



Deliverable D6.3

Risk Assessment and Additional Requirements

Technical References

Document version : 1.0
Submission Date : 31/08/2022
Dissemination Level : Public
Contribution to : WP6 – Risk Analysis and Mitigation Strategies
Document Owner : 7B
File Name : BIECO_D6.3_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.biéco.org>

Revision History

REVISION	DATE	INVOLVED PARTNERS	DESCRIPTION
0.0	09.05.2022	7B	Initial draft.
0.1	09.05.2022	7B	Added the first version of Chapter 2 (Accountability using blockchain technology) (mbyra, 7B)
0.2	20.06.2022	RES	Contribution to Section 3.1. General review
0.3	22.06.2022	CNR	Start contributing to section 6. First draft of the content.
0.4	10.07.2022	CNR	Finalized the content of Section 6.
1.0	11.07.2022	7B	Intro, LFB motivation and conclusions.
1.2	11.07.2022	IESE	ResilBlockly and SafeTbox under Section 4.
1.3	13.07.2022	7B	Minor fixes and the Executive Summary.
1.4	15.07.2022	7B	Minor fixes.
1.5	20.07.2022	7B	Final editing and preparing for the internal review.
2.0	25.07.2022	IESE	Internal review.
2.2	29.07.2022	7B	Updates and fixes after the internal review.
2.3	29.07.2022	CNR	Internal review
2.4	08.08.2022	7B	Updates and fixes after the internal review
2.5	10.08.2022	UNINOVA	External Reviewer
2.6	12.08.2022	7B	Updates and fixes after the External review
2.7	22.08.2022	UNINOVA	Coordinator review and Update
3.0	31.08.2022	UNINOVA	Coordinator Finalization and Submission

List of Contributors

Deliverable Editor and Contributors: Marcin Byra (7B), Radosław Piliszek (7B), Paweł Skrzypek (7B), Enrico Schiavone (RES), Enrico Araniti (RES), Riccardo Introzzi (IFEVS), Sara Matheu (UMU), Ioannis Sorokos (IESE), Said Daoudagh (CNR), Eda Marchetti (CNR)

Reviewers: Ioannis Sorokos (IESE), Eda Marchetti (CNR), Said Daoudagh (CNR), Sanaz Nikghadam-Hojjati (UNINOVA), José Barata (UNINOVA)

Disclaimer: The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies.

All rights reserved.

The document is proprietary of the BIECO consortium members. No copying or distributing, in any form or by any means, is allowed without the prior written agreement of the owner of the property rights.

This document reflects only the authors' view. The European Community is not liable for any use that may be made of the information contained herein.



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

Acronyms

Acronym	Term
AC	Access Control
ACL	Access Control List
ACP	Access Control Policy
ACS	Access Control System
AI	AI Investments
CAPEC	Common Attack Pattern Enumeration and Classification
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CS	Computer System
CWE	Common Weakness Enumeration
ETH	Ethereum currency
GDPR	General Data Protection Regulation
GENERAL_D	Gdpr ENforcEmEnt of peRsonAL Data
GROOT	GdpR-based cOmbinatOriAl Testing
ICT	Information and Communications Technology
I'M GROOT	Integrated environMent for GdpR-based cOmbinatOriAl Testing
MUD	Manufacturer Usage Description
NVD	National Vulnerability Database
RUMI	Relied Upon Message Interface
SoS	System of Systems
UC	Use Case
WPX	Work Package X (X = number)

Executive Summary

The main goal of this deliverable is to report the work related to Task T6.4, building upon what has already been delivered in Deliverables D6.1, D6.2 and D6.4 and worked on in Tasks T6.1, T6.2 and T6.3. This deliverable, being the last deliverable of the WP6, reports also all the remaining work made in WP6. Revising what was proposed in the DoA, the deliverable reports the activities done by 7bulls, CNR, UMU, and IESE that focuses on the risk assessment, dynamic system analysis, accountability, auditability, and authorization mechanisms as well as privacy measures. This work was done in collaboration with Task 7.3. Specifically, deliverable D6.3 focuses on the risk assessment and feedback from the dynamic system analysis, while the deliverable D7.3 discusses the remaining tooling and the overall methodology. Moreover, modelling and analysis of use cases of the BIECO methodology and methods of coordinating and reacting to dynamically detected vulnerabilities are discussed.

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

The publication reflects only the author's view, and the European Commission is not responsible for any use that may be made of the information it contains.

Table of Contents

Technical References	1
Revision History.....	2
List of Contributors	2
Acronyms.....	4
Executive Summary.....	5
Project Summary.....	5
Partners.....	6
Disclaimer	6
Table of Contents.....	7
List of Figures.....	9
1. Introduction.....	10
1.1. The Structure of the Document	10
1.2. Relation to Deliverables of Other Work Packages and the BIECO Platform.....	10
2. Accountability Using Blockchain Technology – LogForgeryBlocker.....	11
2.1. Scope of the Tool.....	11
2.2. Functional Description.....	11
2.3. Introduction to Blockchain	12
2.4. Cost.....	13
2.5. Architecture	13
2.5.1. Client-Side Application	14
2.5.2. Server-Side Application	14
2.6. Implementation Description.....	15
2.6.1. Modules and Submodules.....	15
2.6.2. Communication, Hashing, Application State	18
2.7. Installation Guide	21
2.7.1. Backend.....	21
Frontend.....	23
Smart Contract	23
2.8. User guide.....	24
3. Use Case Modelling in ResilBlockly for Threat Analysis	29
3.1. The Applicability of Threat Analysis with ResilBlockly.....	29
3.1.1. AI Investments Use Case Modelling.....	30

3.2. IFEVS Use Case Modelling	35
3.2.1. Microfactory and Cloud IoT.....	35
3.2.2. FOTA and UPTANE	36
3.2.3. FOTA-UPTANE Modelling	37
3.2.4. FOTA-UPTANE Risk Analysis	38
4. Supporting Rating of Safety-Related Impact of Vulnerabilities.....	39
5. Definition of the Threat MUD for Sharing Mitigations	43
5.1. Threat MUD Model.....	43
5.2. Threat MUD architecture	45
5.3. Usages of the threat MUD	47
6. GDPR-Based Accountability and Auditability of Authorization Systems.....	49
6.1. Contextualization of I'M GROOT in BIECO	50
6.2. Results Analysis Component Specification	51
7. Conclusions	53
8. References	54

List of Figures

Figure 1. Architecture diagram of the LogForgeryBlocker	13
Figure 2. Message structure for Agent-Log Proxy communication protocol	19
Figure 3. The flow of the log request and delivery.....	20
Figure 4. The main page of Alchemy	24
Figure 5. New user registration in LogForgeryBlocker	25
Figure 6. Empty list of logs	25
Figure 7. Creating a new agent	26
Figure 8. Non-empty list of agents.....	26
Figure 9. Agent details	27
Figure 10. List of existing log files in the web interface.....	27
Figure 11. List of fingerprints of a selected log file.....	28
Figure 12. Transaction details.....	28
Figure 13. New interface for the specification of custom weaknesses	30
Figure 14. All Architecture Diagram.....	31
Figure 15. Simplified model of the All communication system implemented in Resilblockly.....	32
Figure 16. UPTANE-like architecture of the FOTA system.....	36
Figure 17. Example of representation of the UPTANE based FOTA architecture – the sketch shows the SoS, with the first CS and RUMI (with message description) in the model	37
Figure 18. Graph representation of the UPTANE-like architecture of the FOTA system (the picture shows only a representative portion of the whole graph)	38
Figure 19. ResilBlockly-SafeTbox Workflow	40
Figure 20. Export from ResilBlockly.....	40
Figure 21. Importing into safeTbox	40
Figure 22 - Changing the file format filter to .ecore.....	41
Figure 23. Accessing the properties of an element in safeTbox.....	42
Figure 24. Threat MUD module	44
Figure 25. ACL module	45
Figure 26. Threat MUD and MUD architecture	46
Figure 27. Sharing discovered threats.....	47
Figure 28. Enforce mitigation.....	48
Figure 29. The Reference Process of I'M GROOT	51

1. Introduction

In this deliverable, we report progress in risk assessment and related aspects of dynamic system analysis. To this end, we discuss matters related to auditability and non-repudiation of the system logs and propose a blockchain-based tool to ensure that system logs are protected against forgery (LogForgeryBlocker). Moreover, we discuss the accountability and auditability of authorization mechanisms to enable compliance with the General Data Protection Regulation (GDPR) [20]. We also show how the ResilBlockly tool can be used to model and analyze the use cases of BIECO. Finally, we show a framework for coordinating and reacting to dynamically detected vulnerabilities via the threat Manufacturer Usage Description (MUD).

1.1. The Structure of the Document

To provide the auditability and non-repudiation of the log files, we describe the LogForgeryBlocker: a blockchain-based tool for protecting against log forgery, along with a detailed background discussion, architecture description, and installation and user guides. Then, we present the analysis of the BIECO use cases with the ResilBlockly software and how SafeTBox can enhance it. Moreover, we show a framework for reacting to dynamically detected vulnerabilities with the thread MUD to complement the dynamic system analysis and assessment presented in this deliverable. Then, we discuss the accountability and auditability of the authorization mechanisms to satisfy the GDPR which deals with privacy aspects. In particular:

- Chapter 2 describes the LogForgeryBlocker tool with details relevant both for end users and future developers.
- Chapter 3 contains a description of the modelling and analysis of use cases in the ResilBlockly tool.
- Chapter 4 describes how ResilBlockly analysis can be enhanced with SafeTBox in the BIECO framework.
- Chapter 5 is dedicated to the threat MUD used for vulnerability details sharing.
- Chapter 6 describes GDPR-based accountability and auditability of authorization systems.

The deliverable ends with conclusions summarising the work reported in it.

1.2. Relation to Deliverables of Other Work Packages and the BIECO Platform

This deliverable builds upon the work foundation laid by two deliverables from WP6: D6.1 [11] and D6.2 [12] as well as the deliverables of WP7: D7.1 [9], D7.2 [10] and, parallel to this one, D7.3 [8]. The WP6 deliverables introduced ResilBlockly and its enhancements relevant to the BIECO platform. This tool is presented here used for the modelling and analysis of the BIECO use cases' platforms: the AI Investments and IFEVS. The WP7 has collected claims and built a methodology and tooling to certify the system under scrutiny. In this deliverable, we focus on providing feedback on the dynamic aspects of risk assessment and related issues. We focus on handling the auditability, accountability, and non-repudiation of the system logs; we describe authorization systems, with emphasis on privacy with GPDRs. We also propose a framework for reacting to dynamically detected vulnerabilities.

2. Accountability Using Blockchain Technology – LogForgeryBlocker

As the BIECO framework aims to provide a complete solution to all security-relevant aspects of the ICT systems, the need to detect and react to runtime issues is paramount. Some of the auditability measures may depend on the quality of the collected log files and this is where LogForgeryBlocker finds itself applicable.

LogForgeryBlocker is being proposed as an open-source tool providing non-repudiation of logs with the help of blockchain technology. It is an easily configurable tool that does not require advanced expertise from the user. Its typical use is to provide a durable medium for recording logs so that it is easy to identify a possible malfunction or intrusion by unfolding and localizing the first illegitimately changed entry. The assumption is that local agents track the log files in real-time and communicate with the main application on a separate cluster, putting the information into a selected blockchain system. Using the blockchain technology allows LogForgeryBlocker to ensure non-repudiation of the logs. Once the logs (or hashes of the logs, as will be explained in detail in the following sections) are stored in blockchain, they cannot be deleted or modified, having the same security guarantees as the whole blockchain used (see section 2.3 below)

The following sections describe the LogForgeryBlocker tool in detail.

2.1. Scope of the Tool

The main feature of the LogForgeryBlocker is providing a lightweight, configurable way to provide non-repudiation of software logs with blockchain and provides an intuitive interface for a user who does not have to be a blockchain expert. In other words, if the LogForgeryBlocker runs, the user can be sure that the logs cannot be modified without their knowledge. [19] The tool can be installed on multiple machines of one client and coordinate log management with a central server.

2.2. Functional Description

From the functional perspective, the tool is divided into two parts: Server and Client.

The server-side is responsible for providing the main backend functionality of the solution: a database, REST APIs providing communication with other modules, authentication, a web interface, and a blockchain communication module along with smart-contract-related code and configuration. The user does not have to have deep knowledge about the blockchain; what is required to provide the configuration parameters (address, etc.) of the blockchain, i.e., the LogForgeryBlocker has to know where the smart contract should be deployed to. After providing this information, it automates generating, monitoring, and pushing the required data to the blockchain without user effort.

The client-side application is installed into a machine with the user software to monitor log files, gather new logs, and send them to the server-side using a secure communication protocol. The client-side is divided into two components: the Client and the File Proxy. There can be multiple File Proxies installed on different machines, if the User software and logs are distributed. They track the user-selected log files and folders,

calculate their hashes and send them to the main Client, which then acts as a communication hub with the server-side.

This approach provides a practical and lightweight solution, allowing the user to provide non-repudiation of the software logs, even when the software is distributed on many machines. Furthermore, the user can provide configuration (files or folders to track, time intervals for pushing the new logs to the blockchain, etc.) and monitor all processed logs in real-time using the web interface.

2.3. Introduction to Blockchain

In this section, we introduce basic ideas of the underlying technology from a high perspective to provide a common ground to understand the architecture and functionality of the LogForgeryBlocker described in the following chapters.

Blockchain is a distributed software network, often described as a shared, immutable ledger, that enables tracking the transactions and secure transfer of assets without an intermediary. An asset can be both 'material' (a house or cash) or 'virtual' (a copyright, brand, patent) entity. Blockchain provides immediate and completely transparent information stored on a shared, immutable medium (the ledger). It can be used to track payments, orders, and a wide variety of other information, guaranteeing a single view of truth [18].

Blockchain first appeared in 2008 when Bitcoin, a secure peer-to-peer electronic cash system, was proposed¹. While it remains the most popular cryptocurrency, from the perspective of this document, the essential fact is that Bitcoin was the first blockchain system that started a whole new industry.

Today, there are multiple types of blockchains, including dozens of cryptocurrencies. These systems are designed to meet various requirements. For example, there are private and open blockchains, permissioned or built by consortiums. They differ in implementation, technical details, usage, and purpose, but the underlying basics remain the same. All blockchains have distributed ledgers containing immutable records of non-duplicated transactions. The records are immutable: once they are in the blockchain, they can never be modified. A fundamental element deployed into blockchain is a smart contract, a set of rules defining conditions of the flow of the assets.

Each new transaction (new record, deployed smart contract) is recorded as a part of a block of data. These blocks are connected to the previous and the following blocks, hence the name: blockchain. Each new block relies on the combined information of the previous blocks and strengthens the chain even more. It brings trust, security, and effectiveness to a variety of applications.

LogForgeryBlocker implements its smart contract with Solidity². Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs that govern the behaviour of accounts within the Ethereum state.

¹ <https://bitcoin.org/bitcoin.pdf>

² <https://soliditylang.org/>

2.4. Cost

The sections below describe configuring LogForgeryBlocker with the Ethereum blockchain and Alchemy service. The reasoning behind this description is that it is a quick and straightforward way of configuring the LogForgeryBlocker. We assume the user would like to configure and test the tool first and later start using it in the production environment.

Using a public blockchain implies incurring costs for it as each transaction requires a payment. In Ethereum, the fee is called gas, but the rule generalises to other blockchain solutions as well. The user wishing to use the LogForgeryBlocker must decide where to store the log hashes. They can use the Ethereum *mainnet* and buy gas to pay for the transactions or other popular networks. These usually provide the highest security because of the scale. There are also other solutions that do not involve using *mainnets* and require significantly lower fees. User can also implement their own chain. However, the cost and work needed to create the infrastructure should be considered.

A comprehensive discussion of blockchains is by far beyond the scope of this deliverable. This section, therefore, is intended to underline two factors:

- Users can follow the guide to configure the LogForgeryBlocker and the related blockchain account using the suggested free-tier solutions to configure LogForgeryBlocker and get it to work initially.
- For long-term production environment monitoring, the user must decide which blockchain system to use. LogForgeryBlocker itself is a tool for gathering, hashing, and sending the logs and requires only adding a reference to the blockchain of choice in the configuration file. The location of storing the logs is the user's decision; it is like a database system: a database producer provides the database software to manage the data, but the client is responsible for providing the underlying disk space to store the data.

