SPARTA

# D5.3

## Demonstrator prototypes

| | |
|---|---|
| **Project number** | 830892 |
| **Project acronym** | SPARTA |
| **Project title** | Strategic programs for advanced research and technology in Europe |
| **Start date of the project** | 1st February, 2019 |
| **Duration** | 36 months |
| **Programme** | H2020-SU-ICT-2018-2020 |

| | |
|---|---|
| **Deliverable type** | Report |
| **Deliverable reference number** | SU-ICT-03-830892 / D5.3 / V1.0 |
| **Work package contributing to the deliverable** | WP5 |
| **Due date** | January 2021 – M24 |
| **Actual submission date** | 29th January, 2021 |

| | |
|---|---|
| **Responsible organisation** | CNIT |
| **Editor** | Gabriele Restuccia |
| **Dissemination level** | PU |
| **Revision** | V1.0 |

| | |
|---|---|
| **Abstract** | This deliverable describes the prototypes coming out of the CAPE program. They include contributions on integration mechanisms coming out of CAPE Tasks 5.1, 5.2, 5.3 and 5.4. Both Verticals (Connected Car and e-Government) are covered. It concludes with the program roadmap. |
| **Keywords** | assessment, certification, safety, security, connected cars, e-government, prototypes |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 830892.

## Editor

Restuccia Gabriele (CNIT)

## Contributors (ordered according to beneficiary numbers)

Maroneze André (CEA)

Massonnet Philippe, Dupont Sébastien, Grandclaudon Jeremy (CETIC)

Nigam Vivek, Dantas Yuri Gil (FTS)

Plate Henrik (SAP)

Sykosch Arnold, Ohm Marc (UBO)

Cakmak Eren (UKON)

Sfetsos Thanasis (NCSR)

Jiménez Víctor (EUT)

Martinez Cristina, Amparan Estibaliz, López Angel (TEC)

Garcia-Alfaro Joaquín, Segovia Mariana, Rubio-Hernán Jose, Blanc Gregory, Debar Hervé, Ludovic Apvrille (IMT)

Bisegna Andrea, Carbone Roberto, Ranise Silvio, Verderame Luca (CINI)

Bernardinetti Giorgio, Palamà Ivan, Pellegrini Alessandro, Restuccia Gabriele, Sirbu Gheorghe, Spaziani Brunella Marco (CNIT)

Yautsiukhin Artsiom (CNR)

Morgagni Andrea, Porretti Claudio (LEO)

Klein Jacques, Bissyande Tegawende, Samhi Jordan (UNILU)

## Reviewers (ordered according to beneficiary numbers)

Norouzian Mohammad (TUM)

Kapusta Katarzyna (TCS)

## Disclaimer

# Executive Summary

Deliverable 5.3, CAPE Demonstrators Prototypes, is the third deliverable of the CAPE program. CAPE stands for Continuous assessment in polymorphous environments. This scientific activity of the SPARTA project addresses the issue of assessing cybersecurity performance of two environments, addressing security and safety co-design on the one hand, complex software systems of systems on the other hand.

This deliverable is the continuation of D5.1, contributing with D5.2 (Demonstrators Prototypes) to the documentation of the first design-implement-integrate cycle of the CAPE program, providing a full picture of the scientific contributions of the CAPE program. D5.3 should be read after D5.2, as it describes the integrated part of the cycle while D5.2 describes the design part.

The first contribution of the deliverable (chapter 3) describes the implementation of each of the 18 CAPE tools. For each of the tools, the description includes the tool's technical characteristics, the procedure for using the tool in test environments, and the results of tests carried out on each tool. The documentation of the internals and the test results is particularly important for CAPE, as it showcases each tool's capabilities.

The second contribution of the deliverable describes the implementation of the tools on the two testbeds representing the two use cases, the "Connected and Cooperative Car Cybersecurity" (chapter 4) and "Complex System Assessment including large software and open source environments, targeting e-Government services" (chapter 5). These two chapters are focusing on the application of the tools in the context of each vertical. They lay out the specificities of each use case, cyber-physical systems on the one hand, complex systems on the other hand, and explain in detail the application and contribution of the CAPE tools on the vertical.

Chapter 6 provides a gathering of these two use-cases with a specific focus on evaluation. This contribution from task 5.4 is still limited in scope in this deliverable but will form the core of the contribution presented in deliverable D5.4.

The third contribution of the deliverable describes the assessment scenarios that will be deployed in the two use cases, the "Connected and Cooperative Car Cybersecurity" (chapter 6) and "Complex System Assessment including large software and open source environments, targeting e-Government services" (chapter 7). They describe the concrete usage scenarios and attacks that can be carried out in the prototype testbeds, with the aim to describe an assessment pipeline formed by the association of CAPE tools. As a result, what is showcased in the context of each use case is a complete toolchain for the cybersecurity (and safety in use case 1) assessment of each use case.

This deliverable is a concrete materialization of cybersecurity and safety assessment and validation in two concrete examples. This prepares the upcoming second cycle of design-implement-validate for the tools in CAPE, as well as the work on certification profiles and cybersecurity certification started earlier in CAPE, in association with WP11.

In terms of governance, the deliverable provides an example of how CAPE partners have successfully been able to collaborate towards integrated research and validation workflow. This is particularly important as evaluation and validation is the conclusion and an extremely important part of the research. It often is extremely expensive for individual researchers. The mutual exchange and joint elaboration of validation tools and processes is thus an important lessons-learned from CAPE.

# Table of Content

# List of Figures

# List of Tables

# Chapter 1 Introduction

Deliverable 5.3, Demonstrators prototypes, is the third deliverable of the CAPE program.

CAPE is listed in the description of action as Work Package 5 (WP5). It is one of the four scientific programmes of the SPARTA project selected during the project construction and being executed during the project lifetime to demonstrate concretely how research governance activities are handled within SPARTA. CAPE stands for Continuous Assessment in Polymorphous Environments. It acknowledges the issue of assessing cybersecurity performance of two environments, addressing security and safety co-design on the one hand, complex software systems of systems on the other hand.

This deliverable provides the prototypes coming out of T5.4. They include contributions on integration mechanisms coming out of the four tasks in the CAPE program.

- **T5.1 - Assessment procedures and tools**: Provide tools and methods for continuous assessment and certification.
- **T5.2 - Convergence of security and safety**: Study techniques and specifications for the integration of security and safety
- **T5.3 - Risk discovery, assessment and management for complex systems of systems**: Address security requirements on SoS using modern software engineering methods
- **T5.4 - Integration on demonstration cases and validation**: Demonstrate that tools and techniques described in T5.1, T5.2 and T5.3 in the CAPE verticals could be evaluable with the purpose of a future unified certification scheme.

The outcome of the WP5 tasks takes a generic continuous assessment framework based on the **V-Model software development** process. Task 5.1 focuses on the **framework specification**, describing how the various tools that compose the framework can contribute to the continuous assessment process. Task 5.2 proposes techniques for integrating security and safety on the **connected car vertical** such as safety-security co-analysis techniques, requirements engineering, modelling and implementation, safety and security co-verification and validation techniques, etc. Task 5.3 proposes a set of tools that can be used by software development organizations for compliance activities by detecting the presence of known **security vulnerabilities in 3rd party software** and addressing supply chain attacks. Task 5.4 will demonstrate the continuous assessment framework in the **connected car** and **e-government** verticals by verifying the **evaluability** of the two verticals.

Also, as stated in D5.1 [1], the vertical related to financial services will not be further pursued. Originally meant to demonstrate assessment tools developed in the context of CAPE task 5.3, further investigation revealed that those tools are largely independent of a given industry or vertical and their specific security and certification requirements. At high-level, those tools aim at the detection and prevention of known and unknown security vulnerabilities in own and third-party code as well as the detection of so-called supply chain attacks. Those objectives, however, apply to virtually every industry and use-case. Moreover, it turned out that many tools developed by CAPE partners target specific technologies, e.g., programming languages and devices. The majority of those technologies are not present in the software application part of the financial services use case, thus, cannot be demoed in this context. For those reasons, it was decided to demonstrate tools developed as part of CAPE task 5.3 at the example of the other use-cases, which will also allow to focus CAPE partners' efforts.

## 1.1 Structure of the document

This deliverable is organized as follows:

- **Chapter 1** is the current section presenting the objectives and structure of the document.
- **Chapter 2** presents a list of all CAPE Tools also indicating the partner, task and Vertical of belonging and its V-model phase.
- **Chapter 3** analyses all the prototypes included in the context of T5.1 using methods for verifying the software requirements defined in D5.1 and D5.2.
- **Chapter 4** discusses the roadmap phases for Task 5.2 activities belonging to the context of the D5.3.
- **Chapter 5** focuses on Synergy and integration of individual contributions in the context of T5.3.
- **Chapter 6** discusses the evaluation concepts and contributions on integration mechanisms coming out of the three other tasks in the context of T5.4.
- **Chapter 7** illustrates all the different scenarios for Vertical 1 demonstrating the development phases previously discussed in T5.2.
- **Chapter 8** illustrates all the different scenarios for Vertical 2 demonstrating the prototypes and tools discussed in the previous tasks.
- **Chapter 9** discusses the future roadmap for CAPE.
- **Chapter 10** presents the summary and conclusions of the report.

# Chapter 2 CAPE Tools

The next table shows a list of the CAPE tools described in D5.1 and D5.2. It also shows their involvement in each of the two Verticals and in which task they belong to.

| Tool | Partner | V-model Phase | Task | Scenario |
|---|---|---|---|---|
| Approver (RAA) | CINI | Development process | T5.3 | e-Government (Vertical 2) |
| AutoFOCUS3 (AF3) | FTS | Development process; All phases | T5.2 | Platooning (Vertical 1) |
| Buildwatch (BW) | UBO | Application development | T5.3 | e-Government (Vertical 2) |
| Frama-C (FC) | CEA | Development, Unit testing | T5.3 | Platooning (Vertical 1) |
| Legitimate Traffic Generation System (LTGen) | IMT | Operations | T5.1 | Stand-alone |
| Logic Bomb Detection (TSOpen) | UniLu | Design (from unit testing to acceptance testing) | T5.3 | e-Government (Vertical 2) |
| Maude (MAU) | FTS | Verification and Validation | T5.2 | Platooning (Vertical 1) |
| NeSSoS Risk assessment tool (RA) | CNR | Risk Management process at the global level | T5.1 | e-Government (Vertical 2) |
| OpenCert (OC) | TEC | Safety Goals definition; Safety Goals validation, Safety Analysis, Trade- Off Analysis, Assessment | T5.2 | Platooning (Vertical 1) |
| Project KB (KB) | SAP | All phases | T5.3 | e-Government (Vertical 2) |
| Risk assessment for cyber-physical interconnected infrastructures (MRA) | NCSR | Requirements analysis | T5.1 | Stand-alone |
| Sabotage (SB) | TEC | Functional and technical Safety concept design; Functional and technical Safety concept verification | T5.2 | Platooning (Vertical 1) |
| SafeCommit (SF) | UniLu | Software development (of the libraries used by the application) | T5.3 | Platooning (Vertical 1) |
| SideChannelDefuse (FS) | CNIT | Deployment | T5.1 | Stand-alone |
| Steady (VA) | SAP | Design (from component design to deployment) and Operations | T5.3 | e-Government (Vertical 2) |
| SysML- Sec (TTool) | IMT | All phases | T5.2 | Platooning (Vertical 1) |
| VaCSInE (VCS) | CETIC | Operations | T5.1 | Platooning (Vertical 1) |
| Visual investigation of security information (VI) | UKON | Security Analysis, Verification and Validation | T5.3 | e-Government (Vertical 2) |

Table 1: CAPE Tools

# Chapter 3 Prototypes for Assessment procedures and tools (T5.1)

## 3.1 Introduction

This task addresses the aspects related to assessment automation, augmenting the assessment toolbox to support pre-assessment by users, as well as incremental assessment and continuous monitoring. Task 5.1 aims at proposing a framework for automated cybersecurity assessment. The framework is based on the V-Model lifecycle for software/hardware development, safety and security and aligns certification activities to the various steps of the model. Figure 1 matches the various tools of the framework with the steps of the security engineering process. The framework is applied to the two CAPE verticals using tools prototypes and will be demonstrated in D5.4.

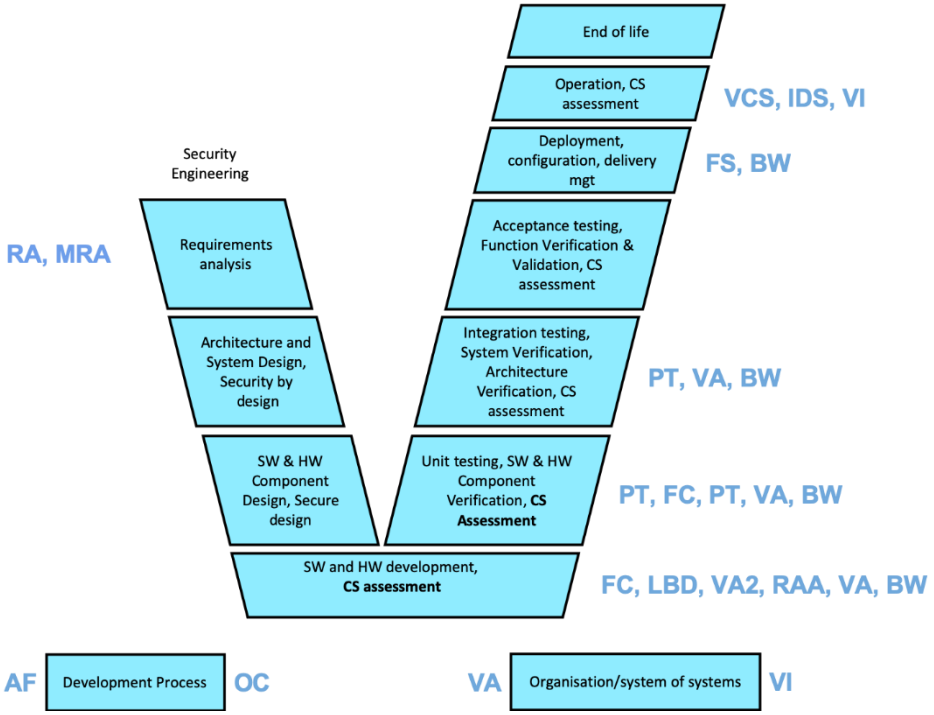


Figure 1: V-Model associated with the tools of the framework

The roadmap for the final version of the framework tools prototypes will be guided by the demonstration scenarios on the verticals. It will follow a similar methodology as for the early prototypes, with a focus on integration in the verticals demonstrations:

- M25-M26: **refined design** of the tools in order to reach the final version of the prototypes
- M27-M28: **implementation** of the final prototype version of the tools
- M25-M35: **verification and validation** of the updated prototypes
- M29-M35: **integration and evaluation** of the framework tools on the various demonstration scenarios

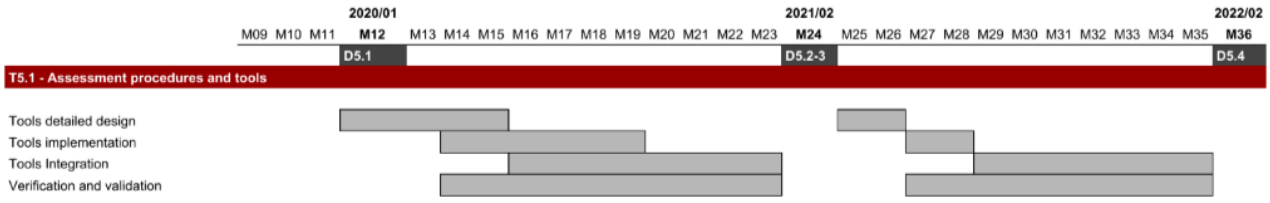


Figure 2: T5.1 Roadmap

## 3.2 Prototypes Analysis

In this chapter, we will analyse all the prototypes associated with each tool describing its:

- **Description and goal**:
  - Objectives and summary description of the prototype.
- **Technical characteristics:**
  - How the prototype was developed explaining its technical characteristics, security functionalities, components, and constraints.
- **Experimental test activities:**
  - Show the experimental tests and test environments.
- **Results:**
  - Results from the tests (when they can be applied at this stage).

These items will be used to verify the software requirements that are intended to be implemented in the different design and development cycles defined in D5.1 [1] and D5.2 [2]. The demonstration upon their completion will be shown in D5.4 [3]. Some tools will also present a complete report for their testing results in the upcoming D5.4.

### 3.2.1 Approver (RAA) - CINI

The high-level development roadmap is to implement the software requirements SR1.1 and S1.2 within 2020-21 such that it can be demonstrated at the project end, as explained in Table 2. Table 2

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1.1** | Submit the app source to the DevSecOps plugin | Check if scans succeed and the tool successfully report the security vulnerabilities to the issue tracker | e-Government (Vertical 2) |
| **SR1.2** | Scan a mobile app package | Check if scans succeed and findings are correct | e-Government (Vertical 2) |

Table 2: Approver - Demo scenarios and verification methods

#### 3.2.1.1 Description and goal

As documented in D5.1, Approver is an automatic toolkit for the in-depth, fully automatic security analysis of mobile applications. Approver automatically detects, evaluates and provides comprehensive reports explaining the security risks hidden in the mobile applications.

Approver will be used in the Vertical 2 scenarios to i) Detect security vulnerabilities in the IDP app package ii) Detect security vulnerabilities in the CIE ID Android app during the development phase and suggest security mitigations.

#### 3.2.1.2 Technical characteristics

At a high-level, Approver is composed of a set of modules for both Static Analysis (SAST) and dynamic analysis (DAST). Each module, developed as a microservice using Docker technology, enables a different security analysis and is managed by an orchestration layer. Besides, each module exposes a set of RESTful APIs. The modules for SAST are in charge of analyzing the application package according to its content. Examples of implemented SAST analysis include vulnerability analysis, permission analysis, and string analysis. Instead, the DAST modules aim to install the application package in a testing environment and evaluate the application's security during the execution. Examples of DAST analysis include network analysis, API monitoring, and filesystem monitoring.

Finally, Approver provides a web front-end that allows to:

View the detailed results of each application analyzed.

1. Download all the artifacts produced during the analysis.
2. Download a security report which contains all the identified issue.

The analyses carried out by APPROVER are coordinated by a central module, called **Orchestrator**, that manages the flow of requests between the various modules during the analysis of applications. The management layer is completed by the Authenticator service, which controls the authentication between the different modules, and the Account Management service, which manages the accounts enabled to use the platform.

From a technical point of view, APPROVER modules are mainly developed using Java (Spring) and Python (Flask). Still, the modular architecture allows the dynamic inclusion of new modules regardless of the language used MySQL (for modules developed in Java) and MongoDB (for modules programmed in Python) are mainly used for data persistence and Amazon S3 for the persistence of files and artifacts generated by the analysis phase. To ensure the highest security level, all communication between the various modules inside APPROVER and between the APPROVER modules and external services takes place exclusively via HTTPS.

***Constraint and limitations:*** *APPROVER can be used as a service (SaaS) or installed on-premise. In the case of an on-premise solution, the tool needs to be installed in a UNIX-like environment that supports nested-virtualization technologies and the docker environment. The suggested minimum hardware configuration comprises three nodes with quad-core processors, 16 GB of RAM each, and 100GB of storage.*

#### 3.2.1.3 Experimental test activities

We described the experimental test activities that will be used to evaluate the effectiveness of the APPROVER tool.

**Submit the app source to the DevSecOps plugin**

**Input**: The source code of an Android application package

**Output**: A list of security vulnerabilities in the issue tracker of the DevSecOps pipeline

**Test Procedure**

The developer is expected to push a commit on the code repository that contains the application source and is connected to the DevSecOps pipeline. The Approver plugin is expected to build the app package and to send it to the analysis backend.

After the analysis is completed, the developer can check in the issue tracker a list of issues that represent the security vulnerabilities contained in the app. At the same time, the Security Analyst can access the same report on the Approver web interface.

