] *European Commission. DIRECTIVE (EU) 2016/1148 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 6 July 2016 concerning measures for a high common level of security of network and information systems across the Union*. [Online] Available: https://eur-lex.europa.eu/eli/dir/2016/1148/oj