2.5. Architecture

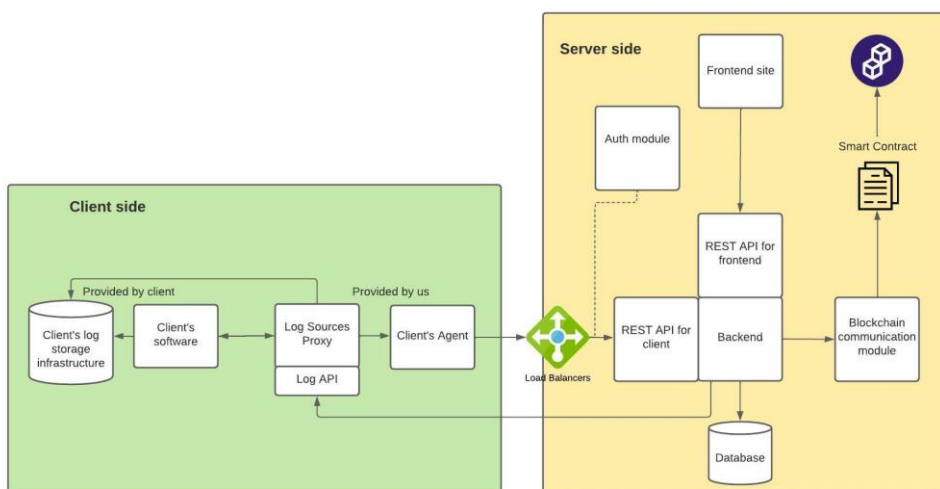


Figure 1. Architecture diagram of the LogForgeryBlocker

A general and formal view of the LogForgeryBlocker architecture is presented in Figure 1. The system was designed as a set of multiple microservices which can be labeled into two categories: Client-side and Server-side.

2.5.1. Client-Side Application

The Client-side application contains elements such as the Client's Agent, Log Sources Proxy, and Log API. The Client's software and log storage infrastructure are not parts of the LogForgeryBlocker but are represented in the diagram as logical parts of the system. An entire Client-side application will be often referred to as a Client Application or Client for the simplicity of the description, despite the fact that it consists of separate modules. The Client can be installed on multiple machines by multiple users, and from their perspective, it can look like a single software package.

The Client's Agent is responsible for providing communication between Log Sources Proxy and Backend. Its tasks are scheduling synchronization, log hashing, and validating logs. It is designed with automatic state restoration after launch or reset to provide a seamless experience for the user.

The Log Sources Proxy's task is to listen for new logs in selected files and directories. It is the only component that requires direct access to the logs in the Client's storage.

Log API is a RESTful API responsible for communication with the Backend.

Client's Software represents a program that generates logs LogForgeryBlocker must track.

Client's log storage infrastructure is a store for log files generated by the Client's Software, e.g., a partition on a hard disk or an NFS.

2.5.2. Server-Side Application

As a server-side application, we refer to all of the components and services that do not belong to the Client-side application. However, there are multiple independent modules, as presented in Figure 1. They consist of the Backend module, REST APIs for the Client and for the frontend, Auth module, Frontend module, Database, and Blockchain communication module, which uses defined Smart Contracts to finalize the whole flow.

Backend can be described as a central point of the architecture. It communicates with the Agents and Frontend module, stores metadata about secured logs, and communicates with a smart contract.

REST API for the client and REST API for the frontend are separate RESTful APIs and underlying services designed for communication with the Client and the Frontend module.

Auth module is responsible for providing authentication and authorization for Clients and Agents.

Frontend module is a web server and all the libraries responsible for providing a web interface for the LogForgeryBlocker. It allows user to easily administer, configure, and monitor the application.

Database is a relational database designed to store all the information about Users, Agents, Logs, configuration, and other activity data.

Blockchain communication module is a separated part of the Backend focused on providing communication with the external blockchain systems, using available configuration parameters.

2.6. Implementation Description

This section gives implementation details of each module of the LogForgeryBlocker, though focusing on the implementation details of each module rather than its general placement in the architecture. Therefore, it is not divided into Client-side and Server-side categories, and some modules are specified in a single paragraph to clarify the description.

2.6.1. Modules and Submodules

Log Sources Proxy

Log Sources Proxy is implemented as a Python application. The aim is to track the selected files or entire directories for new logs to process them in near-real-time and pass the information to the Agent.

The obvious idea to periodically check a file opened in a read-mode for changes would result in high stress for a host machine after adding many files or directories to track, even if most of the logs were completely inactive. Therefore, Log Sources Proxy subscribes to filesystem events instead, using the `pyinotify`³ library. To be specific, Log Sources Proxy listens to the `IN_MODIFY` event for files, and, for directories, `IN_CREATE`, `IN_MOVED_TO` (initializing new file watchers), `IN_DELETE`, `IN_MOVED_FROM` (closing file watchers). Records are split and processed one by line, using a line number and a timestamp. Log Sources Proxy is currently available only for Linux systems.

Client's Agent

Client's Agent (or simply Agent) is a Python application responsible for getting information about log files and changes from Log Sources Proxy, processing and managing the data, and communicating with the Backend. It is a multi-module application, and more than one element of the architecture diagram (see Figure 1.) is discussed here since it is more intuitive to discuss them together.

Scheduler

Scheduler is a module of the Agent responsible for initializing synchronization with the backend if any of the following conditions are met:

- Time of the day

³ <https://pypi.org/project/pyinotify/>

- Time interval
- The maximum size of a log file exceeded

A separate scheduler can run for each log source (file, database, etc.) with its own trigger settings. The scheduler was built on the top of `apscheduler`⁴.

Backend

Backend is written in Typescript, with Node.js and Express.js libraries.

For communication with the Agent and frontend, there are two separate services implementing REST APIs. It is intuitive to present the functionality of the backend by listing the available controllers:

- `auth`: Handles registering and logging of new users. Issues tokens, described in the document about authorization and authentication.
- `organization`: Allows for the creation of management of organizations - i.e., corporate clients of LogForgeryBlocker.
- `agent`: Endpoints for managing agents of organizations. Allows for creating new agents, activating/deactivating, and configuring them.
- `log`: Endpoints for creating new logs, to which snapshots can be added and fetching information about them. Each log creation triggers a blockchain service to add information about such a log to our smart contract.
- `snapshot`: Endpoints for creating new snapshots and receiving information about them. Each snapshot creation triggers a blockchain service to add information about the snapshot to our smart contract, and transaction for that is also saved in the snapshot.
- `verification`: Endpoints for log verification, agents call this endpoint to update information about log cohesion.

Database

PostgreSQL⁵ is used as a database system. It is run as a Docker container. Figure 2 in the previous section (Architecture) presented the database schema implemented in LogForgeryBlocker. The backend uses Prisma with PostgreSQL⁶ as an open-source ORM solution.

Moreover, Redis⁷, an open-source in-memory data store, is used for

- Adding new logs and transactions to the blockchain for separating calling smart contract from the controller logic.
- Verifying whether agents submit logs in configured time periods.

⁴ <https://apscheduler.readthedocs.io/en/3.x/>

⁵ <https://www.postgresql.org/>

⁶ <https://www.prisma.io/>

⁷ <https://redis.io/>

Auth module

The Auth module is responsible for authentication and authorization in the LogForgeryBlocker. It implements its own tokens that must be used in endpoints.

The authentication and authorization system of LogForgeryBlocker is based on custom JWT⁸: an open-source industry-standard method for representing claims securely between two parties.

There are two types of entities in the system: User and Agent. They have different types of tokens and can access resources. For example, some endpoints are only accessible with user tokens, while others might be accessible with agent tokens.

User token schema is presented in the listing below:

```
export type UserToken = {  
  type: 'USER_TOKEN'  
  userId: string  
  username: string  
  roles: Role[] // 'USER' or 'ADMIN'  
  organizationId?: string  
  iat: number  
  exp: number  
}
```

The user token can be acquired by calling either of the endpoints:

```
/v0/auth/login  
/v0/auth/register
```

Agent token schema is presented in the listing below:

```
export type AgentToken = {  
  type: 'AGENT_TOKEN'  
  agentId: string  
  organizationId: string  
  iat: number  
  exp: number  
}
```

The agent token can be acquired by calling:

```
/v0/auth/getToken/:id
```

The auth module implements custom express middlewares that can verify tokens and optionally verify roles.

⁸ <https://jwt.io/>

userAuthMiddleware(roles?: Role[])
agentAuthMiddleware

Blockchain module

The blockchain module of the LogForgeryBlocker is a module inside the main Backend application. It uses web3.js: a collection of libraries allowing interaction with a local or remote Ethereum node using HTTP, IPC, or WebSocket.

In the current version of the LogForgeryBlocker, the contract is configured inside config.ts, where the network and contract address can be specified. The following versions are planned to extend managing configuration related to smart contracts and blockchain systems.

The smart contract is straightforward itself because of the simplicity of the underlying task: what is required from this module is to add new hashed logs to a blockchain to ensure non-repudiation and to verify that the logs are not modified.

Frontend

The frontend is a simple webpage created with React. It provides access to essential features of the LogForgeryBlocker:

- Logging in to a user account
- Viewing the tracked log files or directories
- Viewing the calculated hashes
- Verifying the hashes

It will be described in the User Guide section below.

2.6.2. Communication, Hashing, Application State

Local communication

Local communication, i.e., between the Agent and the Log Proxy, uses TCP sockets and Google Protobuf⁹ for serializing messages. This simple and high-performance solution provides two features essential for the Agent: message ordering and the ability to collect records from machines different from the Agent's one while keeping both sides informed about the active status of the connection.

⁹ <https://github.com/protocolbuffers/protobuf>

Message type	Contents length	Contents
1 byte	int32 4 bytes big-endian	Length varies Contains serialized Protobuf message

Figure 2. Message structure for Agent-Log Proxy communication protocol

In the current development version, messages are not compressed and not encrypted. Encryption and compression will be implemented in the following phase of LogForgeryBlocker development. They are not expected to cause significant performance overhead in the context of time scale of log gathering and communication frequency.

External Communication

External communication is implemented using the HTTP protocol and the 'requests' python library. The application uses a bearer token to authenticate requests, which is a common safety mechanism¹⁰. The token should be provided by creating an entry "TOKEN" in the .env file before the application starts. The backend responds with a JSON containing only two fields:

- Success (true/false)
- data

All the functions necessary for communications can be found in the Agent, in the backend-connector module.

Log Contents Request

Because the Agent has no direct access to logs and wants to verify the logs automatically, it needs some way of requesting content from proxies. Therefore, a small protocol was built on the top of agent-proxy communication, which allows querying proxies for specified log content ranges.

Single content request to one log proxy process flow is:

- The Agent creates the request and gives it a unique request id
- Agent sends GET_LOG_CONTENT message to the proxy
- The proxy receives message, and checks if it has a specified log
- if the log was found, it sends LOG_CONTENT_STATUS message with FOUND_AND_BEGIN_SEND status
- if the log was not found, it sends LOG_CONTENT_STATUS message with NOT_FOUND status (steps 4. and 5. are not executed anymore)
- Proxy loads the requested content range and sends it in batches (of 20) using LOG_CONTENT_DATA messages.
- After all available records have been sent, the proxy sends LOG_CONTENT_STATUS message with END_SEND status.

¹⁰ <https://oauthlib.readthedocs.io/en/latest/oauth2/tokens/bearer.html>



Figure 3. The flow of the log request and delivery

The flow presented in Figure 4 is for one request (one log, one proxy). But in the current system, we can't determine which proxy is responsible for which log. So, if we want to get log contents, we must broadcast the request to all connected proxies. Now it is possible (but still not correct for the system) that two log proxies report they found some log. The strategy for selecting the "leading" one is simple - it is the first that reported it found the log. "Leading" request from the request batch is treated as THE request - its content will be treated as the response for the whole request batch.

Log Hashing

Another essential feature of the Agent is the log hashing. It serves two purposes: logs are hashed as soon as possible so that even the backend doesn't know the real contents of the logs; moreover, storing only the hashes of the logs in blockchain allows for a massive reduction of the cost. Logs are hashed as follows:

- If there are no entries, the new hash (snapshot) is created by running the hash function on the entry data.
- Otherwise, the existing hash is prepended to the log data before running the hash function. The result of that operation becomes the new hash.

After a specified time, the hash is sent to a backend, and the whole process starts over.

By default, a standard sha256 algorithm [17] is used as a hash function. However, the Agent enables using any custom hash function by modifying the *hash_fun* parameter located in the agent.log module.

Application State

The Agent application state consists of four primary elements:

- List of uploaded logs and corresponding snapshots in the LogCollector class.
- Scheduler configuration.
- List of active proxies in the LfbAgent class.
- List of pending validation requests in the LogValidator class.

Every agent's instance automatically restores the necessary part of its state on launch:

- The list of logs and snapshots is downloaded from the backend.
- Scheduler configuration is downloaded from the backend.
- The list of active proxies is lost but will be restored as soon as the application receives a request from every one of them.

The list of pending validation requests is lost.

2.7. Installation Guide

This section contains instructions on how to install each module of LogForgeryBlocker, both on the Client and Server machine. It also describes how to run the software. The software is developed in four separate packages (repositories) in the current version, reflecting the architecture diagram presented in Figure 1. In the following versions, there will be a more automated installation script for the Client and Backend.

2.7.1. Backend

Prerequisites:

- yarn
- Node.js
- Docker

Installation steps:

1. Clone the repository with the Backend:

```
git clone  
https://github.com/LogForgeryBlocker/LogForgeryBlocker_Public.git
```

2. Create an .env file in the main directory and fill the required parameters (these parameters are both confidential and related to a particular user, so they cannot be placed inside the repository and must be filled at this step):

```
DATABASE_URL="postgresql://admin:password@db:5432/lfb-  
database?schema=public"
```

```
JWT_SECRET_KEY=<Authentication secret key>
```

```
PRIVATE_KEY=<Blockchain Wallet Private Key>
```

```
ALCHEMY_API_KEY=<Api Key for Alchemy Api>
```

3. **Version 1 (default):** to run in Docker: build all docker images

```
docker compose build
```

4. Run all backend images (add `-d` to run in the detached mode, add `backend` or `db` at the end to run only a specific container):

```
docker compose up
```

5. **Version 2:** Run locally (without Docker): install dependencies with yarn:

```
yarn install
```

6. Build the app to the dist directory

```
yarn build
```

7. Start the server (on default port 3000):

```
yarn start
```

Important note: if using Version 2, you must have a PostgreSQL database running (you can base it on the default parameters specified in the .env file's DATABASE_URL above)

Agent and File Proxy

Prerequisites:

- Python 3.10 or newer
- protoc

Installation steps for Agent:

1. Clone the repository with the Agent and File proxy:

```
git clone https://github.com/LogForgeryBlocker/LogForgeryBlockerAgent_Public.git
```

2. Create an .env file in the main directory (if not existing) and fill the required parameters:

```
BACKEND_ENDPOINT=<backend server address:port>
```

```
STATE_CONTROL_INTERVAL=<interval between checking the current state to  
send snapshots to backend, and requesting config from backend>
```

```
LOGS_CONTROL_INTERVAL=<interval for validating logs>
```

```
AGENT_ADDR=<hostname to which bind socket for listening to the new log  
proxy connections>
```

```
AGENT_PORT=<port number for the agent address above>
```

```
TOKEN=<token used to authorize to backend>
```

3. Launch the Agent and start listening for log proxies:

```
python -m agent.agent_init
```

Installation steps for File Proxy:

1. Clone the repository with the Agent and File proxy (note, you can use the same repository if you have already copied it following the Agent installation steps):

```
git clone https://github.com/LogForgeryBlocker/LogForgeryBlockerAgent_Public.git
```

2. Create an .env file in the main directory (if not existing) and fill the required parameters:

```
FILEPROXY_WATCHED_PATHS=<filepaths or directory names to watch. Paths  
should be split using the ; sign>
```

```
AGENT_ADDR=<agent's hostname>
```

```
AGENT_PORT=<agent's port>
```

3. Launch the File Proxy, connect to the specified agent and begin listening to file changes (currently working only on Linux platforms):

```
python -m agent.agent_init
```

Frontend

Prerequisites:

- npm version 6 (you can install it with `npm i -g npm@6`)

Installation steps:

1. Clone the repository with the Agent and File proxy (note, you can use the same repository if you have already copied it following the Agent installation steps):

```
git clone https://github.com/LogForgeryBlocker/LogForgeryFrontend_Public.git
```

2. install dependencies:

```
npm i
```

3. run the app:

```
npm start
```

4. Open `http://localhost:3000` to show the interface in the browser

Smart Contract

Prerequisites:

- yarn
- An account on <https://www.alchemy.com/> and a new project with an API key
- Private key of your account

Installation steps:

1. Clone the repository with the Agent and File proxy (note, you can use the same repository if you have already copied it following the Agent installation steps):

```
git clone  
https://github.com/LogForgeryBlocker/LogForgeryBlockerSmartContract_Public.git
```

2. Build and test smart contract:

```
yarn && yarn build && yarn test
```

3. Deploy your smart contract:

```
PRIVATE_KEY=<private_key> API_KEY=<api_key> npx hardhat run  
scripts/deploy.js --network ropsten o
```

2.8. User guide

Assuming all the steps described in section 2.7 Installation guide are successfully executed, except for filling some configuration data, this section provides a quick start guide on using the LogForgeryBlocker.