**Scan a mobile app package**

**Input**: An Android application package (APK).

**Output**: A per-app security report.

**Test Procedure**

To test the new SAST vulnerability analysis module, the Security Analyst is expected to submit an Android application package through the Approver web interface. The new SAST module is expected to analyze the binaries of the pp and send the result to the backend collector to generate the security report.

After the analysis is completed, the Security Analyst can access the security report on the Approver frontend and download a PDF version of the report.

### 3.2.1.4 Results

We describe here a brief overview of the result of the CIE ID Android app. The detailed results regarding the IDP Android app's experimental evaluation in the context of Vertical 2 will be described in the appropriate deliverable (D5.4).

**Submit the app source to the DevSecOps plugin**

In this evaluation scenario, a developer has sent a new commit request to the GitLab repository containing the mobile app's code. At this stage, after the building process, GitLab triggers the Approver CI/CD plugin (see Figure 3) that will send the application to the Approver backend. Once the process has been completed, APPROVER successfully detected 13 different vulnerabilities and opened 13 different security issues (see Figure 4) in the GitLab tracking system containing the description of the vulnerability, its severity, and a description of possible technical remediation.

**Scan a mobile app package**

To test this scenario, a Security Analyst accessed the APPROVER webpage and submitted the Android application package for the analysis. The tool successfully analyzed the application package and showed the results in Figure 5. Also, the tool allowed the download of a PDF report.

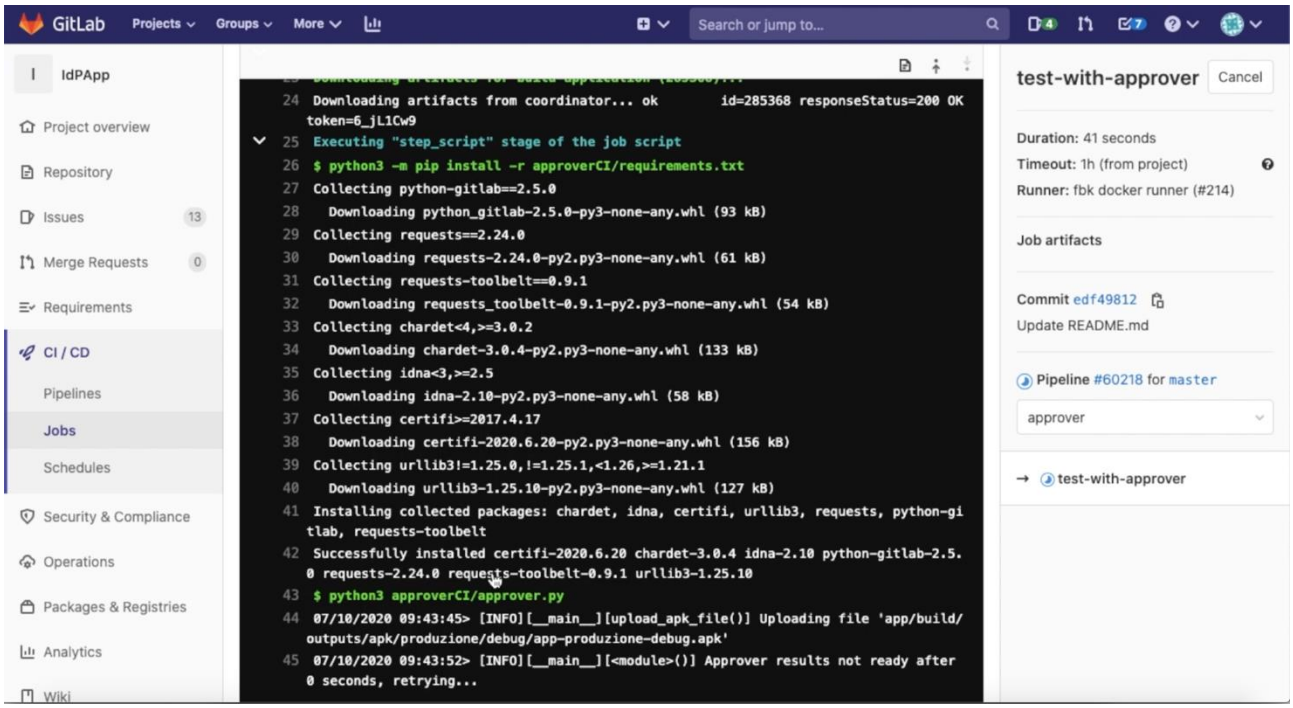


Figure 3: CI/CD plugin - app submission process

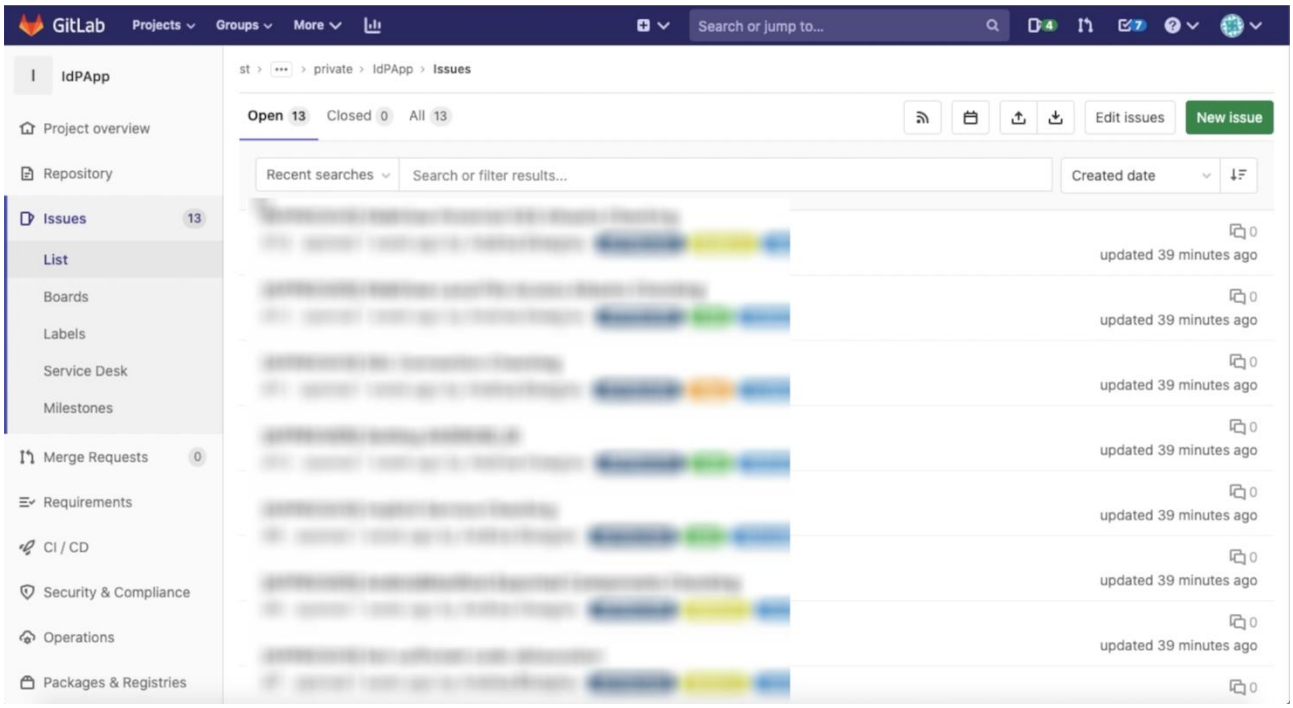


Figure 4: CI/CD plugin - security issues

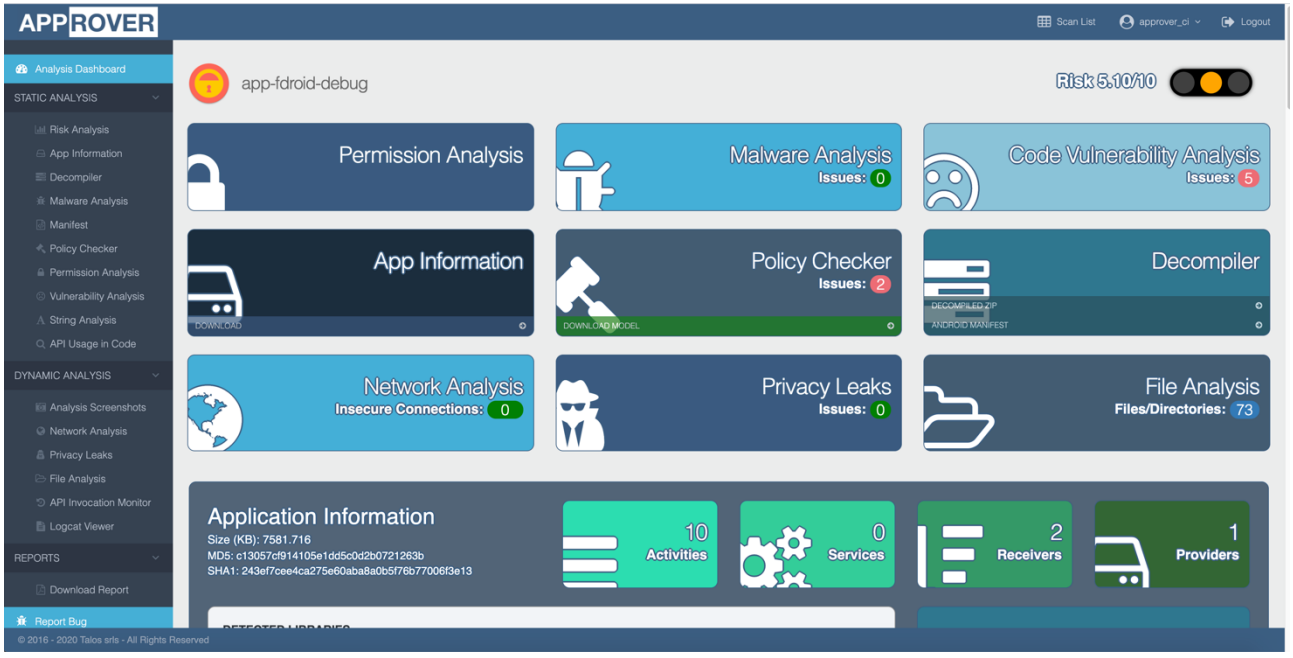


Figure 5: Approver - Analysis Dashboard

### 3.2.2 AutoFOCUS3 (AF3) – FTS

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **C-ACC Safety and Security Co-Validation** | We will use the machinery developed to evaluate the safety and security of the architectures used for Coordinated Adaptive Cruise Control | Carry out simulations and formal verification using Maude. | Connected Car (Vertical 1) |
| **TARA and HARA modelling** | We use models in Goal Structure Notation (GSN) and attack defense trees for modelling, respectively, the safety and security analysis of the platooning scenario | Validation of safety and security analysis. | Connected Car (Vertical 1) |

Table 3: AutoFOCUS3 - Demo scenarios and verification methods

#### 3.2.2.1 Description and goal

As documented in D5.2, the main goal of the development is to enable the existing Model-Based Engineering too AutoFOCUS3 to support the following features:

- **Safety and Security Co-Analysis:** The implemented extension of AutoFOCUS3 supports the modelling of safety cases using Goal Structure Notation (GSN) with quantitative evaluation. Moreover, we extended AutoFOCUS3 to enable the modelling of security arguments based on Attack Defense Tree. Finally, we have implemented algorithms for extracting Attack Defense Trees from GSN models.
- **Modelling of Platooning Scenario:** We have implemented models that enable the implementation and validation of safety and security of vehicle platooning scenarios using simulation and formal verification methodologies

#### 3.2.2.2 Technical characteristics

We have implemented the AutoFOCUS3 prototype so to achieve the goals described above

- AutoFOCUS security plugin has been implemented in JAVA with the modelling elements necessary for the specification of Goal Structure Notation Models with Quantitative Evaluation and the modelling of Attack Defense Trees;
- Modelling Vehicle Platooning: We have implemented in AutoFOCUS3 safety and security analysis for the Connected Car scenario. These have been documented in D5.1.
- Modelling of Vehicle Platooning Cooperative Adaptive Cruise Control (CACC): We have implemented models in AutoFOCUS3 algorithms used for Cooperative Adaptive Cruise Control as well as security countermeasures based on plausibility checks, as described in the D5.2 (Protection Profile).

The tools and models implemented can only be used to justify scenarios where only the vehicle speeds are controlled and protected.

#### 3.2.2.3 Experimental test activities

We have evaluated our machinery by applying it to the process described in D5.1 to develop safe and secure system. Our development has been successfully used for the co-analysis of vehicle platooning supporting the development of requirements.

Moreover, as we described in D5.2, we have applied the models developed with CACC for Vertical 1 (Connected Cars). As we detail in this document, our models are being deployed in the rovers available at FTS and TEC. Finally, we have also used our machinery together with the tool Maude (also described in this document) for the verification of platooning scenarios.

#### 3.2.2.4 Results

Our key results are described in this Document under the Connected Car scenario. In summary, we have successfully generated C code from our models for the CACC and the plausibility checks, and this generated code has been into the FTS rovers. Moreover, our integration with formal verification tools have been used to discover 3 new attack scenarios, also described within this document.

### 3.2.3 Buildwatch (BW) - UBO

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Ingest a common build dependency | Check that a report comprises all cyber observable objects created or modified by the build process of the software | e-Government (Vertical 2) |
| **SR2** | Compute the difference between two versions | Two Versions are built in the Buildwatch Sandbox two times each. The differences are computed between all four reports. The computation yields no result between builds of the same version but computes the same differences on reports of different versions. | e-Government (Vertical 2) |

Table 4: Buildwatch - Demo scenarios and verification methods

#### 3.2.3.1 Description and goal

Buildwatch can monitor the creation and manipulation of forensic artifacts, e.g. File created or read and network connections established, during the software build process. These may be monitored on each update of a dependency of a software project and subsequently compared. If changes between the updates occur, the dependency behavior may be changed dramatically, possibly even maliciously. Buildwatch highlights these changes for manual review.

In software development environments within a critical use case like identity management, the consequences of maliciously behaving software may be disastrous. However, software and algorithms in this field are on active development by a multitude of cooperating parties. This complex and rapidly changing environment renders this field especially susceptible to software supply chain attacks.

In order to prevent these attacks, dependencies have to be reviewed carefully whenever changes are introduced. However, reviewing all changes of every dependency is not feasible. Buildwatch aims to ease the workload by first transposing source code changes to changes made on the host machine during the build of the dependency and further ease assessment of these changes by comparing the different changesets and highlight their differences.

SAML IdP which is developed by CINI within SPARTA leverages Shibboleth, a web-based single sign-on log-in system. Shibboleth is developed as open-source under the Apache 2 license. It comprises several components that ease authentication and authorization of users and thus it handles sensitive data. Given the complexity and criticality of this project makes it a potential target for attacks.

This test series demonstrates Buildwatch's capabilities in a complex real-life scenario.

#### 3.2.3.2 Technical characteristics

Based on the descriptions in "D5.2 - Demonstrators specifications", a custom tool for submission is implemented.

While the main use case for Buildwatch is the monitor of all dependencies in one project, it can detect manipulation of the host system on the monitored process, commonly dependency installation, build, or test. If the malicious behaviour of the dependency is only exposed during runtime of the dependent software (sometimes only when specific circumstances are met), it will not be detectable by monitoring the development processes.

Naturally, Buildwatch is only able to detect the *change* of behaviour. There are attacks on the supply chain that do not fabricate such change, e.g., typo squatting attacks on the dependency definition. Dependencies that were injected into the dependency repository to resemble a benign package will not be able to be detected using Buildwatch.

Limitations: No protection against typo-squatting attacks. Highly targeted/sophisticated attacks that leverage evasion techniques might not trigger malicious behaviour during analysis (the general problem of dynamic analysis).

#### 3.2.3.3 Experimental test activities

The repository of Shibboleth is submitted to the sandbox for every commit and the build process is executed in the sandboxed environment. Buildwatch currently only supports Linux-based build environments. Forensic artifacts are captured by interpretation of the system calls to the kernel using the `systap` interface. Patterns of similar forensic artifacts are learned and filtered. The remaining forensic artifacts are presented to the developer. Statistics on the review requirements by the commit and by version change will be provided to estimate the required efforts during development.

#### 3.2.3.4 Results

A first preliminary study revealed the separability of benign and suspicious forensic artefacts. A malicious version of a software package tends to cause more forensic artefacts and of different types. It was observed that malicious code was often contained in newly added files that are executed by previously unseen processes. Most often this causes additional network connections that are unusual in relation to benign versions of the same package. Further results of Buildwatch based on the large scale analysis of Shibboleth will be provided on D5.4.

### 3.2.4 Frama-C (FC) - CEA

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1.1** | CI-based configuration and use cases | Check applicability and us- ability on a set of existing code bases | Set of open-source code bases |
| **SR1.2** | Standardized output for- mat (SARIF) | Feed output to other tools compatible with SARIF | Integration in the CI pipeline produced in T5.3 |
| **SR2** | Audit-mode outputs and validation as inputs | Modify outputs and re-feed them as inputs to check conformance | Set of open-source code bases |

Table 5: Frama-C - Demo scenarios and verification methods

#### 3.2.4.1 Description and goal

The main goal of the development is to integrate Frama-C in a CD/CI pipeline, allowing the static analysis to take place after each commit, providing faster feedback and without requiring users to have the tool installed in their own machine.

The amount and complexity of the result data require a structured format to be able to take into account as much information as possible. In order to ensure better compatibility with other tools, and future integration into the development process, the use of the standardized SARIF format enables integration with other tools, either viewer (for IDE-based result visualization, such as with VS Code's Sarif Viewer plug-in) or more complex frameworks, such as Autofocus.

A secondary goal, not directly related to build integration, is the creation of an audit mode for Frama-C. It has a dual purpose: to help developers know the implicit assumptions in their analyses, and to aid them in giving the evidence so that external users can more easily re-run the analysis and check that it was properly setup.

#### 3.2.4.2 Technical characteristics

The CI-based integration was developed by creating Docker images of Frama-C which incorporate all of the required plug-ins, ready for analysis. These images are compatible with pipeline integration mechanisms such as Github Action and Gitlab Runners. The Docker images allow running Frama-C automatically and producing a SARIF report after each change to the code. However, the initial configuration of analysis still might require local usage of Frama-C for faster feedback. Security considerations are the same as for integrating Docker images in a build pipeline; for instance, users with non-public code can host their own Gitlab instance, download the Docker images, and run them inside their network. Since the Dockerfiles used to produce the images are also made available, the user can rebuild their own private Docker images for added security. The main limitation of the usage of Frama-C in a CD/CI pipeline is the initial setup, which greatly benefits from immediate user feedback, and thus should be performed locally.

For SARIF support, Frama-C uses tools available in the OCaml language, namely ppx_deriving, which allow semi-automated production of code to import/export documents in the JSON format. Updating support for newer SARIF versions still requires some manual steps. Concerning limitations, the SARIF report contains most but not all of the information present in an analysis. Its output allows mapping the alarms to the original code. However, some aspects of the Frama-C analysis cannot be directly captured by SARIF, so advanced usage of the tool benefits from direct usage of the Frama-C GUI, which is not available remotely via the CI pipeline.

For the audit mode, the use of a JSON file, which serves both to present results and as structured input for validation, enables easy configuration and inspection without the need for specialized tools. The audit mode provides feedback information about the parametrization of analysis via Frama-C itself: due to its large set of tunable options, an oblivious or malicious user could set it up in a way that some assumptions about the analysis could go unnoticed. By providing specific feedback towards code and analysis reviewers, the audit mode prevents such situations from happening.

#### 3.2.4.3 Experimental test activities

Frama-C's public open-source case studies Gitlab repository (https://git.frama-c.com/pub/open-source-case-studies) provides a set of varied C code bases that validate the proposed CI pipeline on small and large samples, both simple and complex. A similar setup using Github allows the validation of the Frama-C Github Action. These runs produce sample output with alarms and artifacts with the SARIF output. They allow checking that the Docker image is working and that the SARIF output is useful. For Gitlab, Frama-C uses its own public instance with a set of use cases serving as an example. The setup of an example Github repository using the Frama-C Github Action ensures that the audit mode works, and that users can refer to it when trying it themselves.