If one starts with no project to deploy the smart contract to, we can recommend using one of the free solutions, like Alchemy¹¹ to start a new project, using *testnet* (e.g., Goerli) for tests or the *mainnet* for the real scenario. Then, the API key can be copied from the webpage and used as an `ALCHEMY_API_KEY` in the backend configuration.

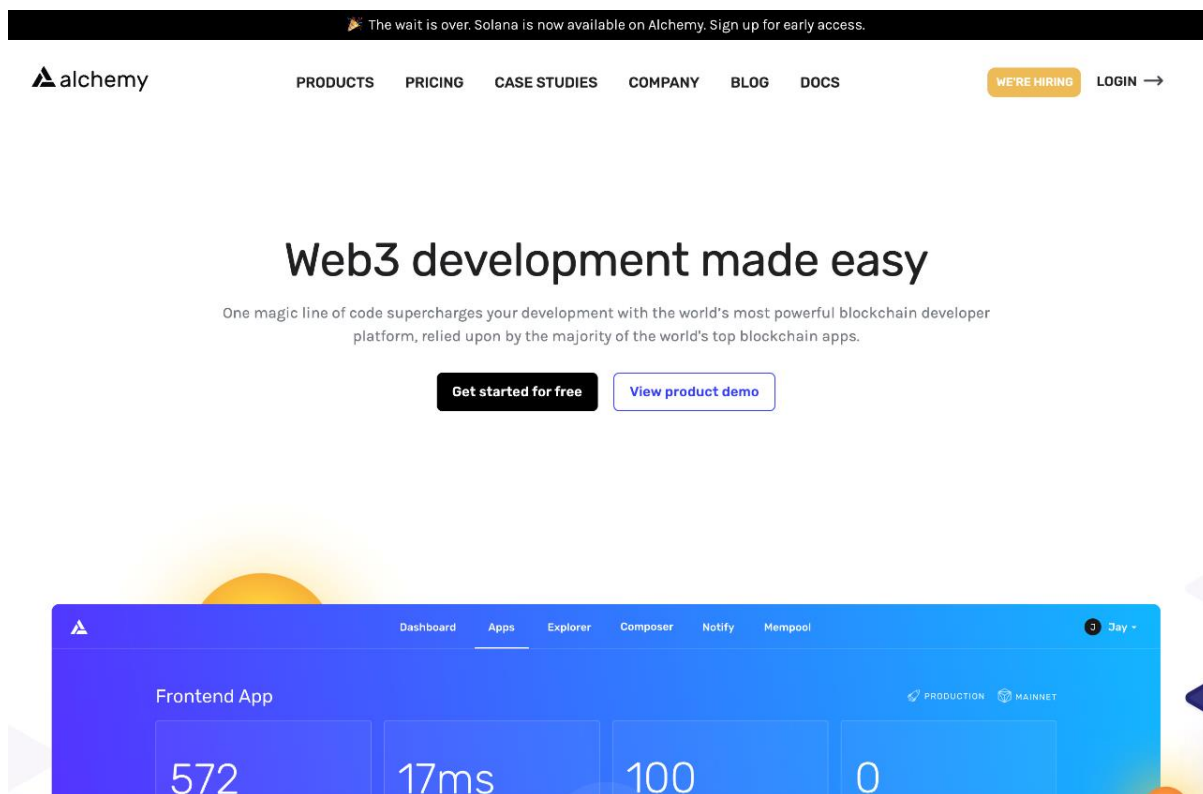


Figure 4. The main page of Alchemy

Similarly, `SECRET_KEY` should be a key to the user's wallet, containing some ether to deploy a smart contract with the hashed logs.

¹¹ www.alchemy.com

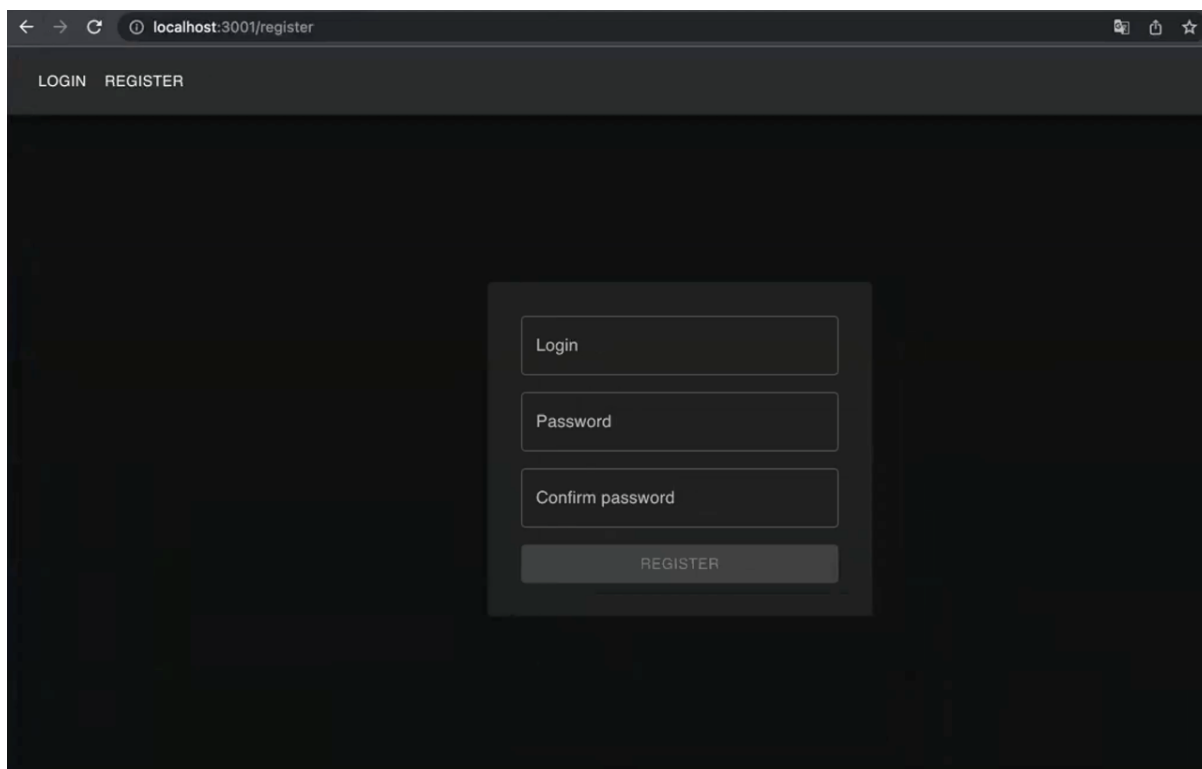


Figure 5. New user registration in LogForgeryBlocker

The next step is registering a new user in LogForgeryBlocker, using a button in the web interface. figure 5 shows the registration page.

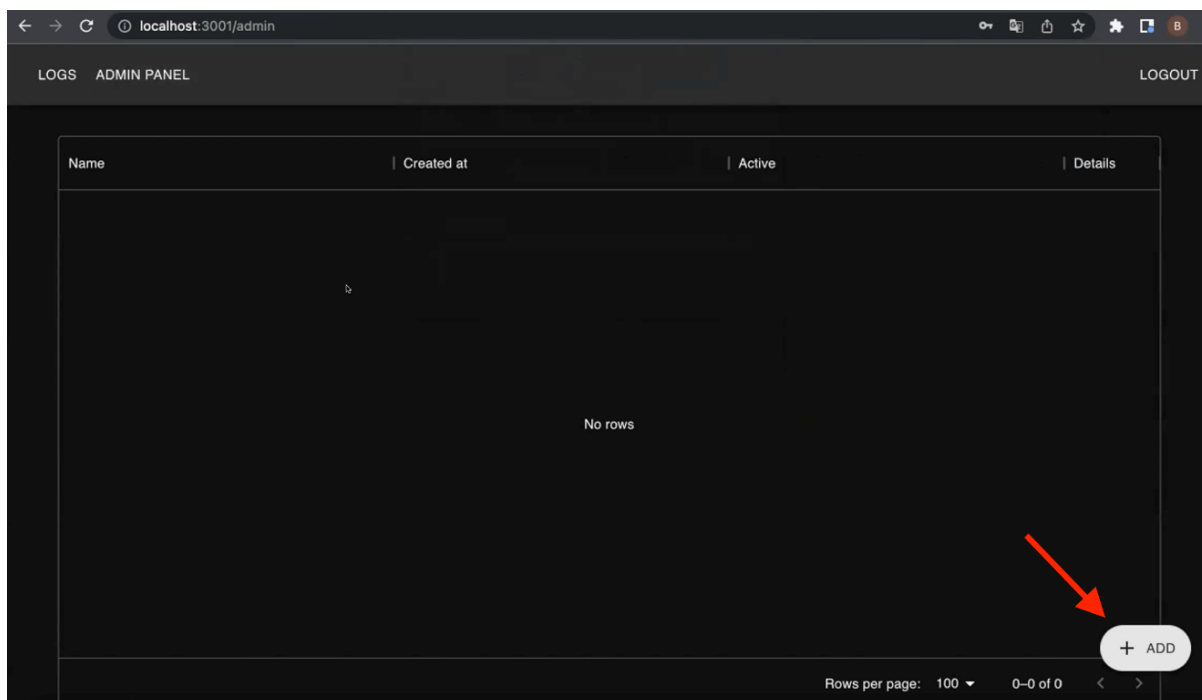


Figure 6. Empty list of logs

After registering the account, the user can see an empty list of logs. A new agent must be added by clicking the Add button, as indicated by the arrow in figure 6 above.

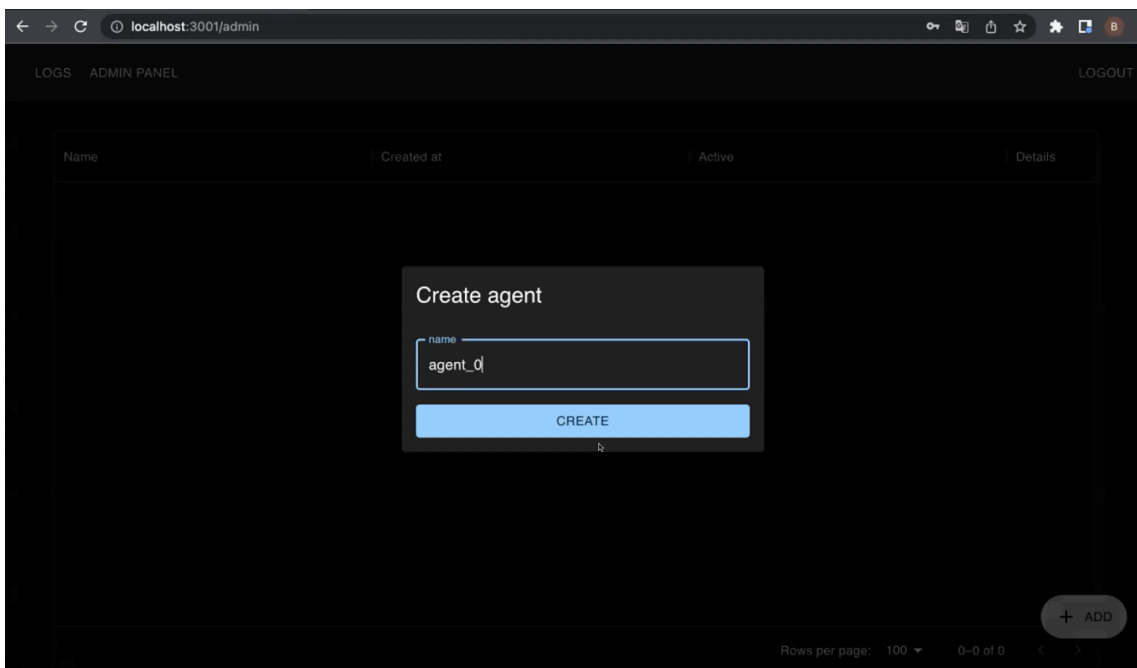


Figure 7. Creating a new agent

Creating a new agent requires only a name (Figure 7), but more steps will be needed to configure it later.

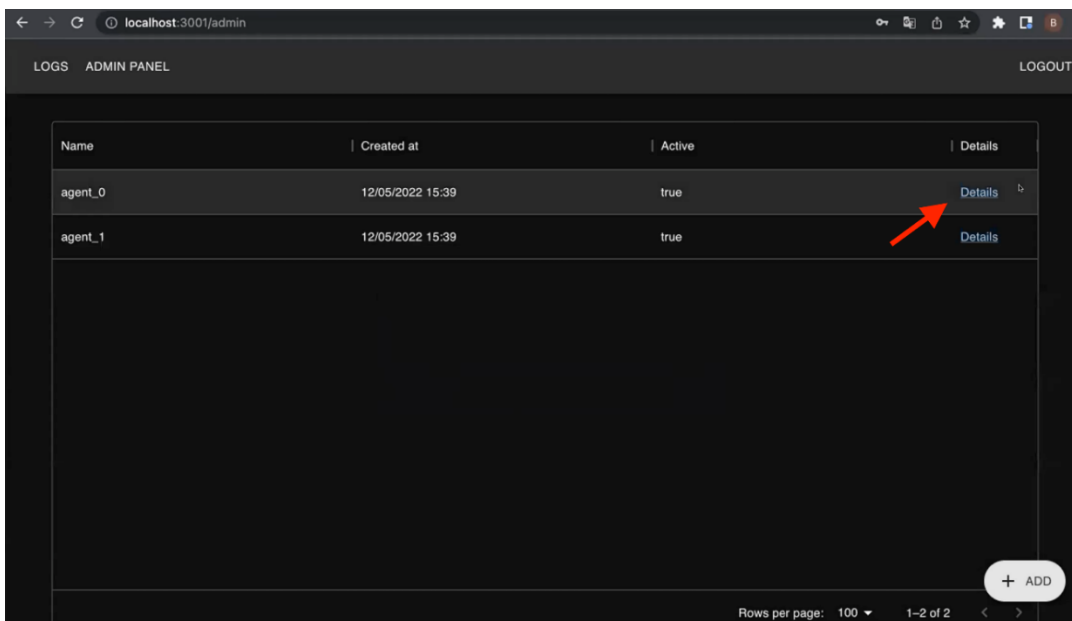


Figure 8. Non-empty list of agents

Enter the configuration page for the Agent, clicking on the Details button indicated by the red arrow in Figure 8.

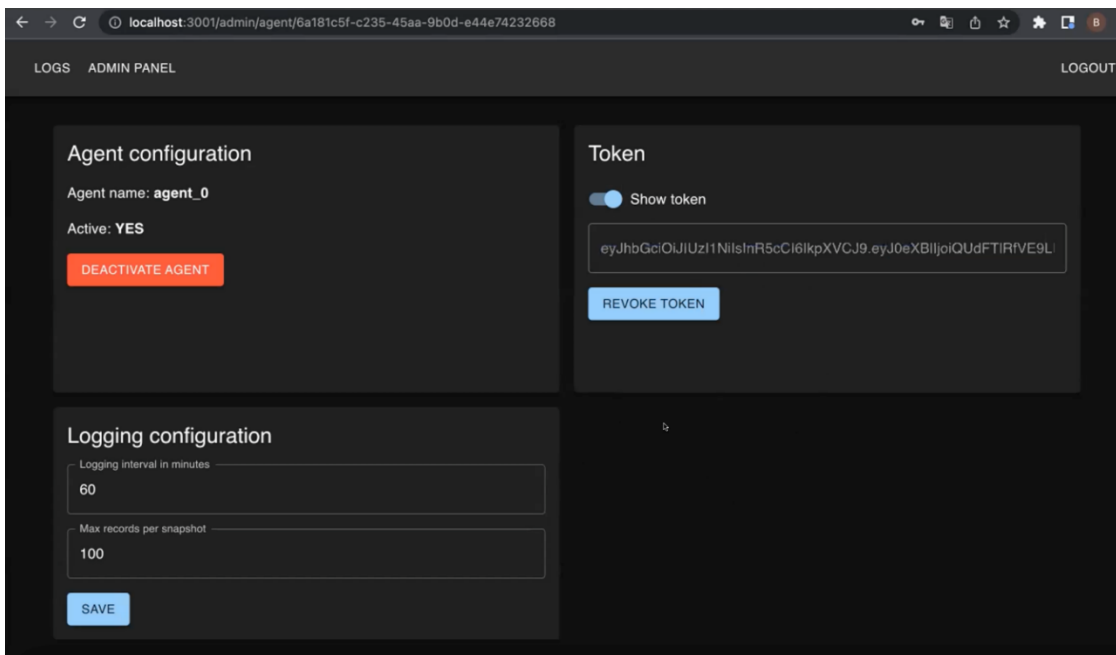


Figure 9. Agent details

Figure 9 shows the configuration page for the Agent. It provides the functionality to activate or deactivate the Agent, configure time intervals, and copy a token required for the communication. The token should be saved into TOKEN parameter in Agent’s .env file, as described in Section 2.7.

Now, you can run the Agent and the File Proxy (or multiple File Proxies). File Proxy will start tracking the log files or directories and communicating the new changes to the Agent.

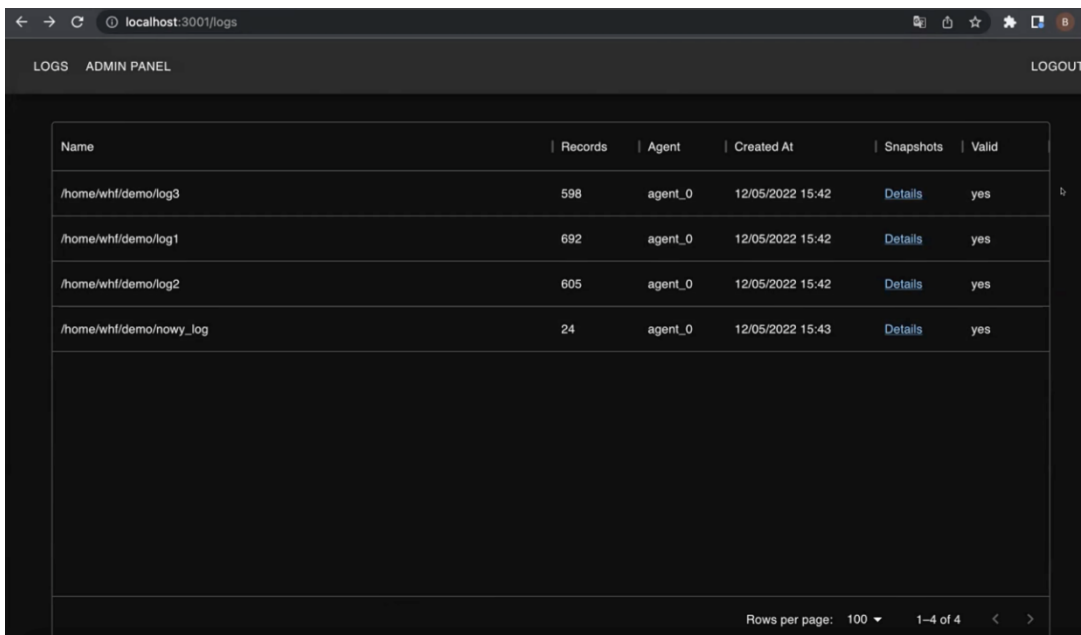


Figure 10. List of existing log files in the web interface

The web interface will show a list of all tracked log files (see Figure 10).

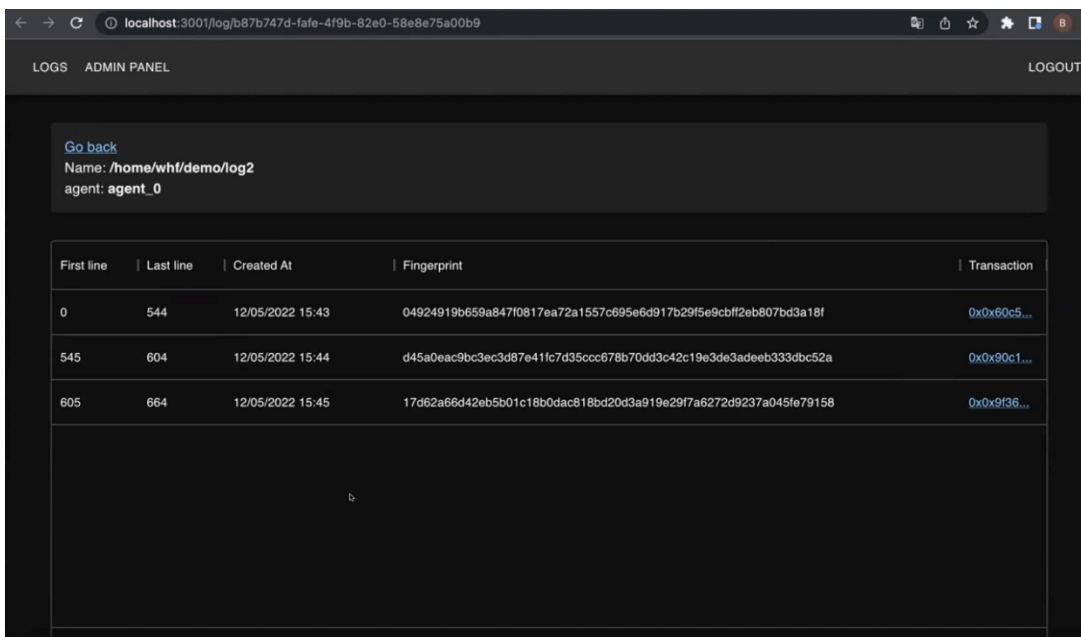


Figure 11. List of fingerprints of a selected log file

After selecting one of the log files (by clicking on the Details button), the user can inspect the history of the blocks of the log that have already been pushed to the blockchain.

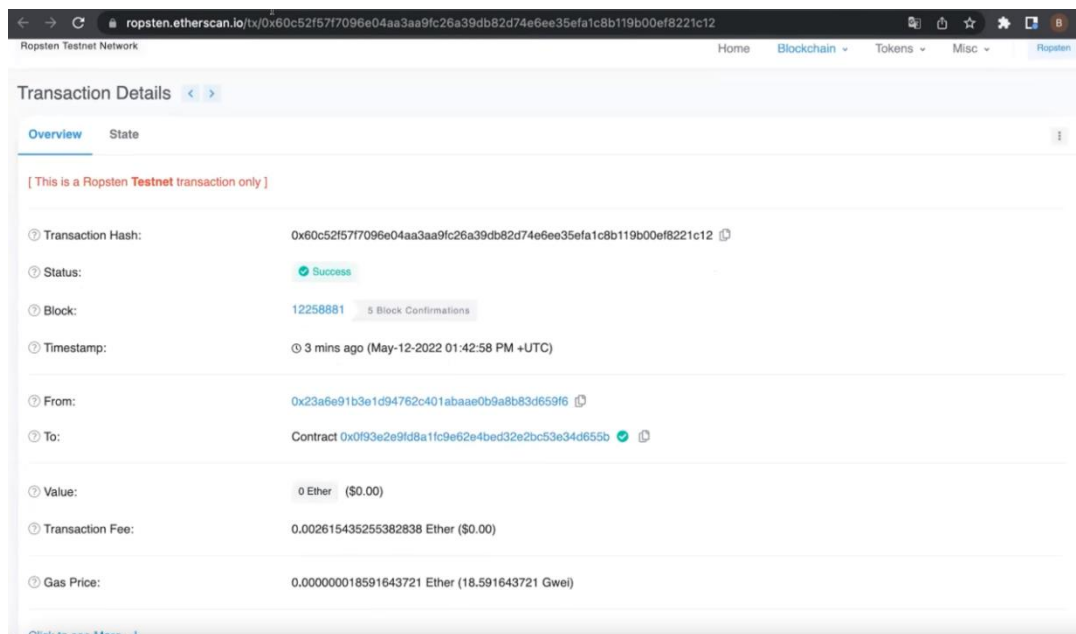


Figure 12. Transaction details

Moreover, by clicking on the transaction number, the user is redirected to the transaction details, where more details of the transaction can be inspected.

3. Use Case Modelling in ResilBlockly for Threat Analysis

Security measures can be identified as a result of detailed threat analysis and risk assessment activities targeted at highlighting potential flaws in the system, or entity taking part in the supply chain, that is under analysis and mitigating or possibly eliminating them. The ResilBlockly tool, proposed as part of the BIECO framework and already introduced in deliverables D6.1 and D6.2, provides features dedicated to the threat analysis and risk assessment. In this deliverable, we give a quick overview of the tool and show how it has been applied in real-world use case modelling scenarios.

3.1. The Applicability of Threat Analysis with ResilBlockly

For the aim of completeness, this Section summarizes the ResilBlockly description already provided in deliverables D6.1 [1] and D6.2 [2]. Indeed, ResilBlockly allows the identification and analysis of weaknesses and vulnerabilities, and the tool offers the possibility to associate custom weaknesses and vulnerabilities in addition to the threats retrieved from the online catalogues. This is important because it allows to make use of the outputs of the Vulnerability Assessment activities conducted and described in the context of WP3, independently from their origin.

As described in D6.1 [1], ResilBlockly is a Model-Driven Engineering software that evolves an existing tool called Blockly4SoS and which, in the context of BIECO, has been provided with a set of new features for addressing typical challenges of ICT supply chains and ecosystems.

Two fundamental concepts, which identify the two main features of ResilBlockly are:

- Profile, is an abstraction of components and relationships for a specific domain;
- Model is an instance of a profile.

Different profiles and models can also be created from scratch, or as an extension of existing profiles or models (for example, previously created within ResilBlockly, shared with other ResilBlockly users, or imported from external sources).

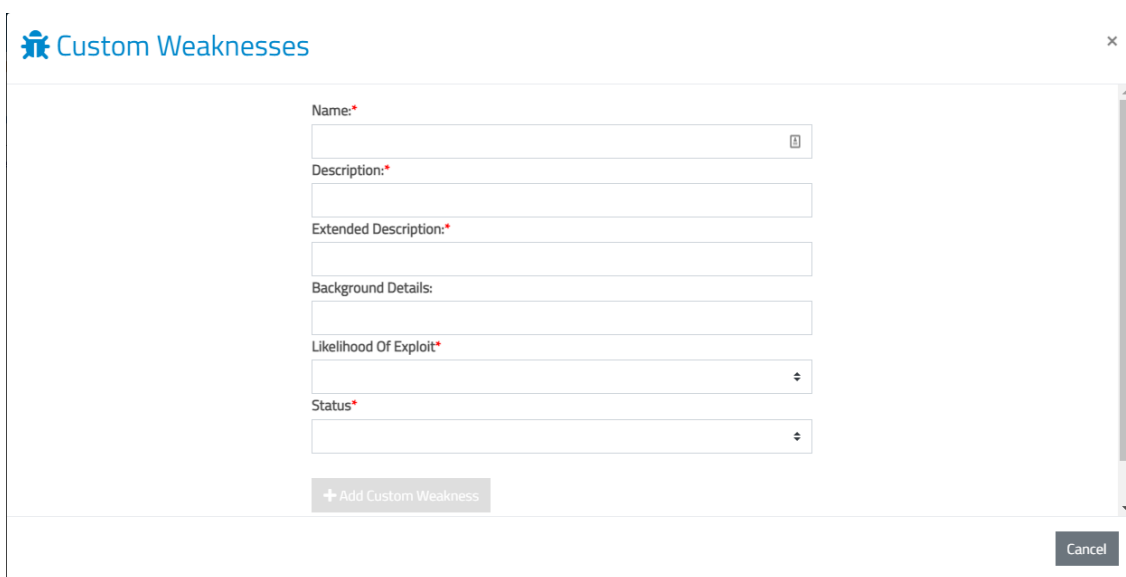
For the creation of Profiles and Models, ResilBlockly uses the functions of the Profile Designer and Model Designer respectively as shown and explained in Section 2.1 and Section 2.2 of D6.2 [2].

The D6.2 [2] provides a user guide for ResilBlockly and describes in detail all the features of the tool for threat modelling, hazard analysis, safety, and security risk assessment. The tool allows one to identify more critical components, functions, and interfaces that might cause great impact if compromised. Moreover, it supports the analysis with a graphical representation of the attack paths that adversaries typically follow to succeed in the exploit of the system or component. In addition, the tool complies with the Manufacturer Usage Description (MUD) [3] standard for specification of communication rules and extends the standard with a set of characteristics derived from the modelling and analysis and exports the resulting extended MUD file. Finally, thanks to the integration with a simulator, it also allows to simulate the interactions between components (e.g., under attack).

D6.2 [2] furthermore provides examples of use of ResilBlockly, showing how to realize or import profiles (meta-models) and models, to analyse components, functions and interfaces that possess weaknesses and are most vulnerable or exposed to the risk of

attacks, how to graphically represent the attack paths and patterns towards the exploitation of those weaknesses and how to conduct two complementary risk assessments (one HazOP-based, which is more safety-oriented, and the other leveraging the integration with online catalogues of threats to security, such as CWE, CVE, CAPEC, NVD, and scoring systems, such as CVSS).

The interface for the specification of custom weaknesses is shown in Figure 13; the one dedicated to custom vulnerabilities is analogous.



The screenshot shows a web interface titled "Custom Weaknesses". It contains the following fields and controls:

- Name:** A text input field with a small icon on the right.
- Description:** A text input field.
- Extended Description:** A text input field.
- Background Details:** A text input field.
- Likelihood Of Exploit:** A dropdown menu.
- Status:** A dropdown menu.
- + Add Custom Weakness:** A button at the bottom left.
- Cancel:** A button at the bottom right.

Figure 13. New interface for the specification of custom weaknesses

Finally, it is important to consider that the risk calculation performed by ResilBlockly in its current release addresses vulnerabilities and weaknesses in isolation; however, an already ongoing activity will provide the tool with a functionality for the specification of dependencies among assets and the computation of the risk of cascading effects initiated by threats as described in [4]. Moreover, as a future work, it is planned to consider approaches that allow to account for dependencies between vulnerabilities and weaknesses in terms of likelihood of successful exploitation, as an example, starting from the approach in [5] and elaborating over it.

3.1.1. AI Investments Use Case Modelling

The AI Investment (All) application is an innovative platform for optimizing financial portfolio management and risk control using the most advanced AI and deep learning solutions available on the market. It was created to improve the investment results of hedge funds and other trading and investing companies. The All application fetches data from various sources, combines and normalizes them, and then uses the latest achievements in time series forecasting and optimization to create the most suitable investment portfolio. Then, the necessary transactions are automatically executed using the selected stockbroker.

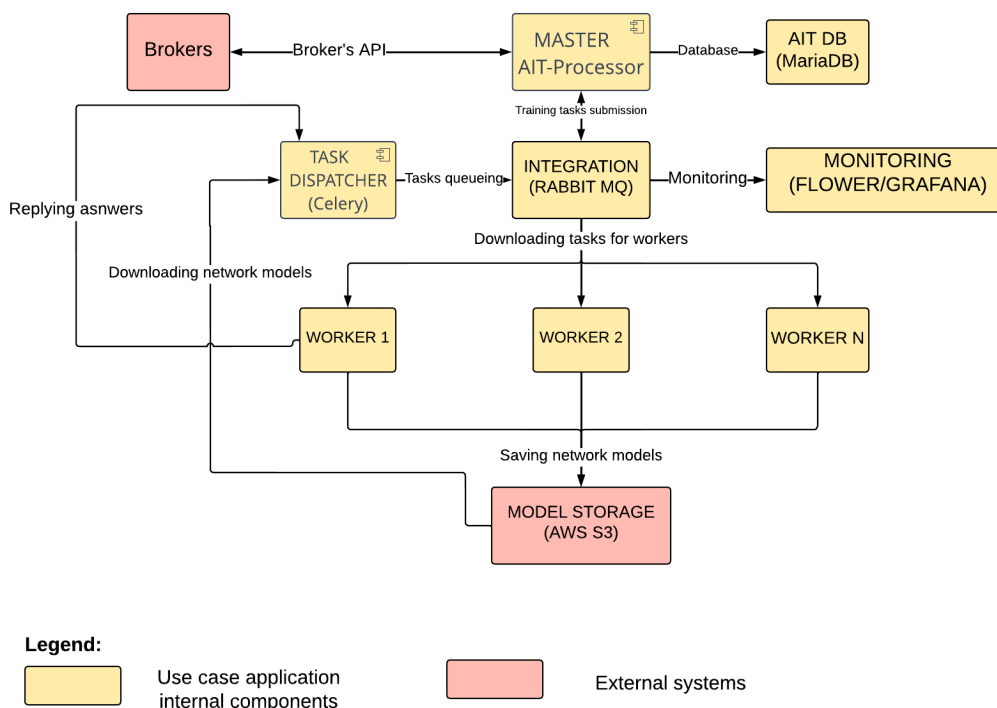


Figure 14. All Architecture Diagram

The All use case architecture, presented in Figure 14, was modeled with ResilBlockly Modeler, showing all the communication in each direction visible in the diagram. We created a single CS (Computer System) containing APIs (RUMIs) exchanging messages between all communicated entities. Moreover, each of the entities (Master AIT Processor, Integration, AIT DB, Worker, etc.) was modeled as a separate system with its own set of services and internal APIs (here modeled as RUMIs). The model is relatively complex, therefore only certain parts are discussed below and most of the fields (parameters) of classes presented in the figures are hidden for readability.