For the audit mode, non-regression testing is available in the Frama-C platform itself, via an oracle and a test that the input does perform the required checks. Another important test is external validation by users in the role of auditors. A review of the analysis documentation has been performed to ensure such users have clear information concerning the implicit assumptions and can thus check that the information provided by the audit mode is sufficient to prevent malicious users from bypassing the checks and reports. An audit being an ultimately human analysis (since it has not been entirely formalized), it is impractical to devise purely mechanical means to prevent all kinds of “cheating”. Manual testing remains thus necessary for this use case.

#### 3.2.4.4 Results

For SR1.1, usage of both Github and Gitlab CI tools allowed the identification of common features and improvement of the Docker images, to ensure they work with both environments without requiring adaptations. They also allowed refining the SARIF output to provide more useful information and filtering of excessive data.

For SR1.2, the SARIF output could be successfully validated using Sarif-Multitool and imported into VS Code to verify that at least the basic information provided by the report (location of alarms, warning messages, etc.) is relevant and meeting the expectation of consumer tools. Testing also revealed that several sources of non-determinism (e.g., file paths, timestamps, tool versions, etc.) should be made optional in the report, as recommended by the SARIF standard itself, and to facilitate regression testing. Such features provide a more usable output for end-users.

For SR2, the development of the audit mode enabled the identification and fixing of some possible misconfigurations of parameters, as well as an update of the documentation to take new scenarios into account. It showed the importance of providing a machine-validated output to prevent it from becoming stale and desynchronized with respect to future developments. It also showed that the human element should be taken into account: the guarantees offered by the tool must be well understood in order to use it efficiently.

### 3.2.5 Legitimate Traffic Generation system (LTGen) - IMT

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1.1** | Metrics to measure realism | Check the metrics against real traffic traces | Test set of real traffic traces |
| **SR1.2** | Anonymization functions | Assess the privacy of processed traces | Privacy impact assessment (PIA) |
| **SR2.1** | Metrics to measure malice | Check the potential damage to a target system | Set of target systems |
| **SR2.2** | Mutation functions | Measure mutation ratio | State-of-the-art mutation metrics |

Table 6: Legitimate Traffic Generation System - Demo scenarios and verification methods

#### 3.2.5.1 Description and goal

As indicated in D5.2, LTGen is a stand-alone component that neither interacts with other components nor integrate into the CI/CD pipeline. It does however, aims at contributing to the evaluation of common cybersecurity tools such as intrusion detection systems (IDS) or security information and event management systems (SIEM) as introduced in D5.1. This is achieved by the generation of legitimate traffic based on the input of previously observed real-life traffic. This enables the characterisation of potential false positives, from previously unseen legitimate samples or corner

cases, and help populate datasets which are hard to come by (mainly for privacy reasons), and often scarce.

Future works involve the exploration of false negatives by the generation of malicious traffic this time. In particular, the new tool aims at generating mutated traffic from malicious samples. The mutated traffic is crafted to look close to legitimate traffic, in order to trigger misclassification.

Both aspects should contribute to the data-driven evaluation of detection tools.

Due to unforeseen hiring difficulties, the development has been stalled and this deliverable can only report on SR1.1. The remainder of SRs will be implemented and demonstrated in D5.4.

#### 3.2.5.2 Technical characteristics

The initial LTGen prototype is inspired by the DARPA project where sequences of actions were designed to represent typical human-initiated network activities (web browsing, email communication, etc.). LTGen thus relies on the generation of flows from a number of protocols to emulate these activities, including HTTP(S), IMAP, SMTP, FTP and SMB. These flows are generated by automated agents running real applications, generating additional flows (DNS, ARP, NetBios, etc.).

On our physical infrastructure (a cluster of fully-meshed servers), we deploy an Openstack instance to host a 9-virtual machine topology, as shown in Figure 6. The topology is divided in three subnets, namely the *DMZ*, the *Internal* network, and the *Office* network. The mentioned *FTP* and *Mail* servers are instances of widespread open-source packages (e.g., postfix, vsftp, etc.) and feature 100 users each. HTTP connections are balanced over two *Client* machines and directed to the *Intranet* server in the *Internal* subnet and the remote *Live stream* server. The *Intranet* server is emulated by combining two pieces of software, namely INetSim[1] and web.py[2]. This allows to receive requests to arbitrary URLs and respond with a custom page containing a number of characters varying according to a custom distribution (e.g., a Gaussian). This enables the generation of a variety of traffic with a wide range of packet sizes. The *Live stream* server handles video streaming activities from the *Client* machines by providing videos from the DASH dataset[3].

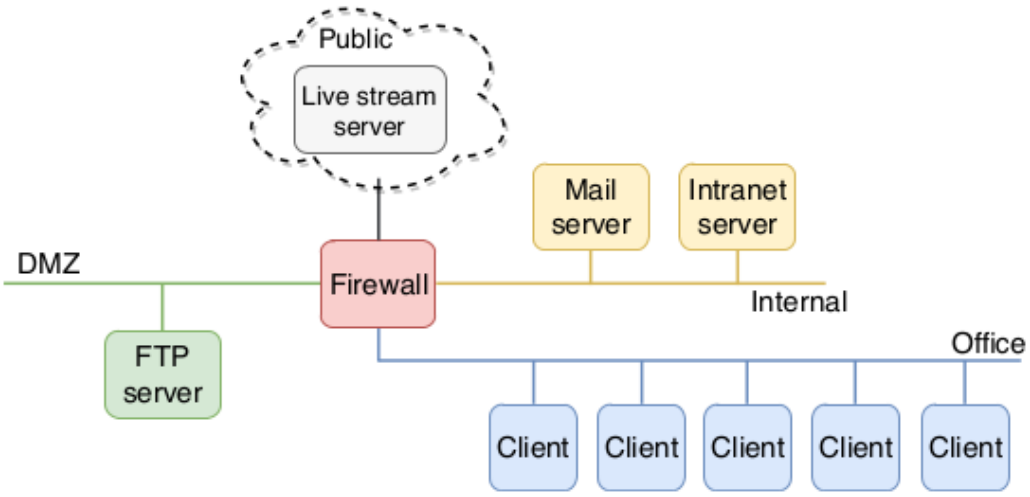


Figure 6: Network architecture hosted on the LTGen's Openstack infrastructure

[1] Internet Services Simulation Suite, available at: https://www.inetsim.org

[2] web.py Python web framework, available at: http://webpy.org/

[3] ITEC – Dynamic Adaptive Streaming over HTTP datasets: https://dash.itec.aau.at/dash-dataset/

Details on the other services present in the LTGen infrastructure are shown in Table 7.

| Machine | Operating System | Capability | Number of instances | Services | Software |
|---|---|---|---|---|---|
| **Firewall** | CentOS Server 5.6 | 2G RAM 1 VCPU | 1 | NAT | iptables |
| **Mail server** | Ubuntu Server 16.04 LTS | 2G RAM 1 VCPU | 1 | email | Postfix Dovecot OpenSSL |
| **HTTP server** | Ubuntu Server 16.04 LTS | 2G RAM 1 VCPU | 1 | HTTP DNS | INetSim web.py |
| **FTP server** | Ubuntu Server 16.04 LTS | 2G RAM 1 VCPU | 1 | FTP | vsftp 3.0.3 |
| **Client** | Ubuntu Server 16.04 LTS | 2G RAM 1 VCPU | 5 | | |

Table 7: Services in the LTGen infrastructure

#### 3.2.5.3 Experimental test activities

The objective of the experimental test activity is to evaluate the accuracy of LTGen in generating traffic that contains pre-defined characteristics (extracted from real traffic). As the traffic of reference, we chose a dataset from MAWILab[4], a well-known public traffic database. Usually, MAWILab collects 15-minute traces daily, but we randomly selected three traces from the 2017 “Day in the Life of the Internet” (DITL) project[5]. We extracted the following characteristics:

- Average throughput
- Volume distributions for 4 selected protocols (HTTP(S), IMAP, SMTP, FTP)

These characteristics are fed to LTGen that will generate the protocol agents’ configuration for traffic generation. We conducted measurements at two levels:

- *Aggregate level*: we measure the errors in the generated traffic, i.e., deviations from the traffic of reference, with respect to the above-mentioned metrics;
- *Flow level*: we investigate further the distributions in terms of TCP session start times and packet size distributions between the generated traffic and the traffic of reference.

[4] MAWILab dataset, available at: http://www.fukuda-lab.org/mawilab/data.html

[5] MAWILab – A Day in the Life of the Internet, available at: http://mawi.wide.ad.jp/mawi/ditl/ditl2017/

#### 3.2.5.4 Results

| | | HTTP(S) (MB) | IMAP (MB) | SMTP (MB) | FTP (MB) | Total (MB) | AT (MB/s) |
|---|---|---|---|---|---|---|---|
| Test 1 | Ref. traffic | 1303.49 (98.05%) | 15.06 (1.13%) | 10.15 (0.8%) | 0.6 (0.04%) | 1303.49 | 1.47 |
| | LTGen traffic | 1391.19 (97.91%) | 17.64 (1.24%) | 11.06 (0.8%) | 0.85 (0.06%) | 1420.75 | 1.57 |
| | Std deviation | 188.18 | 1.83 | 1.09 | 0.27 | 189.85 | 0.21 |
| Test 2 | Ref. traffic | 2404.57 (99.83%) | 1.57 (0.06%) | 2.19 (0.09%) | 0.93 (0.02%) | 2409.28 | 2.67 |
| | LTGen traffic | 2405.61 (99.63%) | 2.29 (0.07%) | 4.05 (0.21%) | 1.07 (0.07%) | 2413.03 | 2.68 |
| | Std deviation | 20.03 | 0.87 | 1.17 | 0.41 | 19.86 | 0.02 |
| Test 3 | Ref. traffic | 2683.14 (99.30%) | 16.22 (0.6%) | 2.51 (0.09%) | 0.06 (0.002%) | 2683.14 | 3.00 |
| | LTGen traffic | 2662.10 (99.36%) | 14.70 (0.55%) | 2.15 (0.08%) | 0.08 (0.003%) | 2662.10 | 2.97 |
| | Std deviation | 27.33 | 2.82 | 0.40 | 0.06 | 27.33 | 0.02 |

Figure 7: LTGen's accuracy in generating traffic close to MAWILab DITL 2017

Figure 7 illustrates the results of the traffic generated by LTGen in the effort of reproducing the characteristics from the traffic of reference, i.e., at aggregate level. Each data point represents the mean of 10 experimental runs (*AT* stands for average throughput). Overall, the generated traffic and the traffic of reference are relatively close to each other in terms of the respective volumes of each protocol, for all three test traces. Regarding the HTTP(S) protocol, the difference in generating 1.303GB of traffic is of 87.7MB, in the first test. The other two tests yield better results with less than 20MB difference for both cases. For other protocols, although they are smaller in volume, the biggest difference observed is 2.58MB, 1.86MB and 0.25MB for the IMAP protocol in Test 1, the SMTP protocol in Test 2 and the FTP protocol in Test 1, respectively. Differences for minority protocols are relatively bigger as for such small volumes of traffic, the traffic generation agent needs a relatively small number of activities to produce. Hence, the random subset of activities may overflow the upper bound of expected traffic. In terms of throughput, the flows generated by LTGen are generally close to the reference ones, with the biggest error occurring in the first test (0.1MB/s difference, corresponding to a 6.8% error).

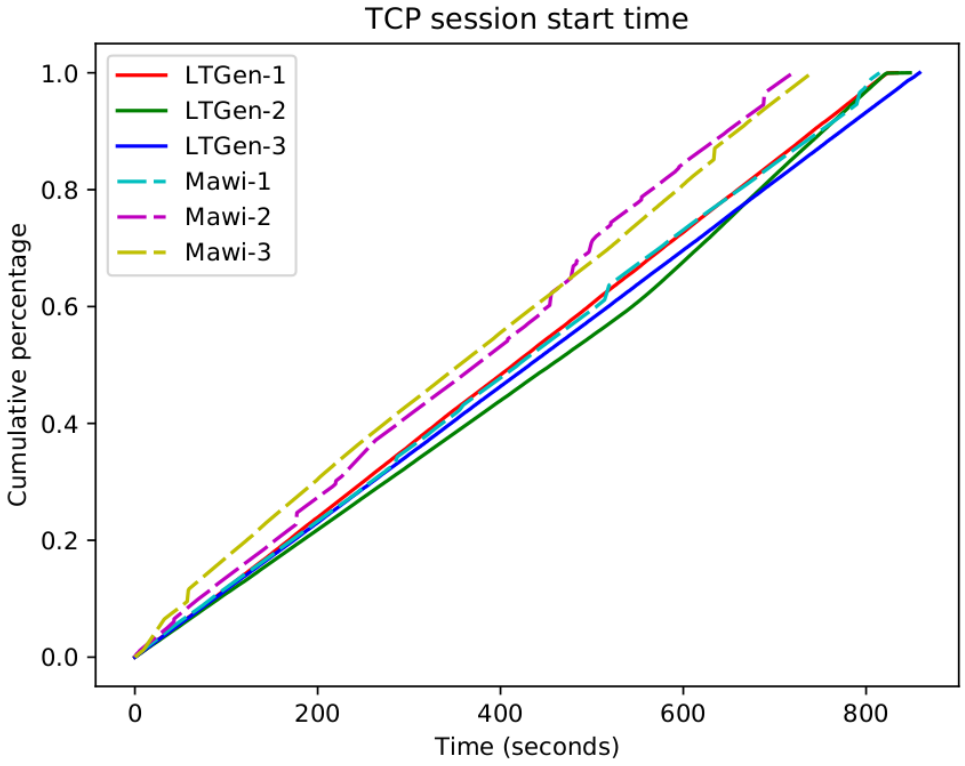


Figure 8: TCP session start time distributions of MAWILab and LTGen traffic

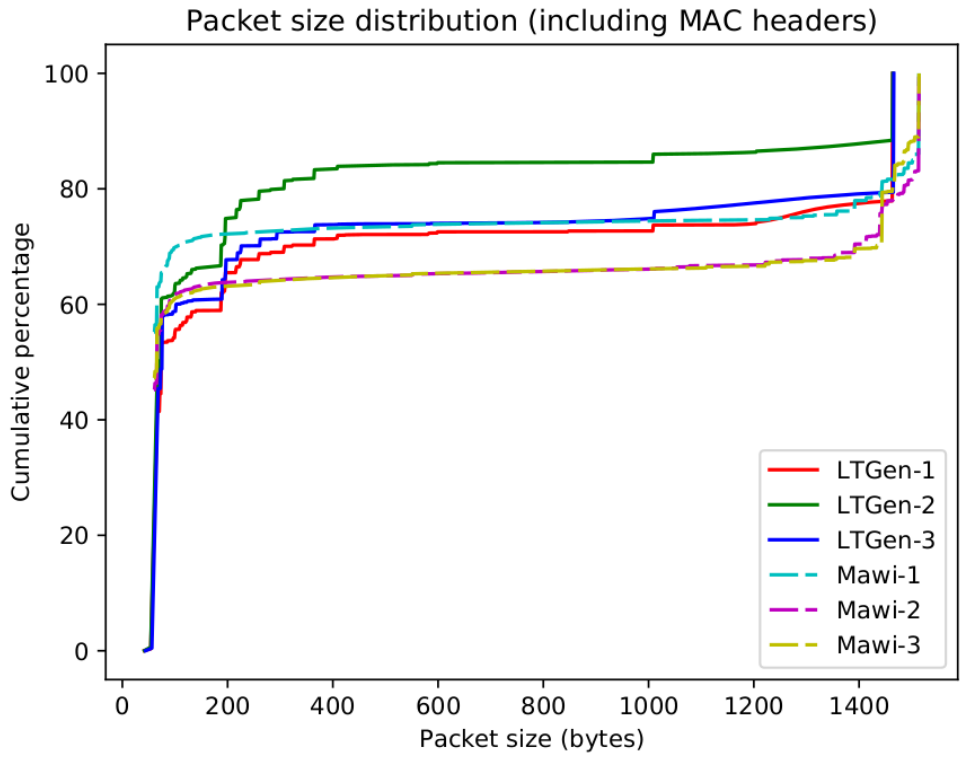


Figure 9: Packet size distributions of MAWILab and LTGen traffic

Regarding the flow level measurements, Figure 8 shows that the TCP sessions are generated at random in both MAWILab and LTGen traffic in all three tests. Figure 9 illustrates the cumulative distribution functions of the packet sizes for both traffic in all three tests. The results demonstrate a similar trend in the distribution of packet sizes but are not completely accurate. Since these characteristics are intrinsic and not taken as inputs for LTGen traffic generation, higher error rates are expected.

### 3.2.6 Logic Bomb Detection (TSOpen) - UNILU

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Standalone command line tool | Check if the tool works properly with right dependencies | Use the tool with the com- mand line |
| **SR2** | Trigger database | Check if the database contains correct triggers | Connect to the database |
| **SR3** | Precise data flow tracking | Manually check reported data flow paths on sample data | Use a benchmark |

Table 8**:** TSOpen - Logic Bomb Detection - Demo scenarios and verification methods

#### 3.2.6.1 Description and goal

TSOpen is a static analysis tool that is able to detect logic bombs in Android applications. Logic bombs are mechanisms used by malicious apps to evade detection techniques. Typically, an

attacker uses a logic bomb to trigger the malicious code only under certain chosen circumstances (e.g., only at a given date) to avoid being detected by the analysis.

The tool can be run by security analysts or directly on the marketplace side. Once the results are available, the analyst can take a decision whether the application should be deployed in a marketplace or not. It can be run over existing applications that could be removed from the marketplace if a logic bomb happens to be found.

### 3.2.6.2 Technical characteristics

TSOpen is developed in Java using existing static analysis framework for Java and Android applications (Soot and Flowdroid) from the state-of-the-art in software engineering,

Those provide the analyst with a precise model of the Android lifecycle and call-backs on which the analyst can run its proper algorithms.

TSOpen combines symbolic execution, block predicate recovery, path predicate recovery, path predicate classification and control dependency to statically detect logic bombs. The tool output the result of the analysis, whether it found a logic bomb or not. In case it finds one, it gives information about its location in the code (class, method, statement, etc.) in order the analyst to manually verify if it is a true positive.

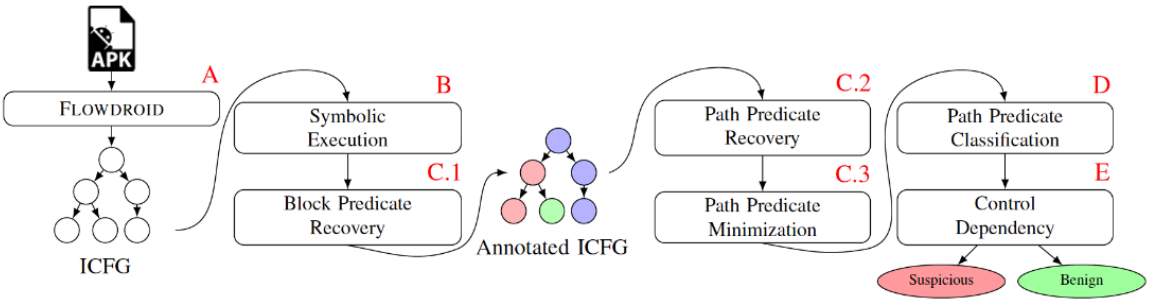


Figure 10: Technical characteristics of TSOpen

The main security item lies in the way conditions are defined as sensitive, here if a variable representing a comparison with at least a TIME, LOCATION or SMS component is found. Then the blocks guarded by the condition are analysed to find at least one method call to a sensitive API.

TSOpen currently faces several limitations such as using non-polynomial algorithms which can lead to applications that cannot be analysed depending on the circumstances. Also, there is no formal definition in the state-of-the-art of what is malicious activity, therefore we used heuristics to try to reach at best a malicious behaviour.