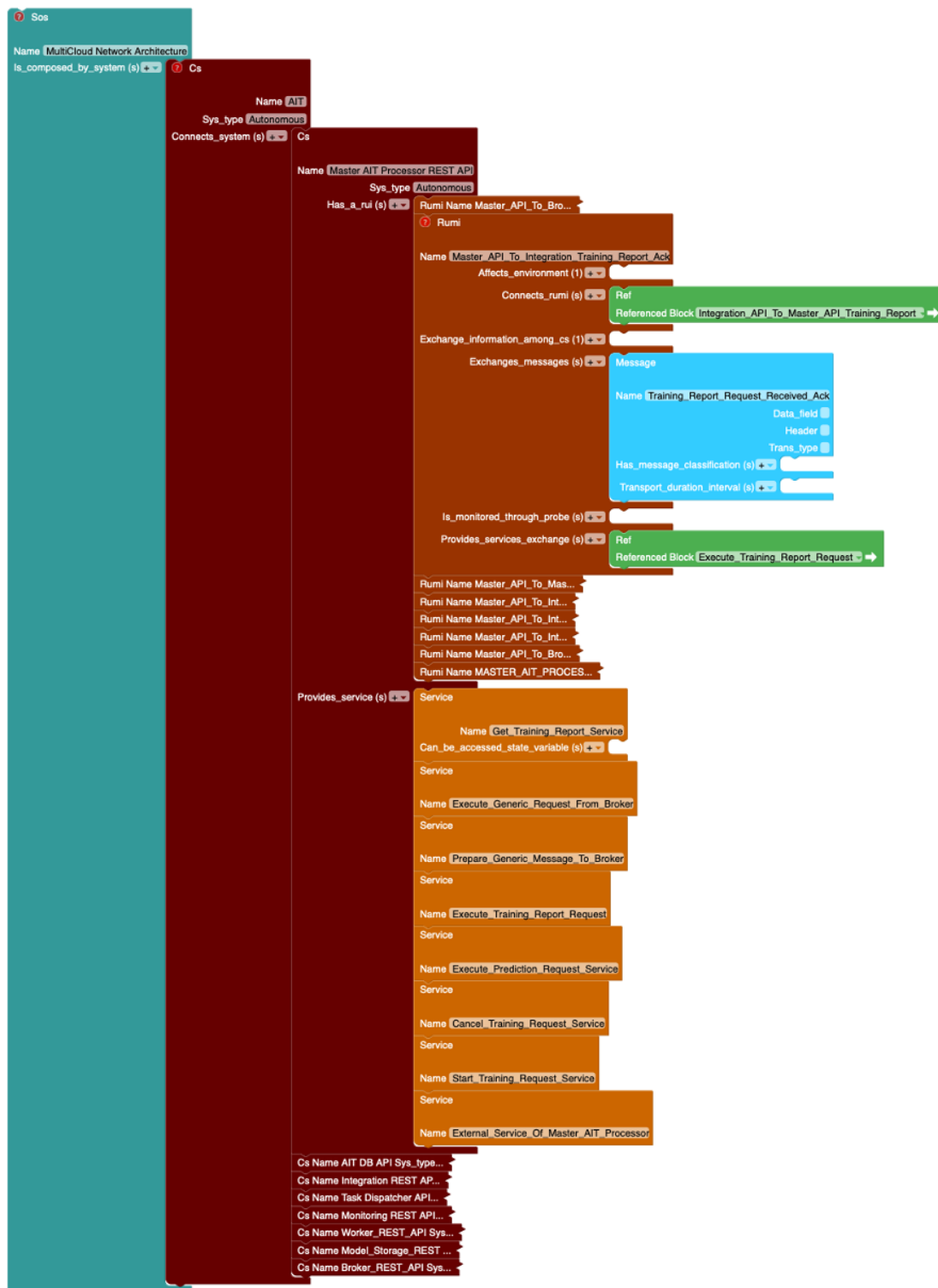


Figure 15. Simplified model of the All communication system implemented in Resilblockly

Figure 15 shows the whole All communication system implemented in ResilBlockly. Most of the boxes are collapsed to fit the model into the deliverable, but the point of the figure is to show the general style of the model; the relations can be easily analysed using Figure 14. Below is the explanation of the model presented in figure 15.

A general SoS (System of Systems, leftmost rectangle) class contains one CS (Computer System, big red rectangle) class: AIT. This CS, in turn, connects a variety of CS, i.e., the entities from figure 14, for example Master AIT Processor REST API, Monitoring API, Worker API, or AIT DB API (smaller red rectangles).

Master AIT Processor REST API box is expanded in Figure 15 to show RUIs (communication exchanges with other entities) and internal services, responsible for the actions behind each of the communications. RUIs include messages to Broker (e.g., Master_AIT_to_Broker_API), to Integration (Master_AIT_to_Integration_Training_Report_Ack) and to DB, as the architecture diagram in Figure 14 suggests. Moreover, there are additional “internal” communications from the Master AIT Processor API to the separate Master AIT Processor entity, which will be described later in this section.

Master AIT Processor REST API contains a list of services implementing its functionality, e.g., Execute_Training_Report_Request, Start_Training_Request_Service, etc.

As an example, Master_AIT_to_Integration_Training_Report_Ack RUMI is expanded. It is a confirmation of a received training report request from Integration API. Therefore, it contains:

- a reference to the Integration API’s RUMI which sends this request (green box with Integration_API_To_Master_API_Training_Report),
- a message (blue box with Training_Report_Request_Received_Ack),
- and a reference to a Master_AIT service implementing the functionality (green box with Execute_Training_Report_Request)

All other RUMIs in Master AIT Processor REST API are modelled in this way. Similarly, other CSs in AIT are modelled likewise. Moreover, all the classes contain more fields (parameters) that were not presented and discussed here. However, they have only more technical meaning and are not of a high importance to the Reader of this document.

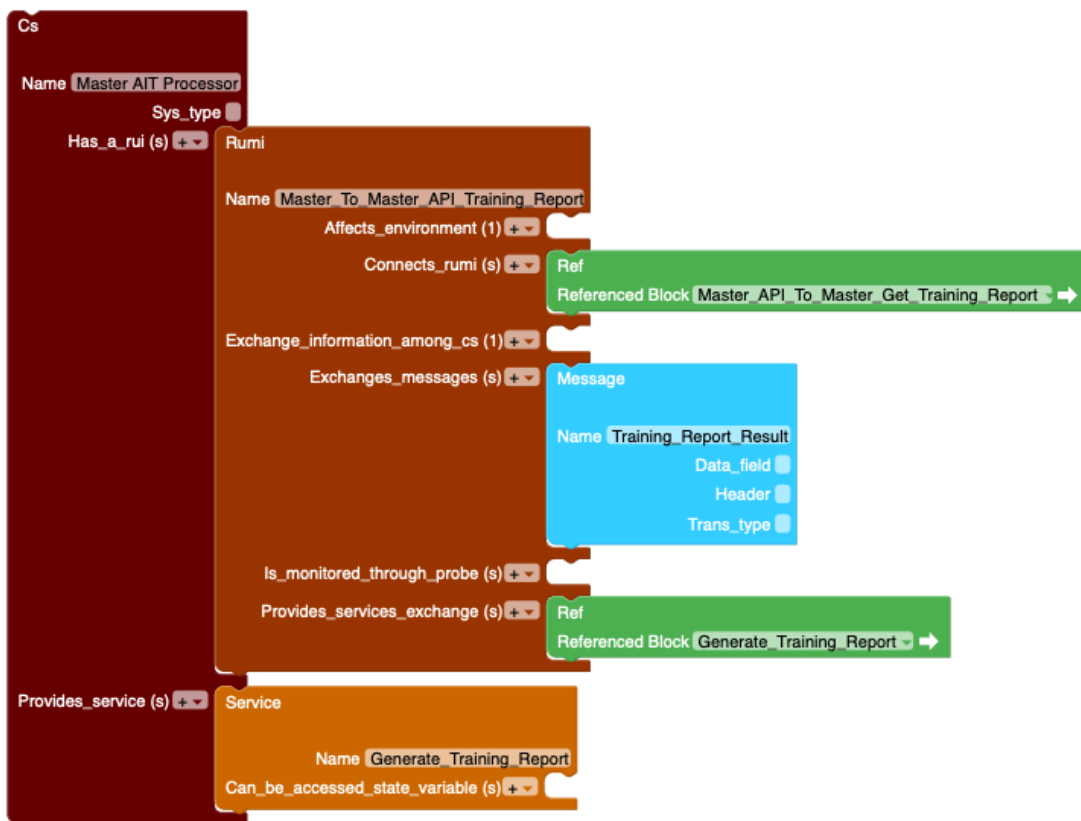


Figure 16. The internal Master AIT Processor entity in All Resilblockly model

While figure 15 presented the Master AIT Processor REST API as a part of the model of all All APIs, figure 16 shows the internal Master AIT Processor entity. The entity was simplified to contain only one RUMI and service for a sake of the readability of the deliverable.

One of the Master AIT Processor REST API functionalities requested by Integration (figure 15) was to generate the training report. To execute it, the Master AIT Processor REST API “calls” the internal Master AIT Processor entity. The call is represented by a green box Master_API_To_Master_Get_Training_Report reference in figure 16. In turn, Master AIT Processor contains a service Generate_Training_Report (orange box) and a RUMI Master_To_Master_API_Training_Report which “responds” with the Training_Report_Result message (blue box) to the REST API.

Other RUMIs and services in Master AIT Processor are implemented similarly, as well as the other internal entities of the All Model in Resilblockly.

3.1.1.1. Risk Analysis

After creating the All model in Resilblockly, the risk assessment analysis was performed. Each of the entities in the model was analysed and associated with a list of weaknesses, imported from the CWE or CAPEC databases, or, in rare cases, custom-made weaknesses. Similarly, the vulnerabilities for each of the entities in the model were

analysed and then associated with the vulnerabilities imported from the CVE database or custom-made.

Then, the severity and likelihood were added (or verified, if the values were imported from the public databases) for the vulnerabilities and weaknesses in each of the entities. At the end of the process, Resilblockly generated the weaknesses and vulnerabilities reports for the whole All model.

3.2. IFEVS Use Case Modelling

3.2.1. Microfactory and Cloud IoT

As already presented in D2.1, I-FEVS proposes the Microfactory concept to deal with the assembly of high-performing full-electric vehicles, with high variants and high quality, on low-cost assembly lines. Significant flexibility is needed to process all variants on the same work-area. The production lots can vary from a few units, in case of vehicles transporting temperature-controlled goods such as pharmaceutical products, to tens, for special freight delivery vans, up to several hundreds, for passenger vehicles.

The low upfront investments necessary for the typology of vehicles addressed in the project (large variants and small lots) shortens the time for breakeven.

The Microfactory and all its supply-chain represent a cyber-physical environment in an automotive industrial framework; the need to protect the communications among the various areas of the production against cyber-attacks is of paramount importance and has been considered from the beginning, starting with the remote communication with the vehicle.

The whole system, within the Microfactory and its supply-chain, is basically made of cloud nodes and these lead to the idea to extend the same approach, adopted for the remote communication with the vehicle and its onboard network, not only to entire fleets of vehicles, but also to the data flow within the shopfloor, the headquarters, the R&D department, the maintenance services, and the suppliers.

Thus, the IFEVS use case can be represented by the remote Firmware (FW) update service, usually indicated as FOTA (FW Over the Air) without losing details or neglecting relevant technical aspects. The E-E architecture on-board the vehicle, on its turn, relies on zone partitioning, where different ECUs (Electronic Control Units), implemented with identical HW, handle different sets of tasks according to the specific FW uploaded and running on them; the vehicle is then communicating remotely through a gateway with the OEM side.

To implement the secure communication system for remote updates, the UPTANE guidelines, a de facto security standard in the automotive sector, have been chosen.

3.2.2. FOTA and UPTANE

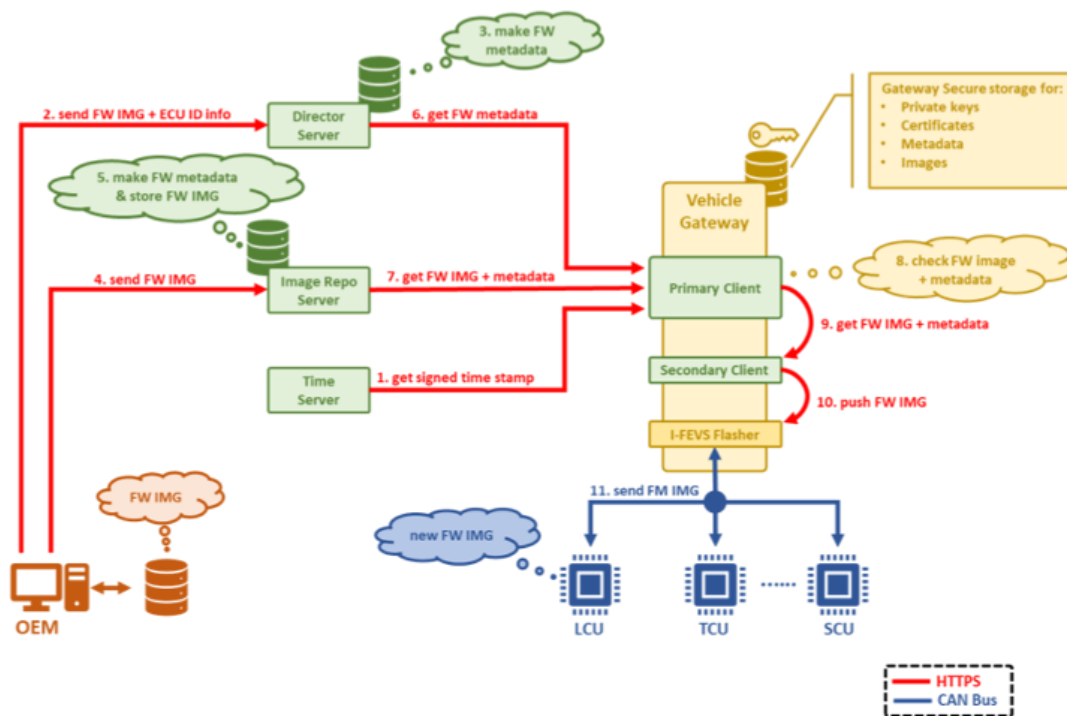


Figure 16. UPTANE-like architecture of the FOTA system

The UPTANE network implemented herein, has been constantly adapted and upgraded to meet the requirements of I-FEVS electric vehicles.

The UPTANE network can be distinguished in the FW supplier, server and client sides. The server side consists of the Director, Image Repo and Time servers while the client side consists of the Primary client and a variable number of Secondary clients. In a real scenario the three servers run in a workstation and the clients in the vehicle gateway.