From the results of the tests, we were able to collect logic bombs that we gathered together in a database that can be used as a ground truth for further experimentations.

### 3.2.6.3 Experimental test activities

In order to test TSOpen over a large set of applications, we used the HPC (High Performance Computing) facility at the University of Luxembourg to parallelize a large number of instances. As the tool is able to provide a readable output for humans, when a large-scale study is performed, we needed a simpler way to represent the results to process it later. Therefore, we used a multi-dimensional vector representation that was stored in a REDIS server for future processing.

The item tested are Android applications from different version that were fetched from an application repository available for researchers (Androzoo).

#### 3.2.6.4 Results

In the following figure we present an example of test run in an application containing a logic bomb:

```
↳ java -jar tsopen.jar -f malicious.apk -p ~/git/android-platforms/
TSOpen v1.0 started on Fri Jan 08 12:46:26 CET 2021

[*] Package                          : com.littlephoto
[*] Timeout                          :  60 mins
[*] CallGraph construction           :   6 s
[*] Simple Block Predicate Extraction : 127 ms
[*] Path Predicate Recovery          :   1 s
[*] Symbolic Execution               :   1 s
[*] Potential Logic Bombs Recovery   : 422 ms
[*] Application Execution Time       :   8 s

Potential Logic Bombs found :
- Statement                : if $z4 == 0
- Class                    : com.littlephoto.server.ShortMessageReceiver
- Method                   : onReceive
- Starting Component       : BroadcastReceiver
- Ending Component         : BroadcastReceiver
- CallStack length         : (2)
- Size of formula          : 4
- Sensitive method         : <android.content.BroadcastReceiver: void abortBroadcast()>
- Reachable                : Yes
- Guarded Blocks Density   : 6
- Nested                   : Yes
- Predicate                : (#sms/#body.equals("$$@@&&@@")) (#Suspicious)
-------------------------------------------------------------
```

Figure 11: Example of TSOpen when run on malicious application

We see that one logic bomb was found by the tool; this result can be used to feed the database that centralises all the logic bomb found. For the moment, 32 real logic bombs were found in malicious applications (not in the Google PlayStore).

Here an example of the raw data result that can be processed later:

*f558aa271c4615f0c19df4889b0801d0329aa38d2ab99b3bd666a3b23b07b35b,vercoop.Zionusung.app,0,18,1,0,0,1545064,705,1852,30,7671,8153,20,1730,848,10126,18367*

It is a vector representing data from the application under analysis. The data represent information such as: the hash of the application (sha256), the package name of the Android app, the different time taken for each part of analyses (call graph construction, symbolic execution, predicate recovery, etc.), the number of classes, the size of the app, whether the timeout has been reached or not, etc.

As we use a full static analysis approach, we face the limitations of static analysis, i.e., the false-positive problem (i.e., 17% of false-positives). That is why we had to manually analyse several hundreds of results to filter out false positives to constitute our database.

### 3.2.7 Maude (MAU) - FTS

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1.1** | Use Maude to formally verify platooning modules | • Discover an attack<br>• Evaluate counter-measure | Connected Car (Vertical 1) |

Table 9: Maude - Demo scenarios and verification methods

#### 3.2.7.1 Description and goal

As described in D5.2, the main goal of our development is to enable the formal security verification of cyber-physical systems, in particular, vehicle platooning scenarios. In these scenarios, attackers can cause harm by manipulating the messages communicated between vehicles in a platoon (or interacting with a platoon).

Our model shall support the modelling of cyber aspects, e.g., communication protocols, as well as physical aspects, e.g., vehicle speed and acceleration. Moreover, it shall also support intruder models that can manipulate the available communication channels. Finally, it shall also support automated verification. This is done by relying on the Maude rewriting tool.

#### 3.2.7.2 Technical characteristics

As described in detail in D5.2, we have built on the framework called Soft-Agents that is a Maude implementation framework for cyber-physical systems. However, until now security aspects have not been considered nor vehicle platooning.

We have, therefore, extended the Soft-Agents framework with the sorts, data-structures and models for enabling the formal verification of platooning scenario. These models are detailed in D5.2.

#### 3.2.7.3 Experimental test activities

We have applied our machinery to several platooning scenarios; some of these were taken from the protection profile described in D5.2, as well as new scenario that has been discovered while implementing the models. The experiments are described in this Document under the section for Connected Car.

#### 3.2.7.4 Results

We have applied our formal verification machinery to several scenarios. These are described within this document in Section 4.3.1. In a nutshell, we have validated the countermeasures based on plausibility checks. We have also identified 3 new attacks that have not yet been reported in the literature. These are detailed in this document as well.

### 3.2.8 NeSSoS Risk Asessment tool (RA) - CNR

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | The tool works on-line | Check if the tool is available on-line and provides the risk levels | Use the tool through the web interface |
| **SR2** | (Additional) Security configuration is suggested | Check if the tool is able to provide reasonable suggestions for (additions) security configuration | Use this functionality through the web interface |
| **SR3** | The tool works auto-matically | Check if the tool is able to receive the input from the monitoring module and re-compute risk levels without human intervention | Use the tool through the machine interface |

Table 10: NeSSoS Risk Assessment Tool - Demo scenarios and verification methods

### 3.2.8.1 Description and goal

The NeSSoS risk assessment tool is a framework implemented by CNR. Currently, the NeSSoS tool is integrated into a cybersecurity observatory maintained by CNR [9]. The observatory provides to the general public various cybersecurity services, including the NeSSoS risk assessment tool (named “Self-Assessment tools”).

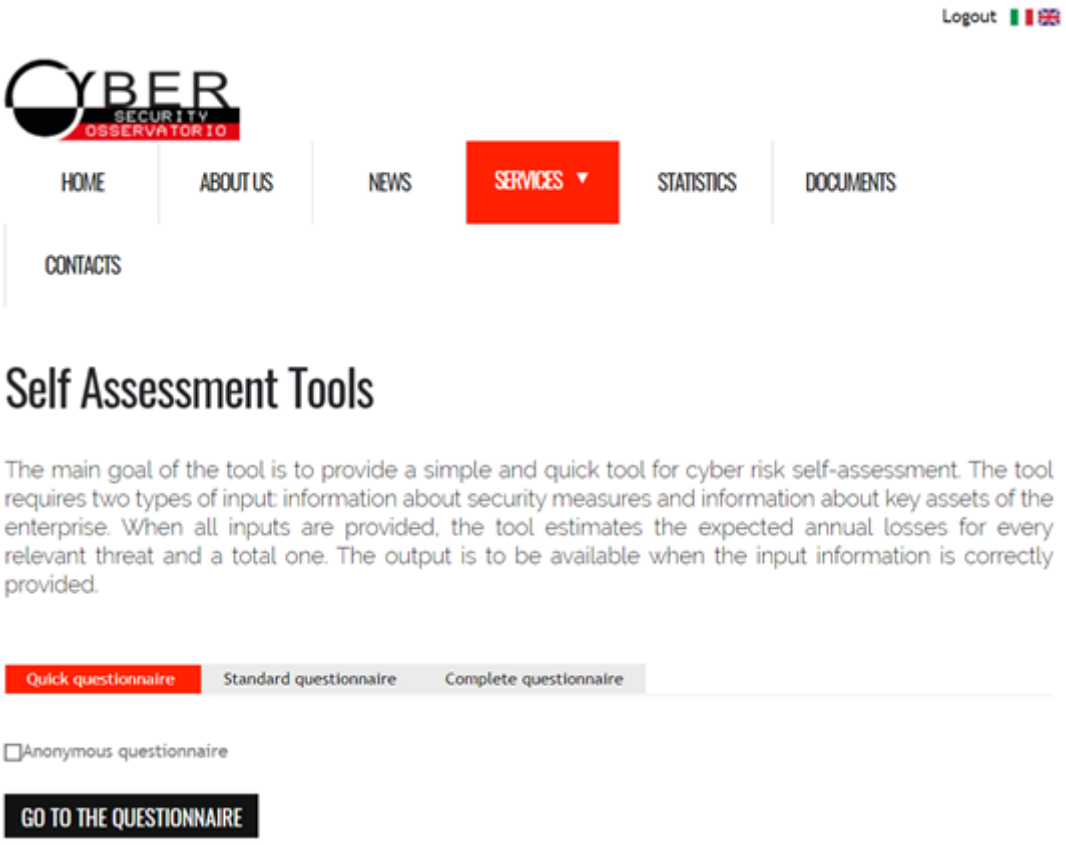


Figure 12: NeSSoS cyber risk assessment tool

NeSSoS risk assessment tool is a simple and relatively fast method for quantitative estimation of annual cybersecurity losses, i.e., risks. The user should provide the information about the system and will get an estimation of the annual losses for various cyber threats. The information required for the analysis is: a questionnaire about cybersecurity practices and security controls implemented and applied in the evaluated system, and the information about the cybersecurity assets which can be damaged by various threats.

The self-assessment tool is currently available in three versions: short, standard and complete. A short version is aimed for a very fast and rough assessment, with a minimal set of questions to be answered. The tool makes the prediction with a typical system (according to the answers) in mind. This version is aimed at those businesses with a standard network configuration, settings, and assets. Naturally, the more the system deviates from the standard settings, the less precise the predications are. Standard and complete versions are aimed for a more detailed analysis. They take into account the peculiarities of the analysed systems, but require more details to be provided as input. Therefore, these versions require more time and knowledge, but are more precise.

### 3.2.8.2 Technical characteristics

The NeSSoS tool is hosted on a server running Ubuntu Server 18.04, located at the CNR in Pisa. The platform offers a website, built on Apache2 Web Server. The web-services available on the website are running over a Tomcat8 instance, running behind Apache2.

The tool is aimed to evaluate cyber risks. The results should help the system owners to plan their cybersecurity strategy and use resources to maximise cyber protection.

The tool is aimed for SMEs, as it provides simple analysis. It is not very customizable to consider the systems deviating a lot from a “common” system type (e.g., using many IoT devices or have a complex network structure). The tool requires inputs with high details, i.e., specific security practices implemented in the system and expected costs in case of compromising assets.

#### 3.2.8.3 Experimental test activities

Internally, the tool is verified by testing input-output dependency. We see that once “more secure” practices are reported, lower risk values for the corresponding threats are obtained. The validity of outputs was checked by comparing the ratio of outputs with the ones reported by various surveys [10][11]. We acknowledge that such test is not precise, as input parameters (i.e., security practices) for the test subjects of the used surveys are unknown (moreover, they range from good ones to bad ones), which does not allow us to mimic their experiments correctly. On the other hand, these are the best available real data for testing.

The tool is available for usage by wide public. The users are free to use the tool, evaluate their systems and provide their feedback. The results of the analysis should be verified by the users if they are in line with their indoor risk assessment. We are going to apply the tool in the eGovernment scenario and evaluate its result against the expectations of the system owners.

#### 3.2.8.4 Results

So far, internal testing shows that the tool works as expected: better security practices lead to lower associated risks.

The external evaluation by practitioners is to be conducted together with the eGovernment scenario owners and reported in the future deliverables.

### 3.2.9 OpenCert (OC) - TEC

OpenCert has been used in its current version and with its current functionalities in the context of Vertical 1, i.e. we have not generated a new prototype of the OpenCert tool.

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Use OpenCert to create an Assurance Case that presents safety and security arguments. | Verification by means of the scenario 4, and also by the CAPE tools integration pipeline. | Connected Car (Vertical 1), scenario 4 |
| **SR2** | | | |
| **SR3** | | | |
| **SR4** | | | |

Table 11: OpenCert - Demo scenarios and verification methods

#### 3.2.9.1 Description and goal

As documented in D5.2 [2], the main goal of the development is to enable OpenCert to give support to the engineers to achieve Safety and Security compliance assessment and certification of the platooning scenario.

#### 3.2.9.2 Technical characteristics

As it was described in D5.2, the OpenCert tool provides support to follow the compliance of a system with the standards by identifying what activities and requirements must be fulfilled for that specific system. It also helps to host the evidences of those requirements that are satisfied.

#### 3.2.9.3 Experimental test activities

The experimental test is fully described in the scenario 4 of the Connected Car case study (see Section 7.5).

#### 3.2.9.4 Results

Our key results are described in this document under the Connected Car case study (see Section 7.5). In summary, we have digitalized the safety and security standards involved in the Platooning use case. In this first iteration we have created an assurance project for each standard to be followed and have added the generated evidences.

### 3.2.10 Project KB (KB) - SAP

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Plain-text format | Integration test according to defined scenario | e-Government (Vertical 2) |
| **SR2** | Digital Signature | | |
| **SR3** | Public and private repositories | | |
| **SR4** | Versioning | | |

Table 12: Project KB - Demo scenarios and verification methods

#### 3.2.10.1 Description and goal

The goal of Project KB is to develop an open, shared and distributed knowledge base with information about security issues of open source projects, including security vulnerabilities or malicious/compromised releases. Compared to existing public vulnerability databases, this information will be code-centric, thus, it lends itself to (automated) code analysis.

#### 3.2.10.2 Technical characteristics

Project KB comprises a data format to express code-centric information about security issues, e.g., fix commits and affected versions in PURL. Project KB furthermore provides the necessary tooling to create, pull, merge and export such statements. Last, it comprises a collection of several hundred YAML statements that have been collected and curated over the course of several years at SAP.

#### 3.2.10.3 Experimental test activities

Project KB will be tested in the context of the e-Government scenario. In particular, it will be tested whether the YAML statements are correctly loaded into Steady's database upon the installation of Steady, and whether any new statements provided in any of the before-mentioned Git repositories will be updated accordingly in delta runs. To this end, a demo scenario will be created that contains one example fix-commit for one of the dependencies of the SAML IdP.

#### 3.2.10.4 Results

Results of the publication and consumption of YAML statements according to Project KB will be provided in D5.4.

### 3.2.11 Risk Assessment for Cyberphysical interconnected infrastructures (MRA) - NCSRD

The MRA development roadmap within SPARTA is to implement its software requirements as described in SR1 starting from Q4 – 2020 and finalising in Q2/Q3-2021 so that it can demonstrated at the project closing stages.

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Stand-alone tool | Check if the MRA tool provides the risk levels as identified. Use the tool through the web interface | Connected Car (Vertical 1) - security profile |

Table 13: Risk Assessment for Cyberphysical interconnected infrastructures - Demo scenarios and verification methods

#### 3.2.11.1 Description and goal

As described within both D5.1 and D5.2 the MRA prototype that has been implemented within the SPARTA project is focused on assessing the attractiveness of infrastructure assets in the cyber-physical domain. The foreseen goal of MRA is to identify the “attractiveness value” of existing infrastructures’ assets-at-risk from security profiles, also employing dynamic information in relevant environments. Target Attractiveness is perceived as a surrogate indicator for the likelihood of an attack. Within MRA is a composite indicator of the perceived value of a target (cyber/physical asset) to the adversary and their degree of interest in attacking the target.

MRA is intended to be implemented in the Vertical 1 scenarios to determine and quantify assets attractiveness and present a relative comparison of different assets.

#### 3.2.11.2 Technical characteristics

Following a high-level description MRA module for cyber-physical attractiveness quantification is currently being developed as a stand-alone tool that could be readily integrated as a service within larger-in-scope cybersecurity solutions. A more detailed description of its technical characteristics will be provided upon further verification and testing as described in D5.2

The analysis conducted by MRA do require the following inputs:

- Manual assessment of the security profile of the cyber-physical system under consideration and recognition of key assets. This could be supported by a Threat Analysis and Risk Assessment (TARA) which may have considered the Hazard Analysis and Risk Assessment (HARA).
- Ingestion of data from relevant demo components in .csv format, which will be further processed and utilized for quantification purposes.

The MRA is currently being developed in the python programming language, after initial layman experimentation in MS-Excel. The later was implemented as a primitive experimentation step to check the correctness of the developed approach.

**Constraint and limitations:** MRA can be used as a stand-alone tool or integrated as a service. The constraints are related to the manual requirement of thoroughly assessing the security profile and if needed re-adjust the attractiveness categories to match application-specific needs. No hardware requirements are present.

### 3.2.11.3 Experimental test activities

The experimental activities were based upon an assessment of the "vertical 1" security profile as presented in D5.1. Section 2.1.3.3 therein introduces the System Assets, which are pertinent to the integrity of the vehicles and their passengers. The identified assets / "surfaces of attack" on the CCCC vertical that may become target of attacks are

- Wireless communication that may provide access to 3rd parties to the vehicle data and poses serious security and safety risks to the vehicle,
- Safety critical functions such as brakes.
- Non-safety critical functions like radio

Asset attractiveness has been identified within the verification framework as a set of complimentary attributes that follow the risk word ladder approach and are introduced below.

**Attribute1:** Impact - defined according to perceived functionality impacts described in section 2.1.1 of SPARTA D5.1.

**Attribute2:** Location – defined following D5.1 – section 2.1.1. Vertical Scenario Conditions.

**Attribute3:** connectivity which is assigned following the DOT HS 812 636 report[6] – Section 3

**Attribute4:** EasinesstoAttack which is related to the CPS security properties, defined in D5.1 – section 2.1.3.

**Attribute5:** DependentElenets which is the endpoint of the impact chain that may occur and is defined according to D5.1 section 2.1.1.

### 3.2.11.4 Results

The following plot is indicative of the results that may be obtained following the MRA approach, and is constructed using MSExcel, as currently the tool is under development. The higher area covered is indicative that the asset is a more attractive target compared to the examined ones.

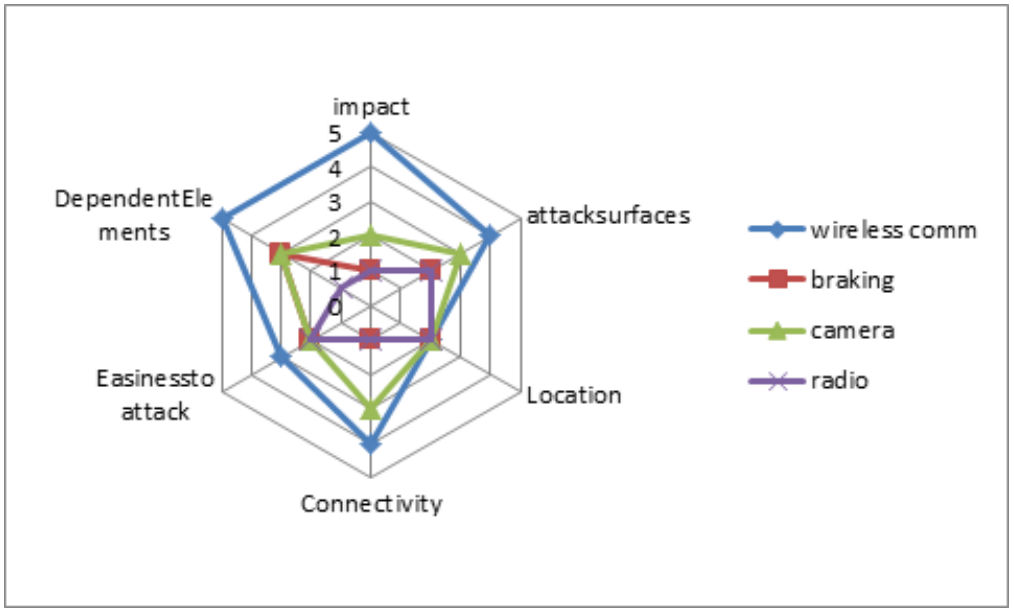


Figure 13: MRA Results

[6] Stachowski, S., Bielawski, R., & Weimerskirch, A. (2018, December). Cybersecurity research considerations for heavy vehicles (Report No. DOT HS 812 636). Washington, DC: National Highway Traffic Safety Administration

## 3.2.12 Sabotage (SB) - TEC

Sabotage has been used in its current version and with its current functionalities in the context of Vertical 1, i.e. we have not generated a new prototype of the Sabotage tool.