The following provides a brief description of the role of each entity:

1. **Firmware Supplier** - it develops the FW to be uploaded into a specific ECU. It produces also metadata containing information related to the firmware version, to the ECU it refers to and to the vehicle associated to the specific ECU.
2. **Director server** - It manages the update procedure and registers vehicles and ECUs. It uses metadata files to keep track of the available firmware images, their current version available for update, the ECU of the vehicle that it refers to, the hash and size of the firmware image. Additionally, it keeps track of which version of firmware is currently installed in each ECU.
3. **Image Repo server** - It is where the images are stored. A vehicle will request firmware images from this server. It also uses metadata that, among other uses, informs about the hashes and size of each available firmware image.
4. **Time server**. It is used to provide signed timestamps which also carry a nonce value generated by the clients. A client sends the nonce to the Time server requesting a timestamp. The Time server generates a timestamp which is signed along with the nonce value and then sent back to the client. The client uses this trusted timestamp to make sure that the metadata and images that will, later, be acquired by the Director and Image Repo servers, are not expired.

5. **Primary client.** It is the frontline of the vehicle's gateway. It is responsible for receiving firmware update images on behalf of Secondary clients. In order to do so, it acquires metadata from both the Director and Image Repo servers. It checks the metadata about available images for the ECUs. If both servers agree that there is a new firmware image then the Primary client compares their metadata to, also, make sure that they agree upon the hashes and size of the file to download. If the integrity check succeeds, then the Primary client proceeds with downloading the firmware image from the Image Repo server. The firmware image is kept in the Primary client for as long as it needs before a Secondary client asks for it.
6. **Secondary client.** A Secondary client is responsible for receiving a firmware image on behalf of an ECU of the vehicle. There should be as many Secondary clients as the number of ECUs of the vehicle that need to check for updates. Each Secondary client requests metadata and images from the Primary client. It uses those metadata to validate the integrity of the firmware image prior to storing it locally as accepted. An implementation such as the I-FEVS ECU flasher can then forward the trusted firmware image to the respective ECU through CAN.

3.2.3. FOTA-UPTANE Modelling

The whole network has been modelled with ResilBlockly, representing all subsystems and their communication links. Given the focus of the project activities, the model describes all nodes-entities of the UPTANE architecture implementation at a high level, without detailing the single HW and SW features of components (e.g., microprocessor, memories or libraries and code).

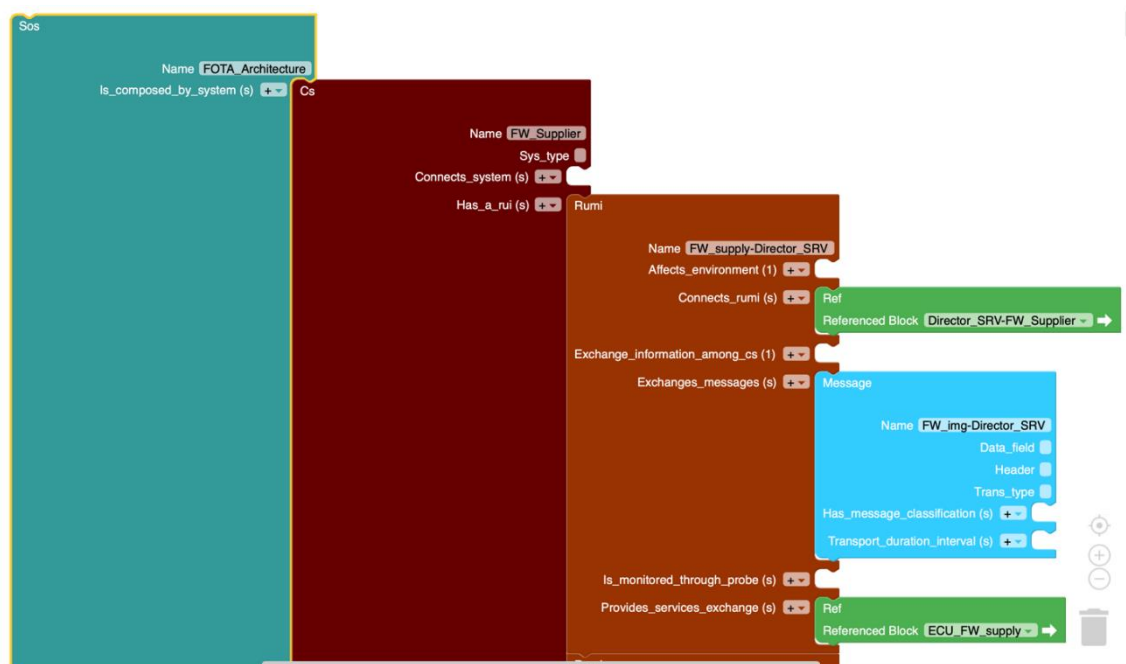


Figure 17. Example of representation of the UPTANE based FOTA architecture – the sketch shows the SoS, with the first CS and RUMI (with message description) in the model

A single SoS represents the FOTA service network, gathering a different CS for each entity operating in the architecture: FW supplier, Director, Image Repo and Time servers, Primary and Secondary clients. RUMIs are used to describe the various communication channels between the pair of interacting nodes with reference to each exchanged message.

The model built by blocks can be represented as a graph as well, to show all links and connections in the architecture.

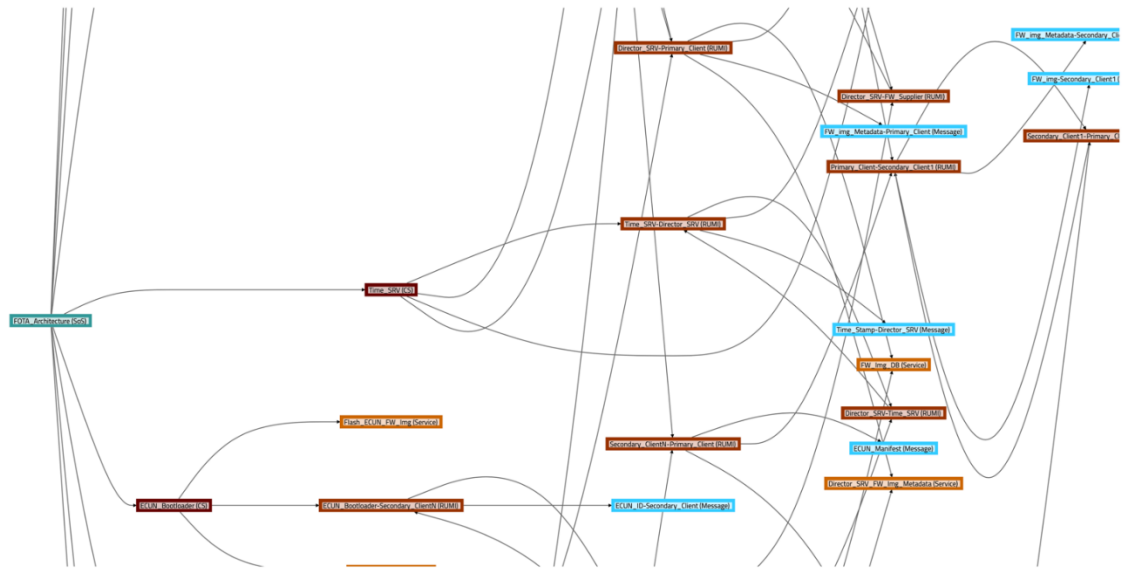


Figure 18. Graph representation of the UPTANE-like architecture of the FOTA system (the picture shows only a representative portion of the whole graph)

3.2.4. FOTA-UPTANE Risk Analysis

Modelling in Resilblockly enables the use of the risk assessment tool. All model components can be analysed and associated with a list of weaknesses and vulnerabilities, as described in Deliverable D6.2. Once the list has been built, each identified weakness and vulnerability is associated with its corresponding impact severity and the related likelihood.

The above-mentioned elements are then taken as the basis to generate the risk analysis report. These steps are being carried out and the report will be generated afterward.

4. Supporting Rating of Safety-Related Impact of Vulnerabilities

In this section, we provide more details on the interoperability between the ResilBlockly and safeTbox tools. While the concept of integrated dependability (focusing on safety and security) has already been touched upon in BIECO deliverable D6.4, the aim of this section is to highlight how the technical process has been designed and implemented, and how users can use both tools to arrive at appropriate mitigations for overall dependability.

ResilBlockly supports an export mechanism based on the Eclipse Modelling Framework (EMF) metamodel¹² aka 'Ecore'. This allows users to export ResilBlockly models to a common file format, which can then be used by other tools.

SafeTbox has been extended to support importing such models from ResilBlockly, a process which is performed in two stages. In the first stage, safeTbox converts the subject file from the generic Ecore-based format into an Open Dependability Exchange (ODE) format¹³. This is implemented prototypically as a text-to-model, then model-to-model, then model-to-text transformation sequence. The reason for this design choice is to exploit already existing support for the ODE in safeTbox. Future work could aim towards simplifying and optimizing the transformation. In the second stage, the now ODE-formatted file can now be imported in safeTbox using existing import mechanisms that support the ODE format.

Once modelling in safeTbox is complete, the updated models can be re-exported for further use. ResilBlockly also supports re-importing the exported safeTbox model files in a reverse flow, allowing, for instance, safety-related risk assurance information to be used in the ResilBlockly tool and other tools downstream.

The overall workflow is represented in Figure 19, where a user first models their system(s) in ResilBlockly, alongside relevant security risk parameters e.g., weaknesses and vulnerabilities. Once ready, the model is exported from ResilBlockly into a file on the local system. The user can then launch safeTbox, import the model file, perform modelling tasks in the tool, and export an updated model file. Finally, the user can import the file back into ResilBlockly and continue their work there.

¹² <https://www.eclipse.org/modeling/emf/>

¹³ <https://github.com/Digital-Dependability-Identities/ODE>

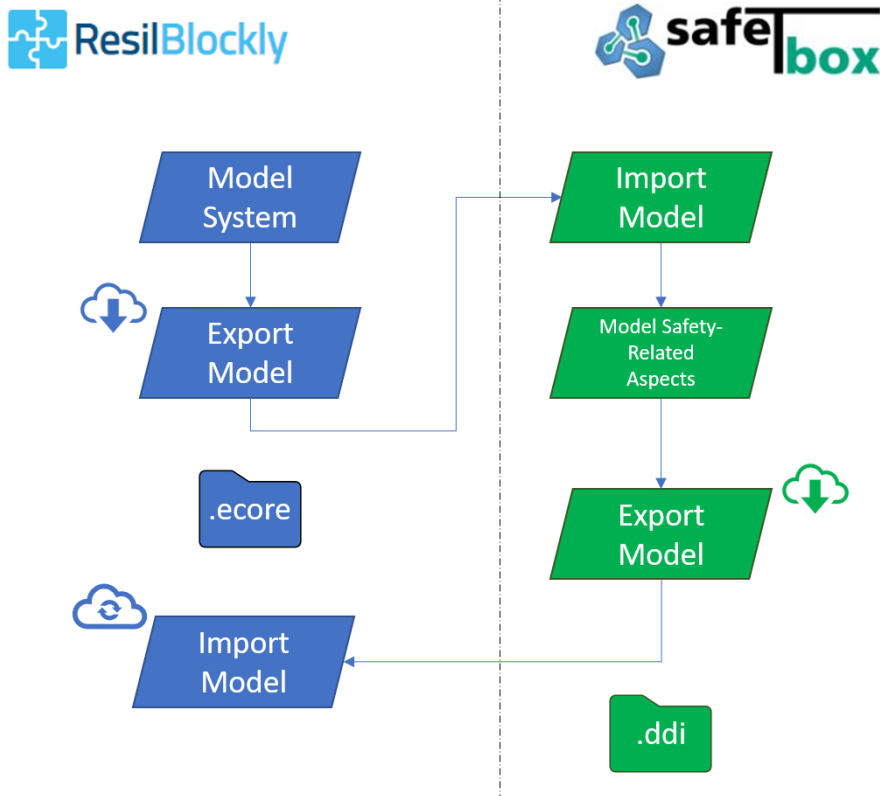


Figure 19. ResilBlockly-SafeTbox Workflow

As an example of the above workflow, starting with ResilBlockly at the point where modelling has been completed (see BIECO D6.2 for detailed user guidance), use the export function (see Figure 20) to store the model in a file on the local storage.



Figure 20. Export from ResilBlockly

This should generate a file with a '.ecore' file format extension. In safeTbox, either open a new project or load an existing one, and then use the 'Smart Menu' (Ctrl+Space shortcut) then navigate to the 'Comfort' submenu, then choose 'Import DDI from file...' option (see Figure 21).

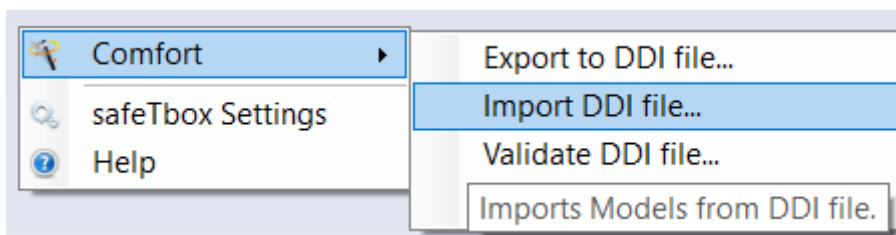


Figure 21. Importing into safeTbox

A 'choose file' dialog will appear, **it is important to change the filter to '.ecore'**, as seen in Figure 22. The dialog can then be used to select the target model file.

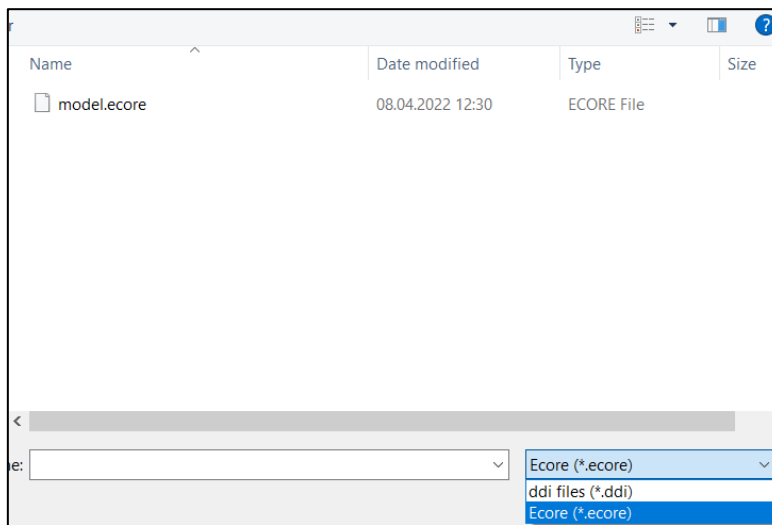


Figure 22 - Changing the file format filter to .ecore

Once modelling in safeTbox is complete (see BIECO deliverable D6.4 and/or the safeTbox user manual¹⁴ for further guidance), navigate to the same set of menus, and use the 'Export to DDI file...' option to store a file of the model in the local storage.

A specific safeTbox modelling feature that has been extended for BIECO supports specifying a safety impact rating for (security claims of) imported weaknesses and vulnerabilities. An example of how this appears in safeTbox, after a model has been imported, can be seen in Figure 23. At the top of the figure, the Smart Menu is used to select the option 'safeTbox Properties', and the bottom part of the figure shows the properties dialog with which corresponding safety impact ratings can be specified (bottom-right). Guidance on the conceptual aspects of this workflow can also be found in BIECO deliverable D7.3.

¹⁴ <https://safetbox.de/docu-samples>

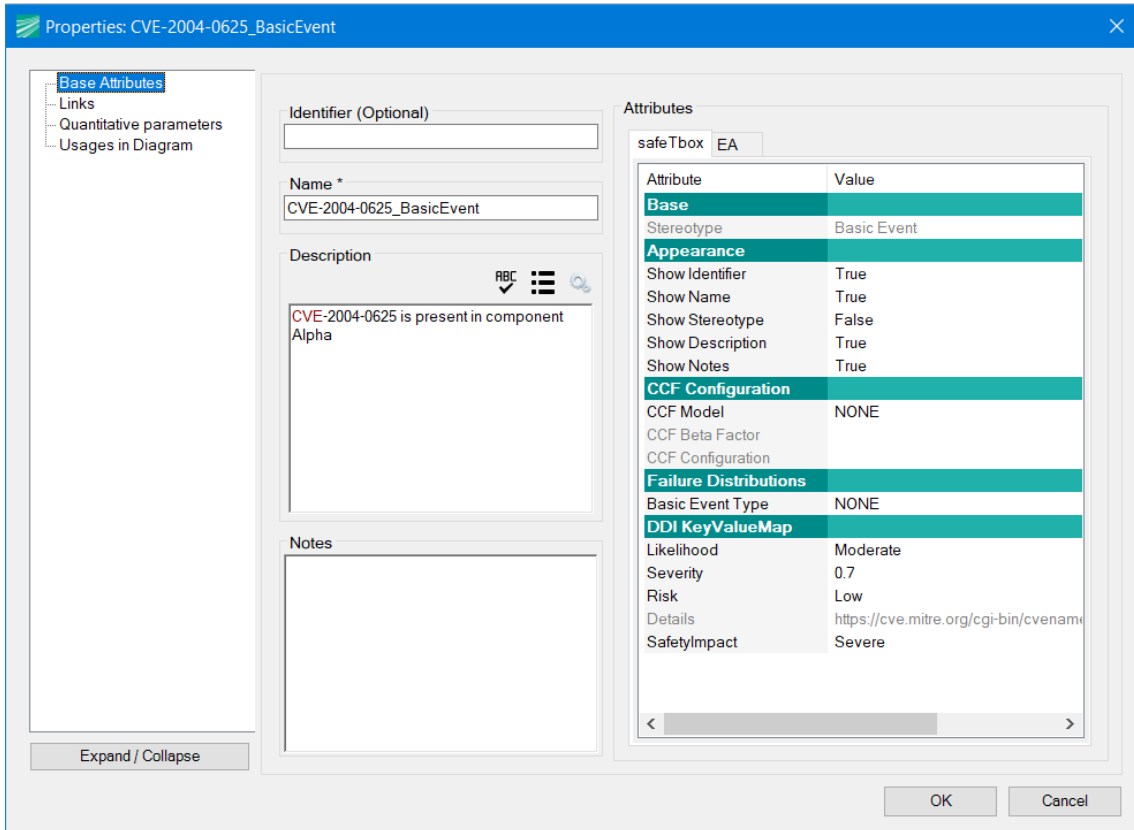
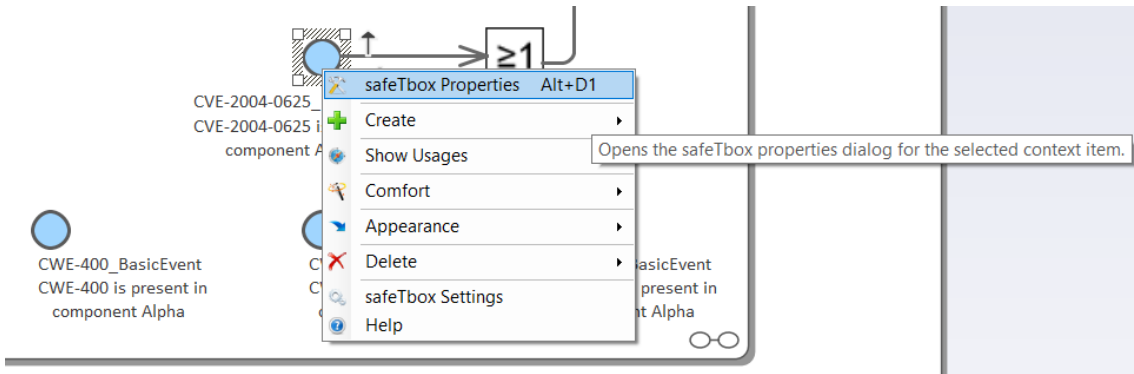


Figure 23. Accessing the properties of an element in safeTbox

5. Definition of the Threat MUD for Sharing Mitigations

In a hyper-connected society, in which humans and devices compose complex interconnected systems, we experience a strong cybersecurity interdependence. In this scenario, the final network becomes much more complex and heterogeneous and therefore it can be much more likely for a vulnerability to affect many more systems and to be propagated very quickly. Due to the borderless nature of the infrastructures and threats involved, any vulnerability or security incident in one country can have catastrophic implications throughout the European Union.

Only in 2021, more than 20,000 vulnerabilities were detected [13]. The fact is that manufacturers cannot quickly deal with new discoveries, since the release of a patch or an update is usually a slow process, especially when the system is composed by third-party components. In this sense, security-information-sharing systems propose an efficient, fast, and collaborative way of sharing recently discovered vulnerabilities or attacks to react in time before a patch is released. The US National Institute of Standards and Technology (NIST) proposed a threat signalling approach using a threat Manufacturer Usage Description (MUD) [14]. The threat MUD is based on the MUD standard for network behavioural specification [15], and it is intended to be structured similarly. Unlike the MUD standard, which was described in D6.1, the threat MUD is a new concept designed as a mitigation mechanism. However, the NIST only gives some indications about the threat MUD model and its similarity with the MUD standard model and therefore, the threat MUD model is still unclear.

Next subsections define the threat MUD model and its architecture, taking the NIST guidelines and the MUD standard as a starting point. In addition, two possible usages of the threat MUD file are proposed, one for sharing encountered threats and one for obtaining information about compromised domains and possible mitigations.

5.1. Threat MUD Model

Although this threat MUD has a structure similar to the regular MUD format, it is intended to serve as a mitigation mechanism, listing external sites to and from which traffic should be restricted due to their association with a specific threat. Therefore, it is not within the scope of the threat MUD to provide a list of sites with which access should be permitted, nor to establish any rules for local network traffic. As a result, rather than being developed by the manufacturer, the threat MUD is supposed to be created by a threat intelligence provider.

Figure 24 shows the first module of the threat MUD. This module has been generated from the standard MUD model (already described in D6.1) and the NIST indications with some variations, and includes the following fields:

- **threat-mud-version**, previously named `mud-version`, indicates the current version of the threat MUD file. It can be used to get the last update of the threat MUD or compare different threat MUD files.
- **threat-mud-url**, previously named `mud-url`, indicates the URL associated with the current threat MUD file. This URL can be used to retrieve the file.
- **last update**, indicates the date of the last update of the threat MUD file, which can be used to get the last version of the file.

- **threat-mud-signature**, previously named mud-signature, indicates the URL in which the signature of the file is located. As in the MUD standard, this signature is used to check the integrity of the threat MUD file.
- **cache-validity**, indicates the frequency in hours to check for an update of the current file. This field is especially relevant in the threat MUD, as an updated threat MUD can be created in a short time if new domains are known to be compromised by the associated threat, and therefore, having the last update of the file is crucial to implement the countermeasures.
- **is-supported** indicates if the threat associated with the threat MUD file is currently being addressed by the involved manufacturers.
- **threat-intelligence-provider** substitutes the mfg-name field, which described the manufacturer name, and now identifies the threat intelligence provider that detected and alerted about the threat.
- **threat-name** substitutes the model-name field and identifies the threat associated with the threat MUD file.
- **cvss-vector** is a new field that indicates the severity of the threat in terms of the Common Vulnerability Scoring System (CVSS) standard [10]. In particular, the CVSS score is represented as a vector string, a compressed textual representation of the values used to derive the score.
- **documentation** points to a URL in which additional information about the threat can be found e.g., a link to the National Vulnerability Database (NVD)¹ entry.
- **extensions**, as in the MUD standard, is reserved for future extensions of the threat MUD model.
- **from-device-policy** and **to-device-policy**, indicate the name of the Access Control Lists (ACL) that should be applied to mitigate the threat. These ACLs are further detailed in the next module of the threat MUD.
- **system-info**, **firmware-rev** and **software-rev** fields have been removed from the threat MUD model, as they are associated with a specific device.

```

1  module: ietf-threat-mud
2    +--rw threatmud!
3      +--rw threat-mud-version
4      +--rw threat-mud-url
5      +--rw last-update
6      +--rw threat-mud-signature?
7      +--rw cache-validity?
8      +--rw is-supported
9      +--rw threat-intelligence-provider
10     +--rw threat-name
11     +--rw cvss-vector?
12     +--rw documentation?
13     +--rw extensions*?
14     +--rw from-device-policy
15     +--rw to-device-policy

```

Figure 24. Threat MUD module

Figure 25 shows the second module of the threat MUD, which integrates the ACLs that could be applied to mitigate the associated threat in terms of network access control. The module is like the MUD standard model. However, as the threat MUD is associated

with a specific threat, not with a device and the configuration to apply should be as generic as possible, some fields have been removed from the `ace/matches/mud` section, in particular, `same-manufacturer`, `local-networks`, `controller` and `my-controller`:

- **same-manufacturer** always had a null value to indicate devices from the same manufacturer specified in the `mfg-name` field from the previous module. However, this field was substituted by the intelligence provider.
- **local networks** were used to remove or allow access to the whole local network of a particular device. As the threat MUD should be applicable to any kind of device, this field has been removed.
- **controller** and **my-controller** were used to indicate the generic controller of a device. As before, this field is not generic enough to be included in the threat MUD.

```

1  module: ietf-access-control-list
2      +--rw acls
3          +--rw acl* [name]
4              +--rw als
5                  +--rw access-list* [name]
6                      +--rw name
7                      +--rw type?
8                      +--rw aces
9                          +--rw ace* [name]
10                             +--rw name
11                             +--rw matches
12                                 +--rw mud
13                                     | +--rw manufacturer?
14                                     | +--rw model?
15                                     +--rw eth?
16                                     +--rw ipv4?
17                                     +--rw ipv6?
18                                     +--rw tcp?
19                                     | +--rw direction-initiated?
20                                     +--rw udp?
21                                     +--rw icmp?
22                                     +--rw egress-interfe?
23                                     +--rw ingress-interfe?
24                             +--rw actions
25      +--rw attachment-points

```

Figure 25. ACL module

5.2. Threat MUD architecture

A particular architecture to obtain the threat MUD was proposed by the NIST in [16] to combine the usage of the MUD standard and the threat MUD. Figure 26 shows a generalization of that build including additional components for threat management.

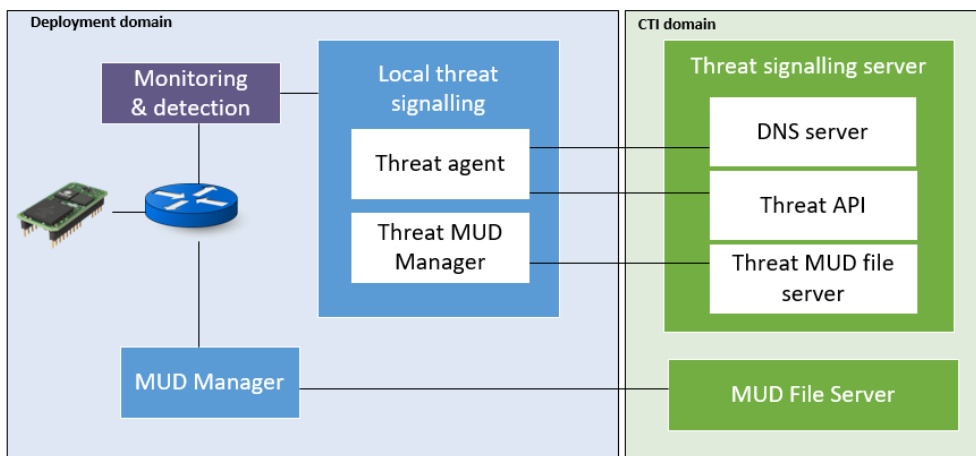


Figure 26. Threat MUD and MUD architecture

The following entities are envisioned:

- The device operating in the network. It is responsible for sending the MUD URL to the switch for obtaining the MUD.
- The router or switch responsible to forward or restrict the device traffic.
- The MUD Manager is the main entity of the MUD architecture. It will oversee asking for the MUD file to the MUD File Manager using the MUD URL, parsing and enforcing the policies.
- The MUD file server, located in the manufacturer domain, stores all the MUD files from devices of a certain manufacturer.
- The Threat Agent monitors the DNS traffic from and to the device to detect when a DNS request is not solved. When a domain is suspicious of being compromised, that is, a DNS request returned a null value, it asks for confirmation to the Threat API and alerts the Threat MUD Manager about this domain. Moreover, it also receives information from the monitoring and detection entity about possible threats.
- The DNS service, which receives information from threat intelligence providers about a compromised domain. In case the domain is marked as compromised, it returns a null value.
- Threat API receives requests from the Threat Agent to verify whether an unresolved domain is compromised. Moreover, it gives information about the Threat Intelligence provider that identified a compromised domain.
- The Threat MUD Manager, analogous to the MUD Manager, queries the Threat MUD file Server for the threat MUD file and signature. In addition, the Threat MUD Manager enforces the filtering rules in the router. It's worth mentioning that the Threat MUD connected with a threat will list all of the domains that are affected by the threat, as well as the filtering rules that will block access to them. The threat MUD Manager is also responsible for creating a threat MUD file in case a new threat without an existing threat MUD file is detected.
- The threat MUD file server job will consist of storing and delivering Threat MUD files associated with a compromised domain (and threat).

- The Monitoring and detection entity is in charge of monitoring the device communications and alerting the threat agent about possible threats.

5.3. Usages of the threat MUD

Sharing discovered threats (Figure 27): The threat MUD is integrated to share encountered threats and mitigation with interested stakeholders. In this context, the monitoring and detection entity in charge of monitoring the device communications (step 1) detects a new threat and alerts the local threat signalling service, specifically the threat Agent (step 2). The threat agent validates if this threat was already discovered by asking the threat API (step 3). If the threat API replies that the threat is unknown (step 4), the threat Agent will request the threat MUD manager the creation of a threat MUD associated with the encountered threat, indicating the compromised domains identified by the monitoring and detection entity (step 5). Once the threat MUD file is created, the threat MUD manager will post it on the threat MUD file server (step 6). If accepted, the threat MUD file server will acknowledge it (step 7) and request an update of the threat API database (step 8). In this way, other domains that may be affected by this threat will be able to have access to this information, apply the pertinent mitigations and collaborate in the construction of the new threat MUD file.

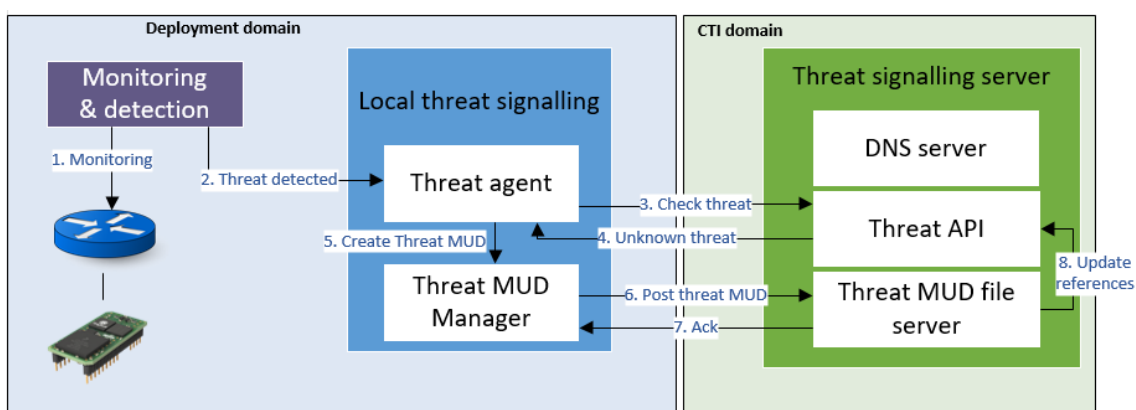


Figure 27. Sharing discovered threats

Enforcing mitigations (Figure 28): This second usage depicted in Figure 28 shows how the threat MUD file can be used to deploy security policies to mitigate encountered vulnerabilities. In this case, the detection of compromised domains is performed through the DNS service. The device will eventually make a DNS request to access a certain domain (step 1). The router or switch is responsible to forward the DNS request to the Threat Agent (step 2) and the DNS server (step 3). The DNS service, which receives information from threat intelligence providers about compromised domains will answer the DNS query of the device. In case the domain is marked as compromised, it will return a null value (step 4). The threat Agent, which is monitoring DNS traffic from/to devices to DNS server, will detect a NULL DNS answer, and it will ask for confirmation to the Threat API (step 5). If the threat API confirms that the domain is compromised (step 6), the threat Agent will alert the Threat MUD Manager about this domain to obtain the threat MUD file (step 7). The threat MUD Manager will ask for the associated Threat MUD file

(and its signature) to the threat MUD file server (steps 8 and 9). Finally, the threat MUD Manager will translate and enforce the threat MUD policies in the switch (step 10).