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1**<br>**SR2** | Simulation-based Fault injection and analysis of faulty scenarios | Verification by means of the scenario 5. | Connected Car (Vertical 1), scenario 5 |

Table 14: Sabotage – Demo scenarios and verification methods

### 3.2.12.1 Description and goal

As documented in D5.2 [2], the main goal of the development is to enable Sabotage to generate and verify the behaviour of a measure, i.e. sensor-based plausibility check algorithm against different possible attacks.

### 3.2.12.2 Technical characteristics

As it was described in D5.2, we have applied the Simulation-based fault injection technique in the Connected Car case study. It will be used to simulate how a fault, originated from a random hardware fault or cyber-attack, can affect the vehicle behaviour by changing the velocity to an abnormal value.

### 3.2.12.3 Experimental test activities

The experimental test is deeply described in the scenario 5 of the Vertical 1 (see Section 7.6).

### 3.2.12.4 Results

Our key results are described in this document under the Connected Car case study (see Section 7.6). In summary, we have generated a sensor-based plausibility check algorithm and have used the Sabotage tool to verify the behaviour of the algorithm against possible attacks.

## 3.2.13 SafeCommit (SF) - UNILU

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Compute performance scores by leveraging the ground truth | Check if the performance scores are high enough | Deploy SafeCommit and Run on the ground truth |
| **SR2** | Assess SafeCommit in a practical settings | Check if SafeCommit is able to detect vulnerabilities in open source libraries used in Vertical 1 | Deploy SafeCommit and Run on a git Repository of open-source libraries of Vertical 1 |

Table 15: SafeCommit - Demo scenarios and verification methods

### 3.2.13.1 Description and goal

This tool aims at automatically detecting commits that introduce vulnerabilities (we will also refer commit as patches for the sake of simplification) in Continuous Integration Ecosystem. SafeCommit is built on top of AI techniques relying on innovative features and advanced patch representation learning.

Systematically and automatically identifying vulnerability introducing patches once a commit is contributed to a code base is of the utmost importance: (1) To reduce the number of vulnerabilities in a software code base; (2) To incite maintainers to quickly reject the relevant changes. The proposed tool aims at being integrated into real-world software maintenance and usage workflows. The objective is to carry out a live study in order to collect practitioner feedback for iteratively improving the tuning of the research output, towards an effective technology transfer.

### 3.2.13.2 Technical characteristics

SafeCommit will use a machine-learning based approach as described in Figure 14. In particular, SafeCommit will address a binary classification problem of distinguishing vulnerability introducing patches from other patches. As any classification problem, well-labelled datasets are more than welcome. To develop SafeCommit, the first main step will consist in building such datasets ("Ground Truth" in Figure 14). Then, we will investigate the possibility to consider a combination of text analysis of commit logs and code analysis of commit changes diff to catch security patches. To that end, the idea is to proceed to the extraction of "facts" from both text and code, and then perform a feature engineering by assessing the efficiency of the proposed features for discriminating security patches from other patches ("Features Set" in Figure 14). Then, we will build a prediction model ("Classifier" in Figure 14) using machine learning classification techniques.

As an add-on, we will investigate a specific learning approach named Co-Training, which has shown convincing results in situation where the training datasets are un-balanced. Finally, one major success criteria of SafeCommit is its ability of supporting the work of developers/maintainers in distributed software development. Once prediction models are learned, we will assess their efficiency by performing extensive empirical studies in the real development environments.

| ID | code-fix | ID | security-sensitive |
|---|---|---|---|
| F1 | #commit files changed | S1 | #sizeof added |
| F2 | #loops added | S2 | #sizeof removed |
| F3 | #loops removed | S3 | S1-S2 |
| F4 | F2-F3 | S4 | S1+S2 |
| F5 | F2+F3 | S5-S6 | Like S1-S2 for continue |
| F6-F9 | Like F2-F5 for *if* | S7-S8 | Like S1-S2 for break |
| F10-F13 | Like F2-F5 for Lines | S9-S10 | Like S1-S2 for INTMAX |
| F14-F17 | Like F2-F5 for Parenthesized expression | S11-S12 | Like S1-S2 for goto |
| F18-F21 | Like F2-F5 for Boolean operators | S13-S14 | Like S1-S2 for define |
| F22-F25 | Like F2-F5 for Assignements | S15-S18 | Like S1-S4 for struct |
| F26-F29 | Like F2-F5 for Functions call | S19-S20 | Like S1-S2 for offset |
| F30-F33 | Like F2-F5 for Expressions | S21-S24 | Like S1-S4 for void |

| ID | text |
|---|---|
| W1-W10 | Most recurrent top 10 word |

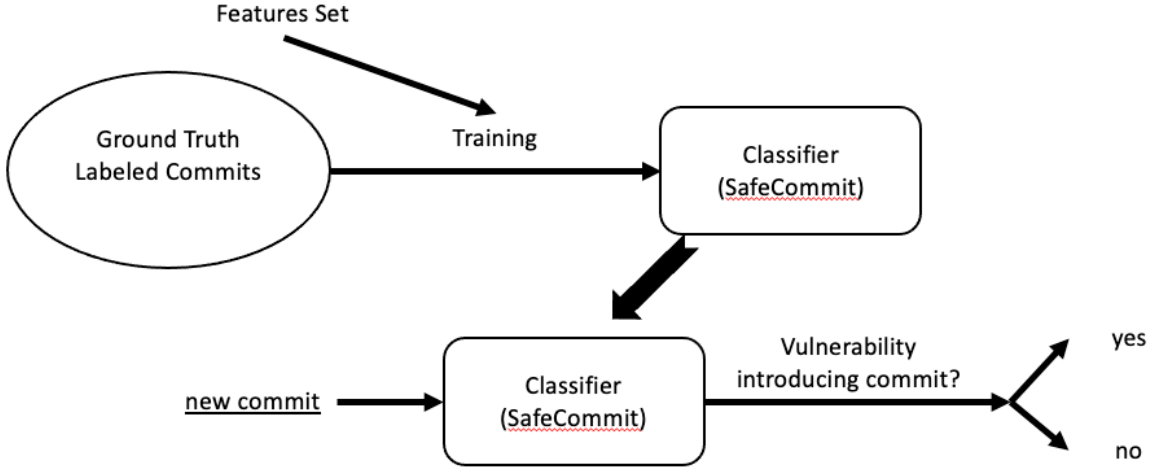


Figure 14: Overall SafeCommit Process

#### 3.2.13.3 Experimental test activities

For the test activities, we first compute performance scores by leveraging the ground truth. The input is the ground truth (I.e., the labelled commits), and the output is the classification performance score. Regarding the test procedure, we follow a classical machine-learning assessment process. We will consider for instance using ten-folds cross validation and compute precision, recall and F1 metrics.

Second, we will assess SafeCommit in a practical setting. As input, we consider large open-source repositories such as Linux. As output, we will yied an assessment report on this "in the wild" experiment. Regarding the test procedure, by considering commit history from large open-source repositories, we will mimic the behaviour of software developers, i.e., we will check if at the time of a commit, this commit can be detected as vulnerability introducing commit.

#### 3.2.13.4 Results

***Regarding the computation of performance scores by leveraging the ground truth:***

To perform this experiment, we use a dataset containing both vulnerability introducing commits (positive samples) and vulnerability fixing commits (negative samples). This dataset is then split into training and test sets. The number of commits per set is detailed in Table 16 .

| | Training | Test |
|---|---|---|
| Vulnerability Introducing Commits (VIC) | 470 | 253 |
| Vulnerability Fixing Commits (VFC) | 389 | 879 |

Table 16: SafeCommit - Dataset Description

By learning on the commits from the training set, SafeCommit tries to differentiate both types of commits in the test set. The performances of SafeCommit are presented in Table 17.

| | True Positive | False Positive | False Negative | True Negative | Precision | Recall |
|---|---|---|---|---|---|---|
| SafeCommit | 58 | 29 | 178 | 850 | 0.67 | 0.245 |

Table 17: SafeCommit - Performance Score

As we can see in Table 17, there is still room for improvement.

**Regarding the assessment of SafeCommit in a practical setting**, we have not yet performed the experiment, This experiment will be conducted between M18 and M36.

### 3.2.14 SideChannelDefuse (FS) - CNIT

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Patch Linux Kernel in order to start the tool. | Check if the tool is correctly installed and running in the kernel. | Stand Alone |
| **SR2**<br>**SR3** | Continuously scan and eventually mitigate the applications behaviour in the system. | Check if the tool correctly reports on going side-channel attacks and mitigate their behaviour. | Stand Alone |

Table 18: SideChannelDefuse - Demo scenarios and verification methods

As stated in the D5.2, SideChannelDefuse is being developed as a standalone tool within CAPE, and hence does not interact with other components during verification and validation. The tool is preinstalled at the kernel level and it continuously monitors processes activities. By doing this it can detect cache attacks but also deploy mitigation strategies on the fly. It follows that rather than integrating it in a pipeline, the most obvious way to use it is to install it and then run other tools on top of the patched kernel.

#### 3.2.14.1 Description and goal

SideChannelDefuse is a tool for continuous assessment and reactive mitigation against side-channel attacks.

If the tool detects that the system is vulnerable, it can activate a continuous kernel-level system-wide detection mechanism which allows detecting whether some application (also running in a virtual machine) is carrying out a side-channel attack. This detection is *continuous*, in the sense that the (host) operating system kernel based detection mechanism is always on, while introducing a minimal overhead in the system. It is *system-level*, in the sense that it monitors all applications running in the system.

If the tool detects that a (virtualized) application is trying to carry out a side-channel attack, that application is deemed as *suspected*. At this stage, the tool can activate per-application mitigation mechanisms, the goal of which is to reduce the likelihood that the application can exfiltrate data using the attack.

The overall resulting tool is able to detect Foreshadow-VMM attacks, as well as other attacks such as meltdown, spectre, or XLate-family attacks.

The tool will be distributed as open-source software. The public repository is not yet available.

#### 3.2.14.2 Technical characteristics

The tool is developed as a patch to the Linux kernel. In our implementation, we have targeted the Intel architecture, considering its widespread adoption and the fact that it has been repeatedly subject to multiple attacks in the last years. Nevertheless, as we discuss, our reference implementation can be easily ported to other architectures, such as AMD. The cloud owner/mantainer has to load into the host system the patched kernel, and let it run. The tool continuously runs in the background. It does not need any further user input or interaction from other sources. In order to execute its detection activity, it relies on the use of Performance Monitoring Units (PMUs) which are equipped with modern CPUs in order to profile the performance or (to some extent) the behavior of applications. PMUs are composed basically of programmable Performance Monitor Counters (PMCs) also referred to as Hardware Performance Counters (HPCs). The detection mechanism is system-wide in the sense that the tool does not make any assumption on which process is the attacker and which is the victim. Since the detection can be fallible, the tool does not take any destructive action with respect to the running process. Rather, it couples the detection with mitigation actions automatically enforced by the operating system as soon as a process is suspected as malicious by entailing a limitation in the scheduler freedom at deciding what CPU resources should be assigned to some process, or the selective (per-process) activation at runtime of security patches against transient-execution vulnerabilities (such as KPTI). At the end, the tool returns some output to /proc/pid.

#### 3.2.14.3 Experimental test activities

As already anticipated in D5.2, to make what follows self-readable, we here provide a brief description of our experimental methodology. Our detection mechanism, as well as the aforementioned mitigations, have been implemented at kernel-level in Linux, and has been exercised on multiple processors of the x86 family. We have used our patched kernel for a month, also in daily usage, both on laptops and on server machines. No false-negative has been observed under that daily usage workload. Of course, this is not a guarantee that our approach could be used to enforce

more intrusive policies with respect to suspected processes like, e.g., killing suspected processes. Rather, it is an indication of the viability of using HPCs as building blocks for articulated detection mechanisms, and for devising strategies where the setup of security-oriented patches can be put in place on a dynamic and per-process basis—rather than paying the cost of these patches by default when any process is active.

We started preliminary testing considering an attacker trying to carry out a cache-based attack and leak information from a co-located victim on the same platform. The attacker is thus sharing some architectural components with the victim, such as the First-Level Cache (L1) or the Lowest-Level Cache (LLC). We do not make assumptions on the privileges with which the attacker is running, nor on whether the side channel is being used to extract leaked information. Indeed, as mentioned, we are interested in detecting the usage of a cache side-channel to leak information while the attack is in progress.

We have carried out an experimental assessment, relying on multiple generations of Intel CPUs, using the following processors:

i5-8250U 4x (SMT) L1 64KB (I,D) 8-way, L2 256KB, shared L3 6MB 12-way;

i7-7600U 2x (SMT) L1 64KB (I,D) 8-way, L2 256KB, shared L3 4MB 16-way (with TSX);

### 3.2.14.4 Results

From Figure 15 we can see how the metrics works in a generic way using (for example) an X-Late attack. The green part is the footprint on the cache architecture of an attacker. There is a kind of continuous pattern, which is quite constant while running the attack. By looking how the profile of the cache usage by the application changes over time, we can detect a process as suspected or not.

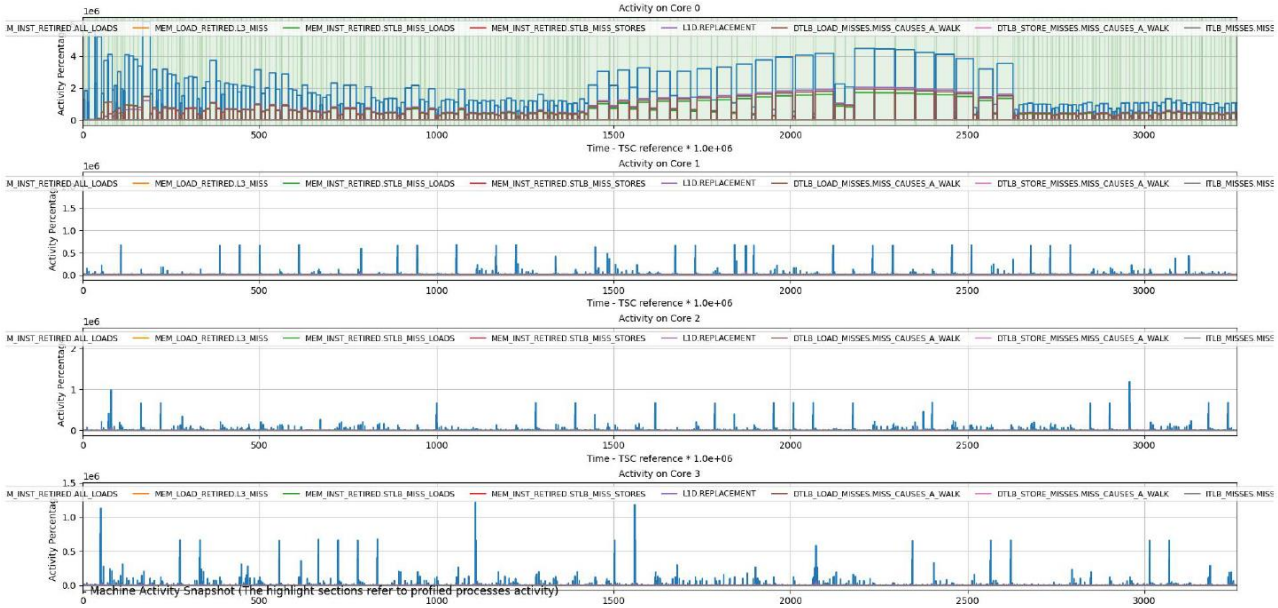


Figure 15: Example of a X-Late Attack

We can already define what kind of attacks we are capable of detecting and mitigating in our experimental setup (Figure 16). We still have to refine the metrics that we use, so as to capture a wider spectrum of attacks.

| Name | Same-Core | Cross-Core | Shared Memory | Measurement | Detected |
|---|---|---|---|---|---|
| EVICT + TIME [55] (taken from [6]) | ✓ | ✓ | ✗ | time | ◐ |
| PRIME + PROBE [41, 57] (taken from [7]) | ✓ | ✓ | ✗ | time | ● |
| PRIME + ABORT [24] (taken from [7]) | ✗ | ✓ | ✗ | TSX | ● |
| FLUSH + RELOAD [94] (taken from [7]) | ✓ | ✓ | ✓ | time | ● |
| FLUSH + FLUSH [30] (taken from [7]) | ✓ | ✓ | ✓ | time | ○ |
| XLATE + TIME [82] (taken from [7]) | ✓ | ✓ | ✓ | time | ● |
| XLATE + PROBE [82] (taken from [7]) | ✓ | ✓ | ✓ | time | ● |
| XLATE + ABORT [82] (taken from [7]) | ✓ | ✓ | ✓ | TSX | ● |
| *Meltdown* (taken from [8]) | | | | | ● |
| *Spectre* (taken from [9]) | | | | | ● |
| *Foreshadow* [81, 92] (taken from [15]) | | | | | ● |

●: yes; ◐: depends on variant; ○: no (additional metrics are required);

Figure 16: SideChannelDefuse list of detections

Detailed results for performance, false-positive and further detection/mitigation mechanisms will be described in D5.4.

### 3.2.15 Steady (VA) – SAP

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Comparison of Java source code and bytecode | Check if an unclassified finding can be resolved through the execution of the new plugin goal “checkcode” | e-Government (Vertical 2) |
| **SR2** | Light-weight scan client | Run light-weight Docker Compose environment and monitor resource consumption and performance | Independent |
| **SR3** | Shared vulnerability database | Check the initial and delta load of vulnerabilities from Project KB into Steady’s database | e-Government (Vertical 2) |

Table 19: Steady - Demo scenarios and verification methods

#### 3.2.15.1 Description and goal

This prototype will demonstrate the integration of Eclipse Steady into CI/CD pipelines of a given application or project.

With every commit or at periodic times, Eclipse Steady will check whether upstream open-source components of the given project are subject to known vulnerabilities. If so, Steady will support the assessment and mitigation of those findings.

#### 3.2.15.2 Technical characteristics

SR1: The bytecode comparison is made available as a separate Steady analysis goal, which can be invoked using Steady's Maven plugin or CLI. Typically, it will be invoked on-demand if previous analyses raised unqualified findings, thus, findings where it could not be determined whether the Java method or constructor contained in an application dependency contains the vulnerable or the fixed body.

Upon execution, the Java bytecode in question is decompiled and transformed into an abstract syntax tree (AST), which is compared with the corresponding ASTs coming from archives that are known to be vulnerable or fixed. If AST equality can be established, the archive in question is qualified accordingly.

The implementation as separate analysis goal gives greater flexibility to the user, who can decide whether to invoke this relatively expensive creation and comparison of potentially many ASTs.

One limitation relates to the chosen decompiler, a 3rd party component, which fails to produce correct Java class names for nested classes.

Another limitation is that questionable Java code is compared against qualified archives existing in a given instance of the Steady backend. If no or few archives have been assessed, the likelihood increases that no equality can be established at all. In other words, the success rate of the implementation depends on the number of archive assessments in the respective Steady backend. However, also note that there are several automated assessment strategies in place, which populate the database over time.

SR2: The light-weight scan client will consist of a reduced version of the Docker-Composer environment for server-side operation, where unnecessary containers will be removed.

SR3: The shared vulnerability knowledge base from Projekt KB, which consists of YAML files (statements) distributed across multiple Git repositories, will be loaded into Steady's vulnerability database. To this end, a new module kb-importer has been developed, which processes the data produced by Project KB and uses Steady's RESTful API for the actual database loading.

#### 3.2.15.3 Experimental test activities

Both SR1 and SR3 will be tested in the context of the SML IdP software of the e-Government vertical. To this end, Eclipse Steady will be installed in a dedicated infrastructure provided and managed by CINI/FBK, such that GitLab build jobs can trigger Steady's client-side Maven plugin, which in turn interacts with Steady's server-side components (cf. Section 8.3.2).

SR2 will be tested independently from any of the SPARTA CAPE verticals. To test the footprint of the local, light-weight scan client, it is planned to measure and compare the resource consumption and performance (esp. at start-up) of the classical and the newly developed Docker-Compose environment.

#### 3.2.15.4 Results

Results of the analysis of the SAML IdP will be provided in D5.4.