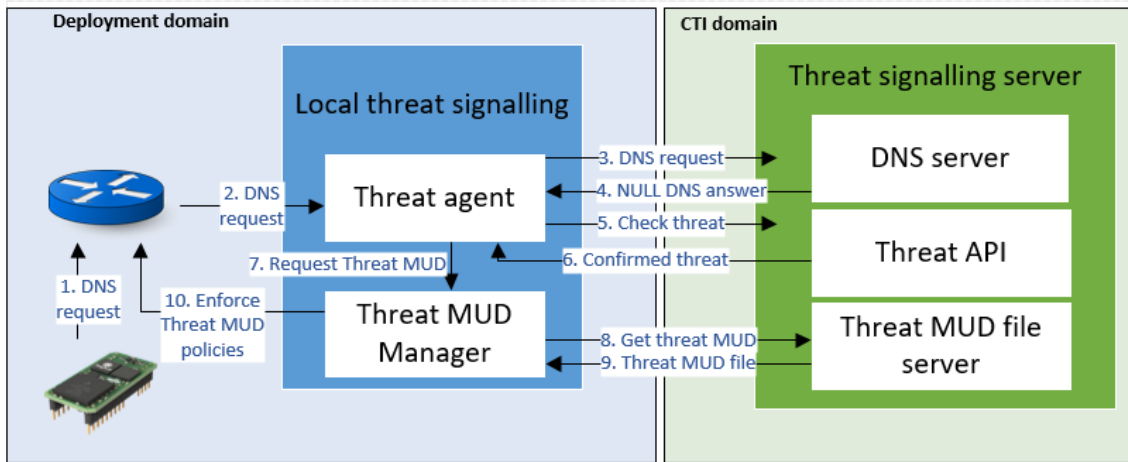


Figure 28. Enforce mitigation

6. GDPR-Based Accountability and Auditability of Authorization Systems

The GDPR is an EU regulation that came into effect in May 2018 and imposes strict requirements on the collection, use, and storage of personal data. Inside the BIECO project, this section focuses on the notion of **Accountability** from the point of view of the GDPR, which is the 7th principle defined in Art. 5 “Principles relating to processing of personal data”, paragraph 2, which words:

“The controller shall be responsible for, and be able to demonstrate compliance with, paragraph 1 (‘accountability’).”

Paragraph 1 contains the first six principles. Among them, this work is focusing on the “integrity and confidentiality” principle, which states that “Personal data shall be: [...] processed in a manner that ensures appropriate security of the personal data, including protection against **unauthorised or unlawful processing** and against accidental loss, destruction or damage, using **appropriate technical** or organisational **measures** (‘integrity and confidentiality’).”

As extensively discussed in [6], by defining the Integrity and Confidentiality principle, the European legislator poses security at the heart of the GDPR, and implicitly calls for adopting Access Control Systems (ACSs). Indeed, ACSs are usually regulated by Access control Policies (ACPs), which specify who, what, when, where, how, and why (i.e., the 5W1H) a user is granted or denied access to a given asset. This information also includes Personal Data.

However, the security of processing is not an isolated obligation, but comes together with the GDPR’s Accountability principle (Art. 6.2). Indeed, according to this principle, security measures are at the same time an obligation and a technical means to implement other data protection obligations.

The BIECO’s solution relies on Access Control (AC) systems for guaranteeing compliance with the GDPR and it is based on the GENERAL_D (Gdpr ENforcement of peRsonAL Data) framework. It supports the integrated GDPR-based process development life cycle for the specification, deployment, and testing of adequate fine-grained authorization mechanisms able to consider legal requirements.

The GENERAL_D framework has the following objectives:

- **OBJ 1:** defining a GDPR-based Life Cycle for authorization systems. That means defining a specific and integrated process development life cycle for the specification, deployment, and testing of adequate fine-grained authorization mechanisms, by considering legal requirements.
- **OBJ 2:** providing an integrated environment for automatically enforcing the data protection or privacy regulations. Indeed, we define an integrated environment where we combine some of the available solutions for specifying the privacy requirements, controlling personal data, processing them, and demonstrating compliance with the GDPR in collecting, using, storing, disclosing, and disposing of the personal data lifecycle.

In work package WP7, GENERAL_D has been enhanced by the GROOT (Gdpr-based cOmbinatOriAl Testing strategy) tool [7], [8] for supporting the testing of GDPR-based access control policies derived directly by the BIECO claims (Task T7.1 and [9]). In Task 6.4, the GROOT enhancement is called I'M GROOT (Integrated environMent for Gdpr-based cOmbinatOriAl Testing). It is a tool under development able to organize and structure the testing data provided by GROOT to improve their testing effectiveness and make GDPR-based access control auditable for accountability purposes, as defined in the GDPR.

Indeed, GROOT produces a valuable amount of input and output data that can be hardly navigated or examined without automatic support. That information, if organized through data warehousing techniques, can be useful for accountability and auditing purposes.

For example, before deploying a given GDPR-based policy, answers to several questions can be obtained such as: Who can access a particular resource or personal data? What are the runtime processing activities allowed on a given personal data? What are the data that can be processed for a certain purpose?

Thanks to the solution proposed inside BIECO, the reply to these questions can be obtained before putting into production a certain GDPR-based policy. Thus, according to the principle of privacy-by-design, the protective discovery of the vulnerabilities can be performed. In addition, the mining of the collected test data can be exploited for building the users' behavioral models useful for improving the runtime monitoring activity (as described in deliverable D5.2 [17]). Indeed, the derived models, which represent the expected runtime behaviors, can be translated into proper monitoring rules and used to discover suspicious or wrong behaviors and promptly notify the security or privacy violations.

6.1. Contextualization of I'M GROOT in BIECO

Figure 29 reports the contextualization of I'M GROOT within the BIECO project, for supporting the testing, accountability, and auditability of GDPR-based access control systems ruled by ACPs derived from the Privacy Claims. As in the figure, different components and artifacts developed within BIECO, including GROOT, are involved.

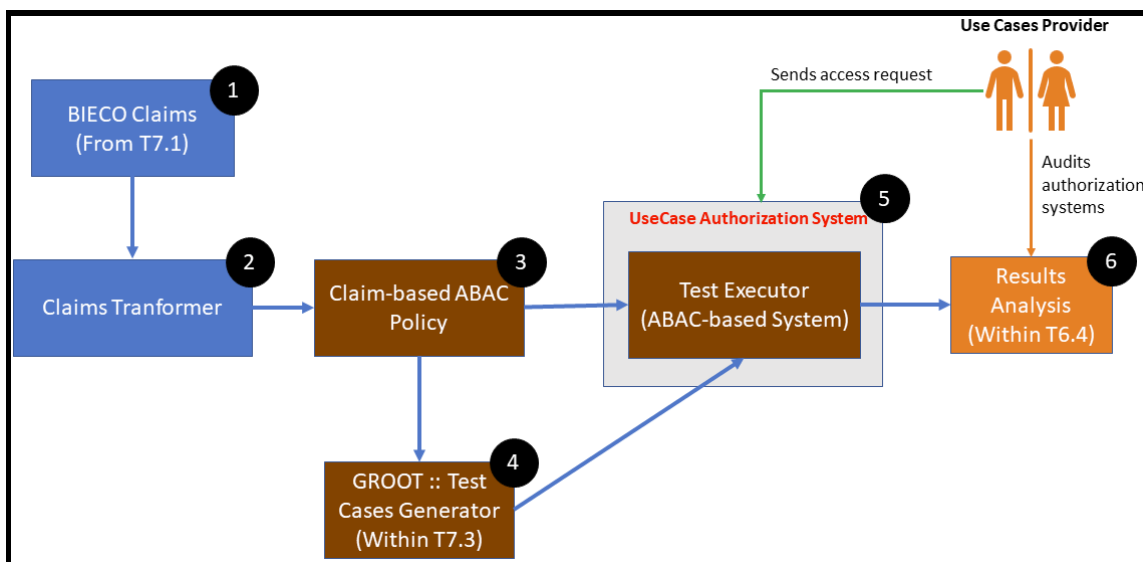


Figure 29. The Reference Process of I'M GROOT

In Deliverable D7.3, the contextualization of GROOT to the BIECO Claims collected in task T7.1 are reported. In Figure 29 this is represented by the BIECO Claims (component n. 1 in the figure). These claims are then translated into authorization policies through a Claim Transformer (component n. 2) that in BIECO are represented by access control policies expressed in ABAC (component n. 3-Claim-based ABAC Policy). As presented in deliverable D7.3, the GROOT tool uses the policy for deriving test cases (component n. 4),

In the I'M GROOT process, and differently from what is presented in deliverable D7.3, the policy and the generated access control requests (in the form of <policy, {requests}>) are delivered to an UseCase Authorization System (component n. 5) that through a Test Executor, is able to associate each request with the actual result.

The UseCase Authorization System component can be is also used during the runtime or operational phases, for evaluating and letting collecting the access control requests sent by the Use Case Provider and providing the proper authorization response, i.e., deny or permit the access according to the GDPR-based access control policy.

In this framework, the component in charge of the collection and analysis of the access control data (requests or responses) is the Results Analysis (component n. 6). This component has also the role of evaluating the accountability and compliance purposes.

6.2. Results Analysis Component Specification

As in Figure 29, the Result Analysis (component n.6) represents the orchestrator of the I'M GROOT proposal and it provides the interface for user interaction as detailed in Figure 30.

This component allows both policy tester and the use cases provider to perform the following main steps:

1. select an input GDPR-based policy and the associated test suite if this has been already derived; generate the GDPR-based requests and visualize their values

- (such as Data Sub, Personal Data, and Purpose) ordered by the identifier RequestID;
2. reduce the test suite using either the pairwise testing approach as in GROOT or the request/response values-based filter, specificity of I'M GROOT or both;
 3. execute the obtained requests on the selected GDPR-based policy;
 4. get a visualization of the test report, namely the values of the executed requests and the associated authorizations rights;
 5. filter the executed requests according to either their values, the associated authorization right, or a combination of them;
 6. validate that the combination of Data Subject, Personal Data, and the other categories' values of the request and the associated authorization right is consistent with the access control claim associated to the GDPR-based policy.

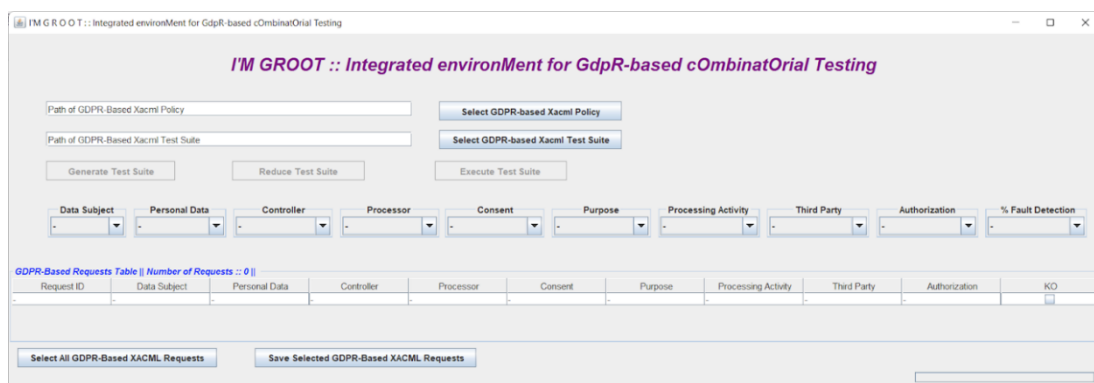


Figure 30. Graphical User Interface of I'M GROOT Framework

Thanks to the navigation and filtering facility I'M GROOT represents a valid help for the analysis of both the test results and the operational authorization requests and responses. Briefly, the main advantages of I'M GROOT can be listed as the following:

- it provides a comprehensive and manageable view of the executed data;
- it lets the analysis of the different combinations of requests values in order to discover violations or misinterpretations of the associated authorization rights;
- it allows the user to focus on specific behaviours associated with few access control constraints at a time;
- it lets an interactive validation of the results;
- it enables at the same time both auditability and accountability;
- it constructs an argument for demonstrating compliance with the GDPR.

7. Conclusions

In this deliverable, we have presented both new additions (non-repudiation of log modification) to the BIECO framework (LogForgeryBlocker), applications of tools already reported (ResilBlockly), as well as new views on the use of other tools to obtain further synergy in terms of risk assessment, security- (threat MUD), safety- (SafeTBox) and privacy-wise (I'M GROOT). The deliverable has looked both at static and dynamic aspects of threat discovery and analysis.

The new LogForgeryBlocker tool has been described in detail, covering the background, the technological aspects, as well as the user manual, and possible applications. Two use cases have been modelled in ResilBlockly to allow for in-depth threat analysis and the results have been reported in this deliverable. Finally, the various additional tools have been shown in context, underlining how they improve solving the risk assessment problem.

It is interesting to consider all the approaches working in tandem. At the moment, this is still in progress, but this deliverable has laid the ground for such a synergistic risk assessment strategy to be detailed and applied in the future.

8. References

- [1] E. Schiavone (edt.) et al. (2021, June) "Blockly4SoS Model and Simulator", Deliverable D6.1 of the BIECO project funded under the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 952702.
- [2] E. Schiavone (edt.) et al. (2021, June) "Blockly4SoS User Guide", Deliverable D6.2 of the BIECO project funded under the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 952702.
- [3] E. Lear, D. Romascanu, and R. Droms, "Manufacturer Usage Description Specification (RFC 8520)," 2019. [Online]. Available: <https://tools.ietf.org/html/rfc8520>
- [4] Schiavone, E., Nostro, N., & Brancati, F. (2021, November). A MDE Tool for Security Risk Assessment of Enterprises. In Anais Estendidos do X Latin-American Symposium on Dependable Computing (pp. 5-7). SBC.
- [5] Aksu, M. U., Dilek, M. H., Tatli, E. İ., Bicakci, K., Dirik, H. I., Demirezen, M. U., & Aykır, T. (2017, October). A quantitative CVSS-based cyber security risk assessment methodology for IT systems. In 2017 International Carnahan Conference on Security Technology (ICCST) (pp. 1-8). IEEE.
- [6] Said Daoudagh. "The GDPR Compliance Through Access Control Systems" [Ph.D. Thesis, University of Pisa], ETD Archivio digitale delle tesi discusse presso l'Università di Pisa, 20 July 2021) <https://etd.adm.unipi.it/t/etd-07112021-124810/> and <https://publications.cnr.it/doc/461891>
- [7] 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
- [8] UMU, 7Bulls et Al., Deliverable D7.3. "Deliverable 7.3 Security certification methodology development" 2022
- [9] UMU et Al., Deliverable D7.1. "Deliverable 7.1 Report on the identified security and privacy metrics and security claims to evaluate the security of a system" 2021
- [10] UMU et Al., Deliverable D7.2. "Deliverable 7.2 Security certification methodology definition" 2021
- [11] RESILTECH et Al., Deliverable D6.1. " Blockly4SoS Model and Simulator" 2021
- [12] RESILTECH et Al., Deliverable D6.1. " Blockly4SoS User Guide" 2021
- [13] "CVE vulnerabilities by date," May 2022, [Online; accessed 25. May 2022]. [Online]. Available: <https://www.cvedetails.com/browse-by-date.php>
- [14] NIST, "Securing Small-Business and Home Internet of Things Devices: NIST SP 1800-15," 2019
- [15] E. Lear, D. Romascanu, and R. Droms, "Manufacturer Usage Description Specification (RFC 8520)," 2019. [Online]. Available: <https://tools.ietf.org/html/rfc8520>
- [16] NIST, "Securing Small-Business and Home Internet of Things Devices: NIST SP 1800-15," 2019.
- [17] W. Penard, T. van Werkhoven, "On the Secure Hash Algorithm Family", https://web.archive.org/web/20160330153520/https://www.staff.science.uu.nl/~werk1108/docs/study/Y5_07_08/infocry/project/Cryp08.pdf
- [18] Narayanan, Arvind; Bonneau, Joseph; Felten, Edward; Miller, Andrew; Goldfeder, Steven (2016). Bitcoin and cryptocurrency technologies: a comprehensive introduction. Princeton: Princeton University Press. ISBN 978-0-691-17169-2.
- [19] Li, Zhaozheng; Lei, Weimin; Hu, Hanyun; Zhang, Wei (2019). "A Blockchain-based Communication Non-repudiation System for Conversational Service". 2019 IEEE 13th International Conference on Anti-counterfeiting, Security, and Identification (ASID). pp. 6–10. doi:10.1109/ICASID.2019.8924991. ISBN 978-1-7281-2458-2. S2CID 209320279.
- [20] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and

on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)

[21] CNR at al., Deliverable 5.2: “First version of the simulation environment and monitoring solutions”, 2022