### 3.2.16 SysML-Sec/TTool (TTOOL) - IMT

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Use TTool to verify the platooning system at a high level of abstraction | • Capture the digital platform at a high-level of abstraction<br>• Look for possible attacks with formal verification<br>• Study countermeasures | Connected Car (Vertical 1) |
| **SR2** | Use TTool to verify the platooning system taking into account both software and hardware aspects | • Capture the digital platform at a high-level of abstraction<br>• Look for possible attacks with formal verification. This security verification takes into account hardware aspects (e.g. access to buses, firewalls, etc.)<br>• Study countermeasures | Connected Car (Vertical 1) |

Table 20: SysML - Sec/TTool - Demo scenarios and verification methods

#### 3.2.16.1 Description and goal

TTool is a UML/SysML tool dedicated to the design and verification of embedded systems. TTool is free and open source.

TTool will be used to demonstrate its ability to verify safety and security properties on high-level model of embedded architectures. To do this, we will rely on the SysML-Sec framework supported by TTool.

- We will first capture a subset of safety and security requirements
- We will also model except of fault and attack trees
- Then, we will proceed to the model of the hardware and software architecture of the system. TTool follows the Y-Chart, that is, functions are modeled independently from the hardware architecture before they are mapped to the architecture. We will experiment with different architecture and see which ones verify safety and security requirements while ensuring good performance.

#### 3.2.16.2 Technical characteristics

Currently, TTool can support the verification of security properties for simple cryptographic protocols. Since TTool also support hardware models, we intend to improve its ability to capture hardware security aspects such as firewalls, mapping of keys, TPMs, etc.

#### 3.2.16.3 Experimental test activities

We intend to use TTool for the Rover use case, taking into account at least one of the two systems (Tecnalia or Fortiss system). We will test that TTool is adapted to model at least one of them and to prove safety and security aspects. Extensions will probably have to be performed to fully support the

hardware aspects and security properties (TTool supports only confidentiality and authenticity properties).

Last but not least, we intend to interact with the AutoFOCUS 3 tool is order to benefit from their fault and attack trees capabilities.

#### 3.2.16.4 Results

Not (yet) applicable.

### 3.2.17 VaCSInE (VCS) – CETIC

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Orchestration of the security policy | Scan for vulnerabilities after the security remediation | Connected Car (Vertical 1), scenario 2 |
| **SR2** | Observability of the security policy orchestration | Remediation logs are available | Connected Car (Vertical 1), scenario 2 |

Table 21: VaCSInE - Demo scenarios and verification methods

#### 3.2.17.1 Description and goal

In this prototype, we will demonstrate how the security requirements of a Cloud/Edge system can be continuously assessed in a vehicle platooning context.

The security requirements for different parts of the Cloud/Edge can be dynamic, for example, when an incident happens in a part of the edge infrastructure, it needs to adapt its local security zone policy, possibly impacting the certification.

The purpose of the demonstrator is to show how to orchestrate the automated deployment and configuration of security services (firewall, honeypot, etc.) based on security requirements. To do this, we will:

- first define security requirements for a Cloud/Edge infrastructure in the context of vehicle platooning: this is done using the Security Content Automation Protocol (SCAP);
- then we will automate the application of the security policy to various parts of the system: VaCSIne will deploy and configure various security services to satisfy the policy;
- to check that the policy is continuously satisfied, we will run vulnerability scans: their output, coupled with the VaCSIne remediation logs can then be used as compliance evidence.

#### 3.2.17.2 Technical characteristics

The federated security controller and the security agents of VaCSIne are developed as Python microservices, for modularity and scalability reasons. The orchestration of the remediations is based on Kubernetes and Ansible, where security services are deployed as containers in Kubernetes based on Helm charts and Ansible playbooks are run to apply remediations. To extend the orchestration of security services to the Edge, the prototype relies on the KubeEdge project built upon Kubernetes. Monitoring of the system security properties and of the remediation logs is achieved with the help of Grafana dashboards, Grafana Loki and the Prometheus data source. The DevSecOps aspects of the prototype are implemented using GitLab-CI to orchestrate the continuous integration and deployment of the security services and playbooks. The security requirements are described using

the SCAP format and provided as input to the federated security controller, they will be exploited by the vulnerability analysis performed by OpenSCAP and the remediations by the VaCSIne security agents.

### 3.2.17.3 Experimental test activities

The experimental environment is composed of a private Cloud on which we deployed a Kubernetes cluster as a container orchestration engine, the Edge infrastructure with KubeEdge deployments and rover platoon. In the first iteration, we deployed the platoon on the test infrastructure with a default security policy and performed a vulnerability scan to check the default security policy is verified on the platoon. We then let the platoon drive between edge infrastructure nodes having different zone security policies. Figure 17 provides an excerpt of a sample SCAP security policy where the presence of a firewall is required. Those tests provided us with vulnerability scan reports and security remediation logs that can be correlated and used as input for further compliance or operations activities in a continuous way.

```
<oval-def:definition class="compliance" id="oval:ssg-service_firewalld_enabled:def:1" version="1">
  <oval-def:metadata>
    <oval-def:title>Service firewalld Enabled</oval-def:title>
    <oval-def:affected family="unix">
      <oval-def:platform>Fedora</oval-def:platform>
    </oval-def:affected>
    <oval-def:description>The firewalld service should be enabled if possible.</oval-def:description>
    <oval-def:reference ref_id="service_firewalld_enabled" source="ssg"/>
  </oval-def:metadata>
  <oval-def:criteria comment="package firewalld installed and service firewalld is configured to start" operator="AND">
    <oval-def:criterion comment="firewalld installed" test_ref="oval:ssg-test_service_firewalld_package_firewalld_installed:tst:1"/>
    <oval-def:criteria comment="service firewalld is configured to start and is running" operator="AND">
      <oval-def:criterion comment="firewalld is running" test_ref="oval:ssg-test_service_running_firewalld:tst:1"/>
      <oval-def:criteria operator="OR" comment="service firewalld is configured to start">
        <oval-def:criterion comment="multi-user.target wants firewalld" test_ref="oval:ssg-test_multi_user_wants_firewalld:tst:1"/>
        <oval-def:criterion comment="multi-user.target wants firewalld socket" test_ref="oval:ssg-test_multi_user_wants_firewalld_socket:tst:1"/
      </oval-def:criteria>
    </oval-def:criteria>
  </oval-def:criteria>
</oval-def:definition>
```

Figure 17: Sample security policy – Extract of SCAP content to check that a firewall is enabled on the target

### 3.2.17.4 Results

Test results for the prototype include execution logs of the platoon, edge and security orchestration, Figure 18 provides an example of logs produced by a test where a platoon changes from one edge to another: depending on the changing security policy, vulnerability tests are automatically run, security remediation is executed by VaCSIne and another vulnerability scan confirms that the remediation managed to apply the security policy.

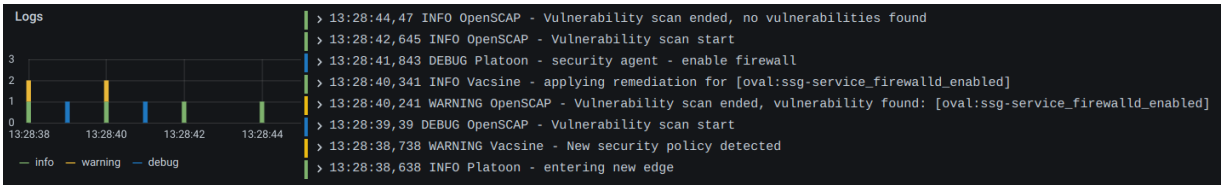


Figure 18**:** Sample vulnerability scanning and associated remediation logs in Grafana dashboard

Figure 19 provides a sample HTML report generated by OpenSCAP after a remediation that activated the firewall. We can see that the firewall check passed, that to satisfy the policy the firewall default ruleset needs to be stricter and that some network protocols need to be disabled.

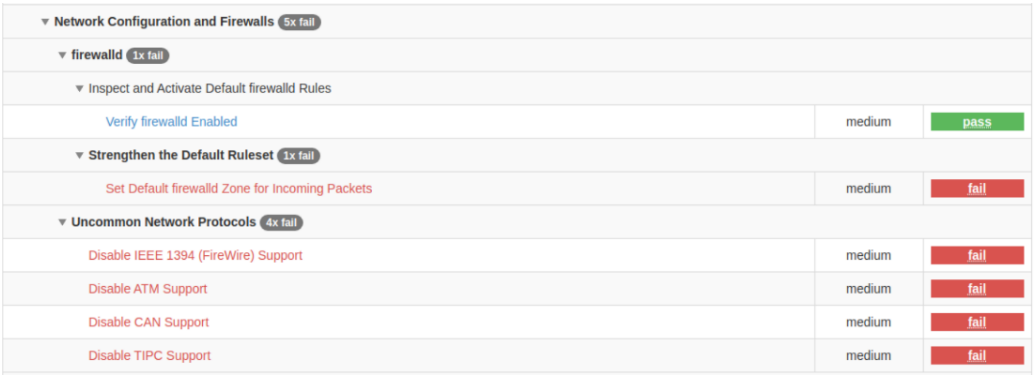


Figure 19: Sample OpenSCAP report

## 3.2.18 Visual Investigation of security information (VI) - UKON

| SR id | Description | Verification method | Demonstration scenario |
|---|---|---|---|
| **SR1** | Web Application Prototype | Display projects developed in software organizations (e.g., Eclipse Foundation) | Visually investigate the Eclipse Foundation projects |

Table 22: Visual Investigation of security information - Demo scenarios and verification methods

The demonstrator for the visual investigation (VI) of security information allows exploring the exposure of software projects from an organizational point of view. The developed vulnerability explorer allows us to visually assess the number of vulnerabilities in projects and their organization-wide dependencies among the components.

### 3.2.18.1 Description and goal

The primary purpose of the demonstrator for the use case visual investigation of security information is to get insights about the consumption of exposures in open-source components on the level of a whole software development organization. The implemented interface supports to discover various security-relevant information, for instance, the most/least-used open source components, the most/least-vulnerable open source components, the most/least-vulnerable applications, or most/least-relevant vulnerabilities.

### 3.2.18.2 Technical characteristics

The demonstrator uses the API of the Eclipse Steady from SAP (see Sec. 3.2.15) and uses the results of individual package scans to highlight the security status of whole software organizations. The backend of the vulnerability explorer is implemented in Java and uses modern web technologies such as the JavaScript library D3 (Data Driven Documents) to display the whole software organization. The user interface allows us to investigate the results of the scans and their internal dependencies as a directed acyclic graph (DAG). For the development of the demonstrator, we crawled all open-source Java projects of the Eclipse Foundation and scanned the projects using Eclipse Steady. The main input files for the demonstrator are Java Maven projects. The user interfaces are shortly described in the following results subsection. A limitation of the approach is that it only enables us to assess the exposure vulnerabilities in larger software organizations. The individual vulnerabilities cannot be resolved with the demonstrator.

### 3.2.18.3 Experimental test activities

We developed the demonstrator using data from the crawled open-source projects from the Eclipse Foundation. Further, the demonstrator was presented to potential users at SAP to collect feedback and improve the interface. We conducted interviews with think-aloud protocols to capture the requirements and suggestions of software developers to validate the objectives of the demonstrator and the designed interface. The outputs of the demonstrator are shown in the results section.

### 3.2.18.4 Results

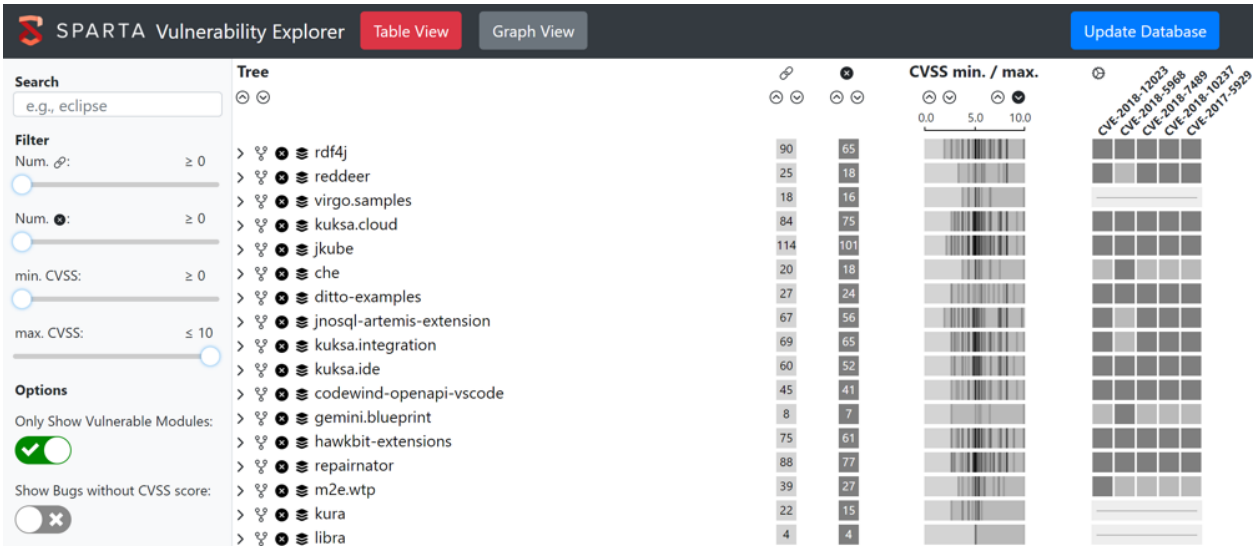


Figure 20: The Tree View of the Vulnerability Explorer

**Filter panel:** The system allows for the search and filtering of repositories and modules. The filter panel is located on the left of the screen (see Figure 20). The table can be filtered by, the number of dependencies, number of CVEs, the minimum CVSS score, the maximum CVSS score. In the search field, you can search for a specific repository name. This allows for the location of the most vulnerable repository or module. Additionally, options for showing modules without vulnerabilities and CVEs without CVSS score. These options allow for a more detailed insight into the project structure and software bugs that are only functional and do not pose a security risk.

**Tree view:** The system allows for the exploration of repositories, modules, libraries, and bugs through a tree table. The tree view is located on the right of the screen (see Figure 20). The vulnerability information is displayed in the columns. The “Dependencies” column, indicated by the chain-link icon, shows the number of dependencies for a repository or module. The table can be sorted by the number of dependencies by clicking one of the arrow buttons in ascending and descending. The “Errors” column, indicated by a dark circle with a cross, shows the number of errors for a repository, modules, or libraries. The table can be sorted by the number of errors in the same way as the “Dependencies” column. The “CVSS min./max.” column visualizes the CVSS score of a repository, modules, files, or bugs in a heatmap visualization. A line represents the presence of a bug with a given CVSS score, the darker the line the more CVEs with this score are present in the software artifact. A lighter shade of gray highlights the range. The table can be sorted by the minimum or maximum CVSS score by clicking one of the arrow buttons in ascending and descending order. The “Top-Bugs” column, the right-most column, visualizes the presence of a specific Bug. By default, these bugs are the top-5 bugs by their number of occurrences. However, the list can be customized by clicking the gear symbol, where the user can search, add, and remove specific CVEs. This column allows for a fast inspection of those bugs.

**Tree hierarchy:** The lowest level represents a repository. It can be expanded to view errors and modules. The presence of errors is visualized on the left side of the repository symbol. The table entries show the number of errors, a visual representation of the CVSS scores, and whether a “top bug” is present. The CVEs are always displayed above the modules and can be expanded to reveal buggy libraries. Modules are separated from libraries to allow for an easier distinction. You can expand the errors row to reveal the relevant buggy libraries. The same information as for repositories and modules is available for libraries. Expanding a library reveals the relevant bugs for the selected repository or module, so bugs that affect the security of the repository or module. The CVSS score is shown in the appropriate column to allow for risk assessment. A bug is clickable and links to its entry on nvd.nist.gov.

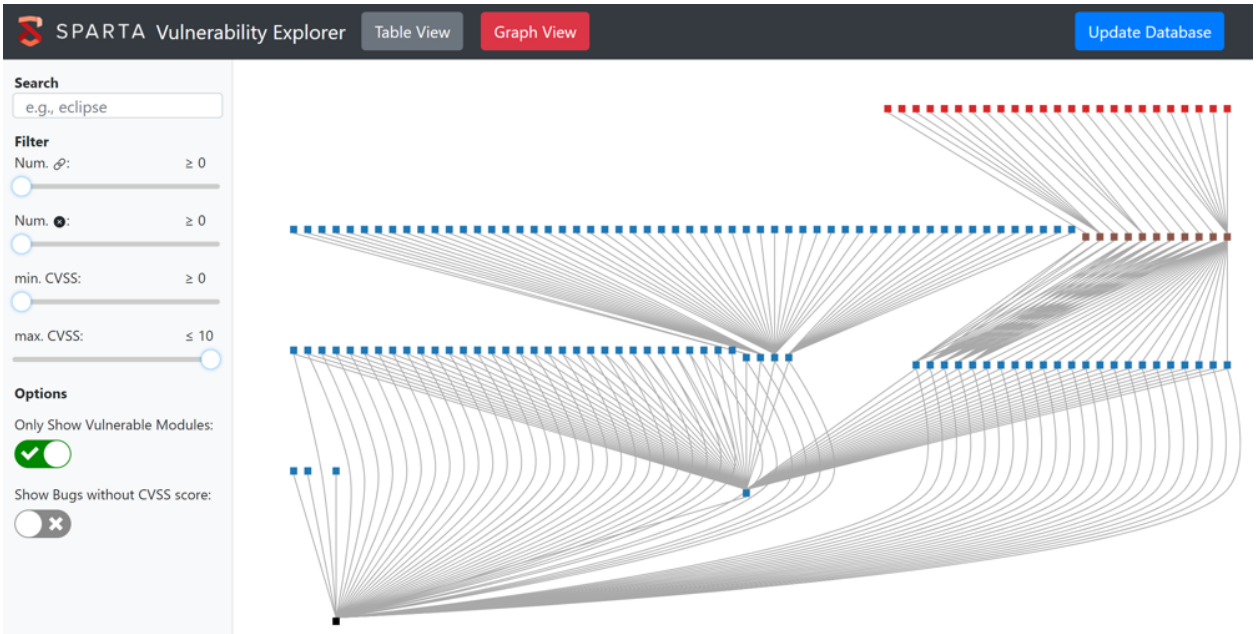


Figure 21: The Graph View of the Vulnerability Explorer

**Graph view:** The Graph view shows the dependency structure of a specific repository or module. It serves as an overview of a specific software artifact. The dependency graph is a directed acyclic graph. A repository is shown as a black square. A module is visualized as a blue square. A library is shown as a green square A CVE is shown as a red square. A mouse-over pop-up reveals detailed information about the specific software artifact. An item dependency is shown as a line where the lower items depend on the ones on top. Figure 21 shows an example of the reddeer repository (see https://github.com/eclipse/reddeer).

# Chapter 4 Prototypes for Convergence of Security and Safety (T5.2)

## 4.1 Introduction

This section describes the implementation efforts carried out by T5.2 for the Convergence of Security and Safety. The goal of this task is to advance the techniques and tools for the integration of safety and security. This is particularly important given the increased interconnectivity of safety-critical systems, such as industry 4.0 applications and autonomous cars. This means that if adequate countermeasures are not put in places, attackers can exploit the increased attack surface to cause harm, such as accidents, by disabling, for example, safety features.

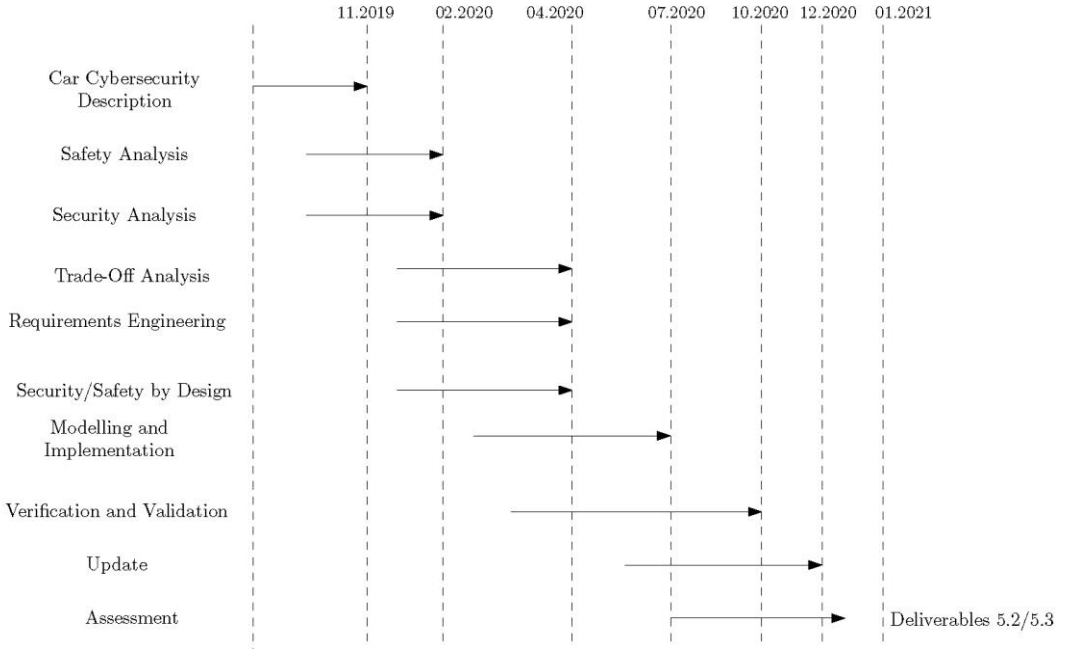


Figure 22: Roadmap for Task 5.2 Activities

In D5.1, the roadmap has been laid out which is depicted by Figure 22. The following activities have been carried out and are described in D5.2: The roadmap starts with the description of connected car cybersecurity (Vertical 1). From the identified scenarios, safety and security analysis have been carried out by extending existing machinery, such as Goal Structure Notation, KAOS models, and Attack Defense Trees. These analyses have led to requirements, written as a Protection Profile used by Common Criteria and machinery developed for carrying out trade-off analysis. Finally, formal verification techniques have been used to support security by design approach for some of the identified scenarios.

We detail in the following sections the activities carried out until the time of the deliverable related to the Modelling and implementation, Verification and Validation, Update, and Assessment.

## 4.2 Modelling and Implementation

AutoFOCUS3 is a model-based engineering tool for safety-critical embedded systems. AutoFOCUS3 supports the design, development and validation of safety-critical embedded systems in many development phases, including architectural design and implementation [6]. We describe here a brief overview of how to use AutoFOCUS3 for developing component architectures, validating

them by means of simulation and generating C code from the defined architecture. For more details, we refer the interested reader to [7].

AutoFOCUS3 allows one to define datatypes and functions by using a so-called Data Dictionary. More specifically, it allows one to define enumeration datatypes, structure datatypes, arrays, and functions. Datatypes, arrays, and functions can be used when specifying the behaviour of component architecture.

Architectural-wise, AutoFOCUS3 supports the creation of components as well as channels between components. Figure 23 illustrates two components in yellow, named as ComponentA and ComponentB. The black arrows denote two channels connecting the outputs of ComponentA to inputs of ComponentB. Channels are connected to output and input ports, illustrated by the small black and white cycles on the edge of components. Components may be declared as weakly or strongly casual. Weakly casual components instantaneously react to a value that arrived from an input channel (i.e., at the same time step), whereas strongly casual components only react to an input value at the next time step.

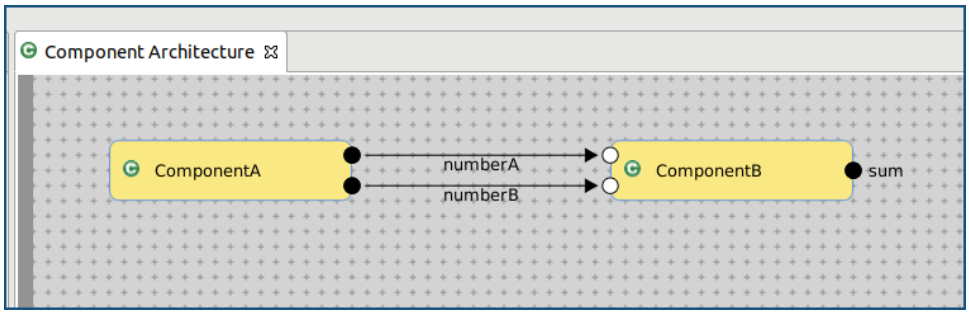


Figure 23: AF3 - Component architecture

AutoFOCUS3 provides two ways to specify the behaviour of components, namely code specification and automaton specification. Code specification allows one to specify the behaviour of components on code level in a C-like language. Figure 24 illustrates the code specification view from AutoFOCUS3 for ComponentB. It calculates the sum of two integers (numberA and numberB) received as inputs from ComponentA. The result of this computation is assigned to the integer sum, which is the output port of ComponentB.

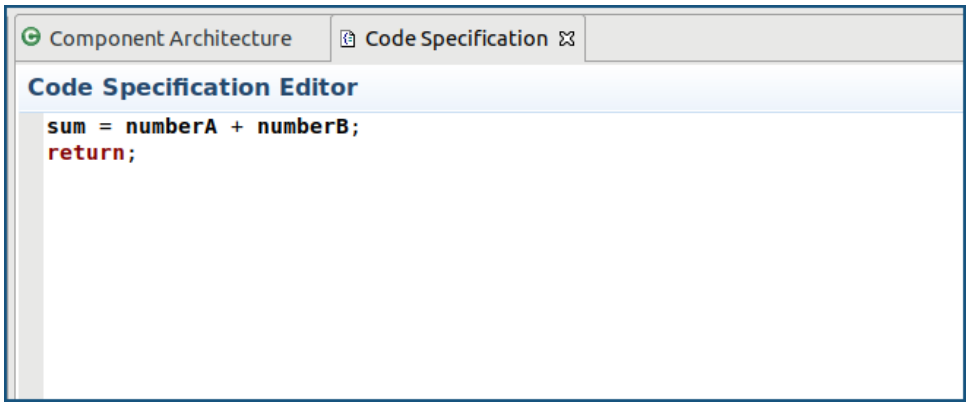


Figure 24: AF3 - Code specification view

Automaton specification allows one to specify the behaviour of components in a graphical state automaton diagram, where one can specify states and transitions between states. Figure 25 illustrates the automaton specification view from AutoFOCUS3. It contains two states, namely StateA

and StateB. The blue color of State1 denotes that the StateA is the initial state of the automaton. The black arrows denote the transitions between StateA and StateB. Guards may be specified to define which transition is used at the next step. Guards must be specified if a state has multiple outgoing transitions. For each state, one can define idle actions. Idle actions are fired in every simulation state if the state is active and no outgoing transition guard is fulfilled.

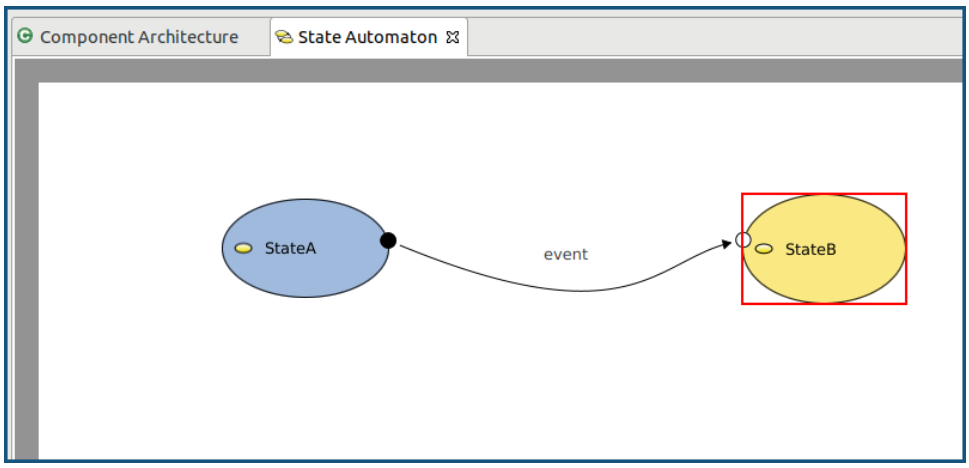


Figure 25: Automaton specification view

AutoFOCUS3 provides a way to validate the behaviour of components by means of simulation. Figure 26 illustrates the simulation perspective of AutoFOCUS3. One can simulate the entire component architecture or selected components only. Figure 26 illustrates the simulation of the behaviour of ComponentB. To run the simulations, one needs to manually provide the input values for components. For instance, in the middle of Figure 26 the variables numberA and numberB were manually assigned with values 2 and 4, respectively. One can run the simulation in a stepwise fashion by clicking on the simulation step button at the top of the simulation view. The result of the simulation is shown on the right-hand side of Figure 26. It shows that after one simulation step the variable sum is assigned with value 6, as expected.

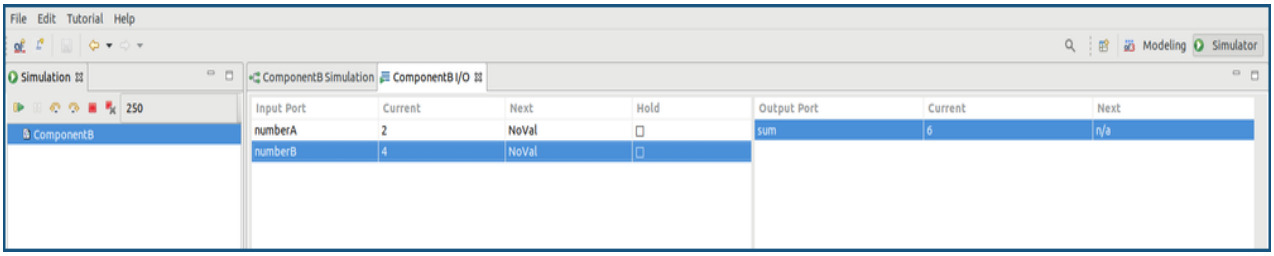


Figure 26: AF3 - Simulation perspective

AutoFOCUS3 supports the automatic generation of C code from the specified component architectures. This feature allows one to directly deploy the generated code into the target system (e.g., into a Raspberry Pi used by the FTS rovers). The tutorial on how to deploy the generated C code into systems can be found here [8] (access by request). Code-wise, AutoFOCUS3 will create a dedicated folder for all header files (with extension .h), and one dedicated folder for the actual implementation (with extension .c). For each component architecture, AutoFOCUS3 will generate a file with the C code for that component architecture. AutoFOCUS3 will create one dedicated file for the defined data dictionary that includes datatypes, arrays, and functions.

## 4.3 Technical characteristics Verification and Validation

We modelled and implemented methods for the verification and validation of the safety and security of selected connected car scenarios. Our methods can be classified into three types of verification and validation.

- **Formal Verification of Platooning Scenarios:** We constructed models based on the Soft-Agents [4] a formal framework for security verification and assessment. This model is reported in a submission under peer review also described in the following subsection. In a nutshell, our framework supports the specification of cyber aspects, such as communication protocols, and physical aspects, such as position, speed, acceleration, of cyber-physical systems, such as vehicle platooning. Moreover, our framework enables the specification of intruder model that can manipulate communication channels to, for example, cause harm. Our machinery has been implemented in the rewriting tool framework Maude [5]. By using our framework, we were able to discover three novel attack scenario, detailed Section 3.2.
- **Penetration Testing:** We have used current tools from the project related to this task to get all the possible information about the system to test. Once analysed the system architecture, a HW set of tools has been designed and set up. On top of this HW tools, a questionnaire has been filled up with all the possible protocols used in the Platooning Scenario of the Demo. By analysing these protocols, a list of possible vulnerabilities has been studied at the same time the HW has been constructed. More details about this HW setup and the vulnerabilities are explained in next sections.
- **Simulation-based fault injection:** We have used the Sabotage tool (see Section 3.2.12), which is based on the Fault injection or the deliberate introduction of faults into a system, that has been widely used in order to assess the dependability of a system under test. Specifically, the tool consists on the Simulation-based fault injection which involves the construction of a behavioural model of the system. The simulation models can be developed in different level of abstractions such as Simulink, SCADE or using very high description languages like Very High-speed integrated circuit Hardware Description Language (VHDL). In the context of CAPE, we focus on the Simulink behavioural model.

### 4.3.1 Formal Security Verification Framework

In D5.2, we detail the main model elements for the specification of vehicle platoon scenarios. We summarize below some of the modelling elements of our framework. Further details are available in D5.2. Then we focus on how these elements can be used for the verification of safety and security of vehicle platoons.

The key model elements are listed below:

- **Knowledge Base:** Vehicles have a local knowledge base (*lkb*). It represents the vehicle's view of the world, e.g., the speed and position of itself and of the other vehicles. Formally, a vehicle knowledge base is composed by a set of grounded facts, *p*, i.e., facts not containing variable symbols, of the form *p*, or associated with a timestamp, *p@t*, where *t* is natural number.
- **Sensors:** A vehicle is equipped with three sensors *locS*, *speedS* and *gapS*. They measure, respectively, the vehicle's location, speed and the gap to the vehicle immediately ahead. As we illustrate below, at each tick, vehicles use these sensors to query the environment knowledge base and update the vehicle's local knowledge base.
- **Communication Channels and Protocols**: We assume that vehicles may communicate using peer-to-peer connections or by broadcasting messages. Based on this assumption, we implement several protocols for platooning including: Heartbeats protocols from Leader to Follower, Follower to Leader, and Leader to Joining Vehicles; Emergency brake protocols.
- **Actions**: Vehicles decide to accelerate or decelerate. Since there may be infinitely many possibilities of acceptable speeds (for safety and fuel efficiency), we abstract actions by using facts of the form *act(id,vmin,vmax)* denoting a set of actions of changing id's speed to values

between *vmin* and *vmax*. Actions are evaluated with a value that is the result of a soft constraint problem specification described next.

- **Soft Constraints:** The evaluation of possible actions is done by taking into account the vehicle's concerns specified as a soft constraint problem. The soft-agents framework supports several types of soft-constraint theories, such as probabilistic, fuzzy and classical logic. We have implemented fuzzy controllers that take into account two concerns, fuel-efficiency and safety.
- **Intruder Model:** The intruder can impersonate an honest vehicle, listening to messages, injecting messages, and blocking messages from communication channels. These capabilities enable us to carry out similar verification done for safety, but now considering a malicious intruder. Currently, our intruder model supports two capabilities: Message Injection and Message Blocking.

We illustrate how our model can be used with an example.

The following local knowledge base of vehicle *v(1)* specifies that he is following vehicle *v(0)*. The vehicle *v(1)* has speed 20 and position 945 distance units. He believes to be immediately behind vehicle *v(0)* with a gap of 55 distance units. The vehicle *v(1)* has a maximum acceleration of 3 acceleration units. Moreover, he keeps track of the three last speed values, 25, of *v(0)*.

*LKB1 : (clock(3) (atloc(v(1),loc(945)) @ 3) (mode(v(1),following(v(0))) @ 3) (speed(v(1),20) @ 3) (gapNext(v(1),55) @ 3) (idNext(v(1),v(0)) @ 3) maxAcc(v(1),3) (histSpd(v(1),v(0),25 @ 3; 25 @ 2; 25 @ 1) @ 3) (histGap(v(1),55 @ 3; 55 @ 2; 55 @ 1) @ 3) fuel(v(1),1,3) safe(v(1),2,4))*

The following initial platooning configuration, *S0,* illustrates a scenario with two vehicles *v(0)* and *v(1)*, where *LKB1* is the local knowledge base above.

*{ [eId | kb ]*

*[v(0) : veh | lkb : LKB1, sensors : (locS speedS gapS), evs : (tick @ 0)]*

*[v(1) : veh | lkb : LKB, sensors : (locS speedS gapS), evs : (tick @ 0)] }*

Notice that the vehicles have sensors locS, speedS and gapS that sense, respectively, the location, speed and gap to obstacles in front. Moreover, [eId | kb] specifies the environment and its knowledge base *kb* contains the actual state of the platoon.

From a given initial configuration, we can verify the safety of platoons. For example, we can use our framework to check whether vehicles can crash even without an intruder. For that we specify the function *crash(S1)* returns true whenever the configuration *S1* consists in a state where the vehicles *v(0)* and *v(1)* crash.

Consider the initial configuration *S0'* obtained from the initial configuration above, but where the speed of *v(1)* is 40 instead of 20. We can run the command:

*search[1] S0' => S1 such that crash(S1) .*

And the Maude engine attempts to find whether a configuration *S1* that constitutes a crash scenario can be reached. Maude is able to find such configuration within 52ms. This result means that the chosen parameters for the soft-constraints do not work w.r.t. safety. Indeed, one could expect a crash when the speed of a vehicle is much greater than the speed of its preceding vehicle.

The configurations above did not include an intruder model and therefore are only useful to verify the safety of platoons. The following intruder configuration extends the configuration above with an intruder *intSpec* with the capability of injecting and blocking messages.

*Isys0 : { S0 ; intSpec }*

For example, consider an intruder that can inject at any time the following message:

*msg(v(0),v(1),hbl2f(v(0),70,loc(1070), none))*

where he impersonates vehicle *v(0)* informing that he is at much greater speed of 70 than *v(0)* actually is (of 25). We can use Maude to find whether such an intruder can cause a crash by running the following command:

*search isys0 =>* isys1 such that crash(isys1) .*

It takes around 90 seconds for Maude to find an attack.

One way to mitigate these attacks is by specifying plausibility checks, as described in D5.2. Our machinery also supports the specification of such plausibility checks. However, as described in Chapter 3, we discoverer through our formal framework 3 new attacks for which some cannot be mitigated by the proposed plausibility checks.

To this end, we formalized such attack scenarios using the intruder model supported by our Soft-Agents verification framework. We run the search command in Maude to automatically check whether two vehicles crash under the presence of the intruder. We run all experiments on a 1.90GHz Intel Core i7-8665U with 16GB of RAM running Ubuntu 18.04 LTS with kernel 5.4.0-47-generic and Maude 3.

| | Attack Scenario | Capability | Countermeasure | # States | Execution Time (min) | Attack Successful |
|---|---|---|---|---|---|---|
| II-B | Injection of False Msgs against Follower | INJ + BLK | N | 15351 | 0.034 | Y |
| | Injection of False Msgs against Follower | INJ + BLK | COMM | - | 120 | - |
| | Injection of False Msgs against Follower | INJ + BLK | SNSR | - | 120 | - |
| | Injection of False Msgs against Follower | INJ | N | - | 120 | - |
| II-C | Slow-Injection of False Msgs against Follower | INJ + BLK | N | 3315284 | 52.764 | Y |
| | Slow-Injection of False Msgs against Follower | INJ + BLK | COMM | 3286681 | 53.251 | Y |
| | Slow-Injection of False Msgs against Follower | INJ + BLK | SNSR | - | 120 | - |
| | Slow-Injection of False Msgs against Follower | INJ | N | - | 120 | - |
| II-D | Injection of False Msgs against Joining Vehicle | INJ + BLK | N | 9408 | 0.023 | Y |
| | Injection of False Msgs against Joining Vehicle | INJ + BLK | COMM | 9408 | 0.027 | Y |
| | Injection of False Msgs against Joining Vehicle | INJ + BLK | SNSR | 9407 | 0.023 | Y |
| | Injection of False Msgs against Joining Vehicle | INJ | N | - | 120 | - |
| II-E | Injection of False Emergency Brake Msgs | INJ + BLK | N | 593 | 0.002 | Y |
| | Injection of False Emergency Brake Msgs | INJ | N | 2218 | 0.011 | Y |
| II-F | Blocking Legitimate Emergency Brake Msgs | BLK | N | 6539 | 0.013 | Y |

Figure 27: Evaluation of the attack scenarios. Some experiments were aborted after 120 minutes

Figure 27 summarizes our main results. We considered 5 different types of attack scenarios, specified in detail in D5.2, where attacks are carried during different platooning conditions.

- **II-B - Injection of False Messages Against Follower**: In this attack, an intruder sends false position and speed values to a vehicle in order to cause a crash with the preceding vehicle. This attack works because CACC algorithms ensure that a vehicle maintains a desired distance from the preceding vehicle based on the received messages from other vehicles in the platoon (especially from the leader)
- **II-C – Slow-Injection of False Messages:** The goal of the previous attack II-B is a quick crash between two vehicles. To this end, the intruder injects extreme false position and speed values into the CACC communication channels.

- **II-D – Injection of False Messages Against Joining Vehicles:** A new vehicle may join a platoon after a negotiation phase (a.k.a synchronization handshake) with the leader of the platoon. During this negotiation phase, the leader sends the platoon information to this vehicle, including the position and speed of the last vehicle, so that the joining vehicle can adapt itself to catch up to the platoon. An intruder may impersonate the leader to send false information during this negotiation phase.
- **II-E – Injection of False Emergency Brake Messages:** The emergency brake is a safety-type message that may be triggered by any vehicle in the platoon to avoid crashes. An intruder, however, might take advantage of this situation to carry out attacks causing vehicles to apply emergency brakes without the need to do so.
- **II-F Blocking Legitimate Emergency Brake Messages:** Instead of injecting false emergency brake messages, the intruder may block legitimate emergency brake messages from the CACC communication channels in order to cause a crash.

The scenarios II-D, II-E and II-F are novel, being discovered while using our formal verification machinery.

We also considered two different plausibility countermeasures denoted as COMM (communication-based) and SNSR (sensor-based) in Figure 27.

- COMM: The communication-based countermeasure works as follows. Whenever a vehicle receives a message with the speed of the preceding vehicle, the countermeasure checks it against the history of speed values communicated that is stored. The countermeasure is triggered if the incoming speed value deviates from 30\% w.r.t. the average of the last *n* speed values received by the vehicle.
- SNSR: The sensor-based countermeasure estimates the speed of the preceding vehicle based on the information obtained from the gap sensor. That is, we estimate the speed of the preceding vehicle by computing *(spd + (gap2 - gap1))*, *spd* as the speed of the vehicle and *gap2* and *gap1* as the last two gap distance measurements.

Our intruder using both capabilities has successfully carried out the attacks II-B, II-C, and II-D against a platoon without countermeasure. The attack II-B, however, has not led to a crash when the countermeasures were deployed. In fact, this result was expected as attack II-B sends high-speed values to a target vehicle. We run the search command to look for a crash between two vehicles without the countermeasure being triggered. We could not find any crash (in 120 minutes). The attack II-C bypassed the communication-based countermeasure, but not the sensor-based countermeasure. Next, the attack II-D led to a crash even when the countermeasures were deployed.

Interestingly, neither of those three attacks led to a crash using the injection capability only (i.e., no blocking at the same time). This happens because the target vehicle receives legit and false messages during the attack, and dynamically adapts its acceleration based on the received messages. That is, the target vehicle accelerates when receiving a false message from the intruder, e.g., with high-speed values, and decelerate when receiving legit messages from the leader. Therefore, we speculate that anti-jamming countermeasures could serve as an additional layer of defense against injection attacks for CACC platoons.

Finally, both the attacks using emergency brake messages led to a crash. In particular, the attack II-E is effective even without blocking messages from the communication channels. This is due to the fact that vehicles immediately stop driving upon receiving an emergency brake message regardless of any message (usually blocked by the intruder) sent by the leader.

### 4.3.2 Penetration Testing

The collection of information for penetration testing to be done in the assessment phase has begun. Once the architectures and the outputs of the tools used in previous stages are ready, they are being analysed as inputs for the penetration testing phase.

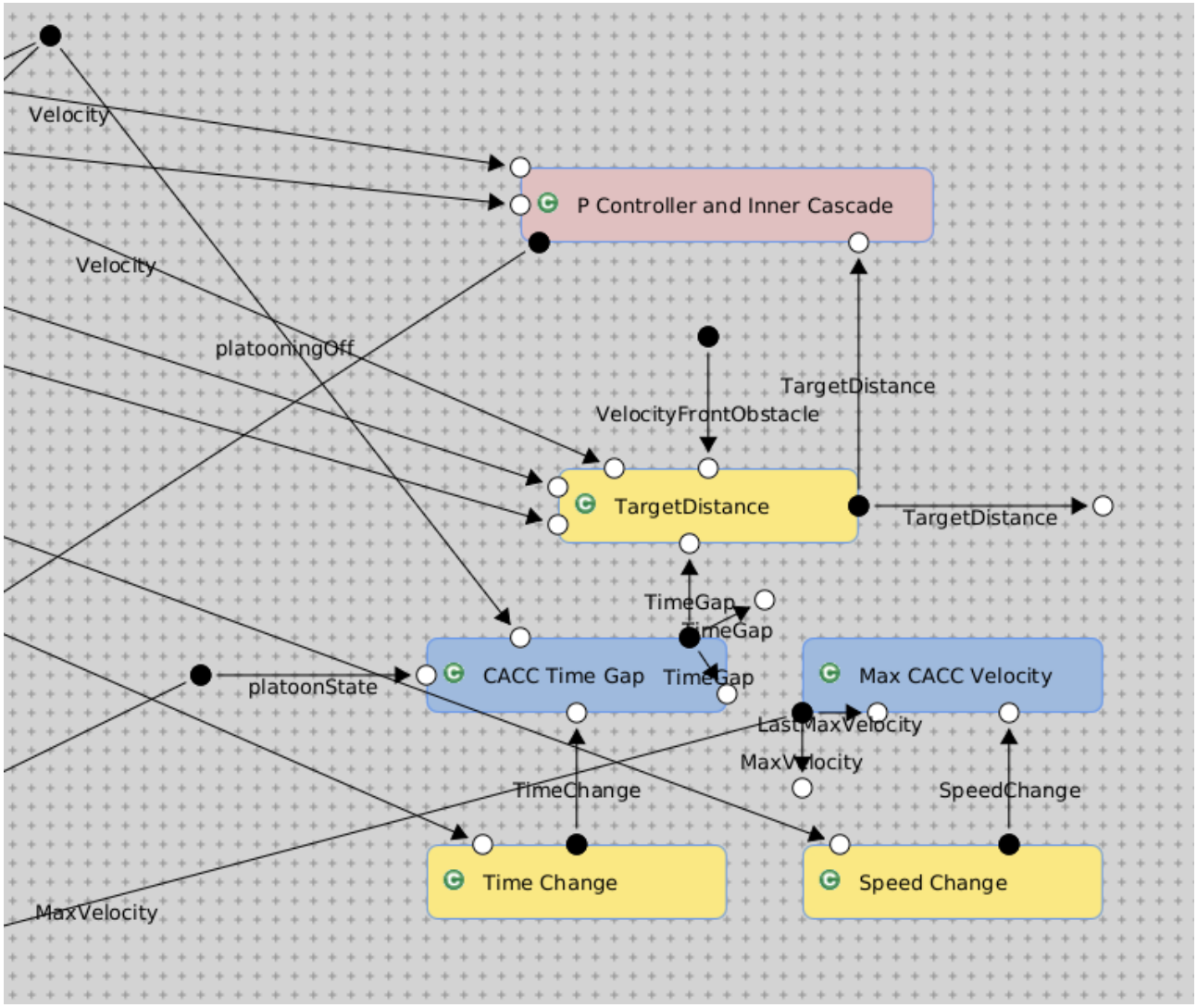


Figure 28: AF3 Tool provides a great overview to analyse possible attack vectors

A questionnaire with the protocols and sensors used by both Rovers' architectures has been collected which helps Eurecat to begin to search for already known vulnerabilities which could be exploitable. Here below there is a table of the main protocols used by each Rover implementation. As part of the AVA_VAN document a list of possible vulnerabilities associated to these protocols is going to be searched.

| | Tecnalia | Fortiss |
|---|---|---|
| Ultrasounds | | HC-SR04 40kHz |
| WiFi | 802.11n | 802.11n 5.0GHz |
| TLS | 1.3 | N/A |
| Python | 2.7 | N/A |

Table 23: Main protocols provided to Pen Tester as part of a grey-box strategy

On the HW side, the tools for penetration testing are being prepared. A Raspberry PI 4 with Raspberry OS has been set up and an ultrasound sensor with the same frequency range of the rovers has been chosen to be connected and controlled by the Raspberry. The possibilities of the Raspberry to be used for monitoring and packet injection have been considered. Nexmon framework is going to be tested (https://github.com/seemoo-lab/nexmon/blob/master/README.md) for this purpose

For penetration testing, it is also considered to use a Kali Linux distribution in the Raspberry. In any case, the use of common exploration tools like NMAP is going to be used. As one of the main vectors is the WiFi protocol, some possible scripts have been considered as for instance the ones described in https://github.com/vanhoefm/krackattacks-scripts.

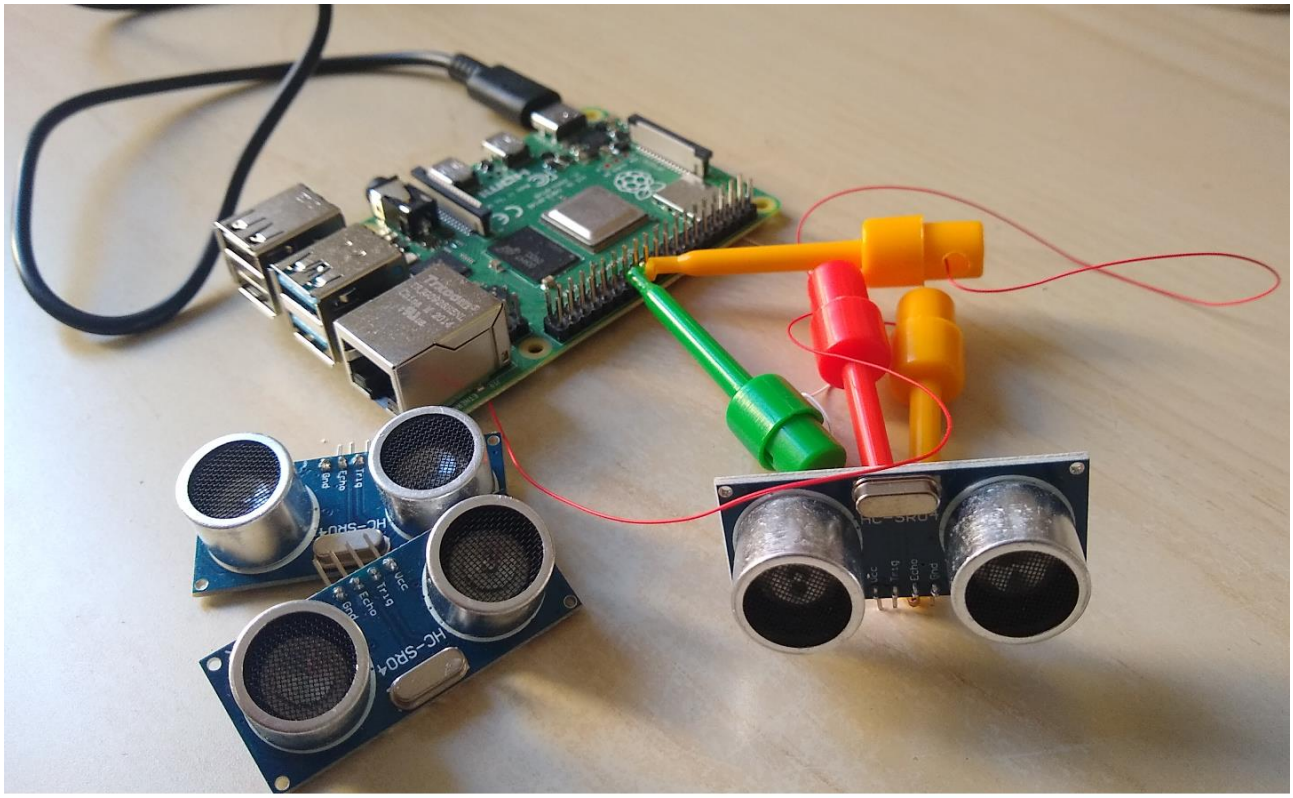

Figure 29: HW tools: Raspberry PI and ultrasounds transducer

### 4.3.3 Simulation-based fault-injection

Fault Injection contributes to the safety analysis phase, which includes the verification and validation of safety concepts and requirements. Some of its most remarkable aims are to support the assessment of implemented safety requirements, and the correct implementation and the effectiveness (diagnostic coverage) of safety or fault-tolerant mechanisms. However, sometimes traditional safety analysis methods such as Fault Tree Analysis (FTA) or Failure Mode and Effect Analysis (FMEA) are not sufficient. Manual reviews are normally needed to proof the completeness and the correctness of those analyses. Furthermore, the failure logic or the effects of certain faults cannot easily be determined by those analysis techniques. A promising approach to overcome this limitation is to combine traditional analysis with fault injection approaches. It is important to understand that fault injection mitigates the new challenges but cannot replace safety assessments such as FTA or FMEA. Therefore, fault injection and traditional safety analysis techniques complement each other.

In order to identify differences in the system´s behaviour and to automate the fault injection campaigns, the simulation results of a faulty system under test (faulty SUT) or extended system model with faulty behaviour are compared versus a fault-free system (golden SUT) under the same workload. Extra model blocks (saboteurs) are injected into the component inputs, which reproduce a certain failure mode. After that, the effect of that fault can be observed in the output by including extra read-out blocks or monitors. These fault injectors simulate failures at input ports and the inclusion of monitors in the outputs tool in order to detect whether and in which ways an output assertion is violated in consequence. The results can be stored as part of the safety case as applies to conventional safety analysis techniques.

Section 7.6 describes deeply how this technique has been used in the context of the Connected Car case study.

## 4.4 Update

This section describes the implementation status at M24 of the two platooning V2I scenarios described in D5.2: the firewall reconfiguration scenario and the firewall update scenario.

In the V2I **firewall reconfiguration** scenario platoons communicate with traffic control centers via edge clouds distributed along the road network (see Figure 30: Firewall update – Security orchestration with vulnerability scan on edge change. As the platoon progresses it must change edge clouds to get the best network latency available. This requires reconfiguring the certified vehicle firewalls. From the certification point of view only authorized firewall reconfigurations should be made as specified by the certified requirements. The firewall must be monitored for detecting firewall intrusions and unauthorized changes to the configuration that would allow an attacker to launch injection or jamming attacks. This monitoring mechanism is currently under implementation.

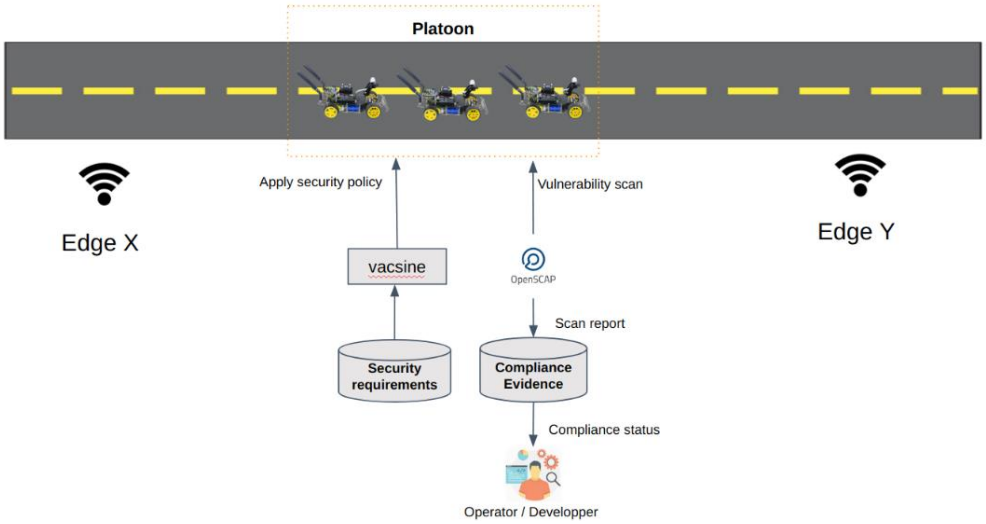


Figure 30: Firewall update – Security orchestration with vulnerability scan on edge change

The above figure shows the VACSINE tool transforming security requirements into the edge security policies and deploying them on the platoons that are passing by. To ensure that firewalls are not compromised and remain compliant, vulnerability scans are launched after the reconfiguration. The current tool used for monitoring and compliance is OpenScap. Compliance evidence include firewall reconfiguration and operation logs, as well as the vulnerability scan reports.

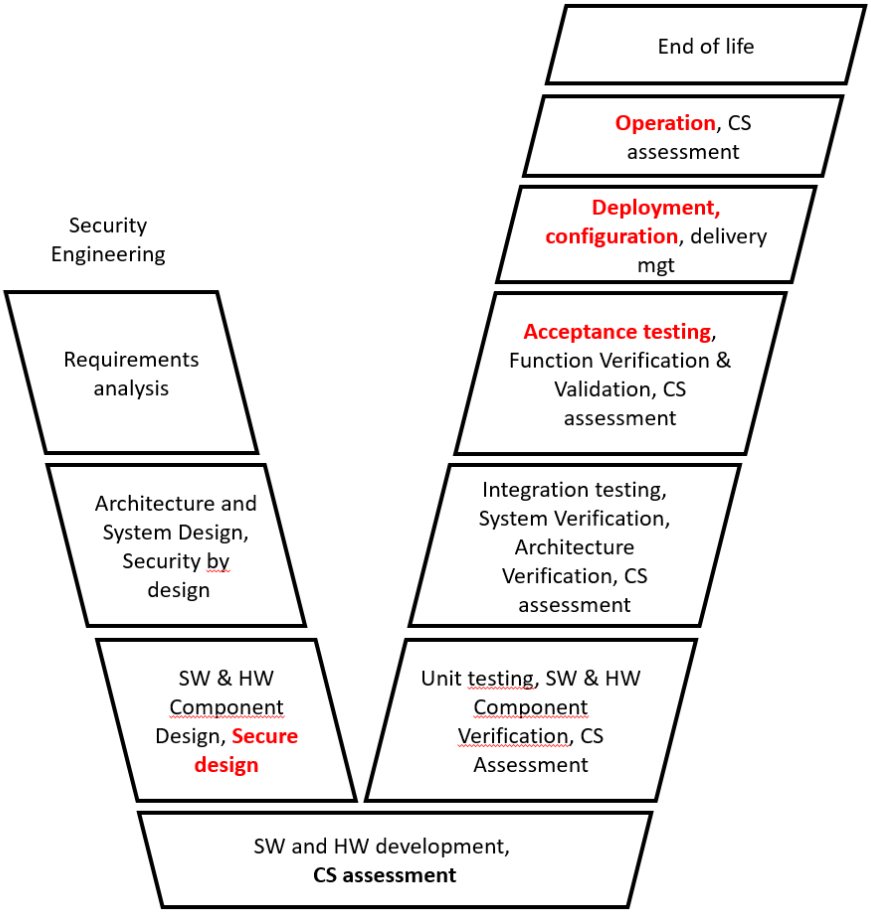


Figure 31: Firewall update - V-Model related to software/security engineering process

In the V2I **firewall update** scenario, a new version of the firewall is available and needs to be deployed on customer vehicles. The update is performed when vehicles are not being driven. The firewall update is orchestrated by the VaCSIne tool that is running in a Cloud and has agents deployed in the vehicles. From the certification point of view, if some certified requirements are impacted then the new firewall version must be re-certified on vehicles. This requires following the certification process for the impacted parts.

Figure 31 highlights the V-Model security steps associated with the firewall update activities: secure design using SCAP for security policies, testing by, for example, running vulnerability scans, orchestrating the deployment and configuration of security services and monitoring the results of the reconfiguration and scans in the operations step. Section 7.3 provides more details on the Firewall update implementation in an Infrastructure to Vehicle (I2V) case study where we show how we maintain continuous compliance when security requirements are changing

## 4.5 Assessment

The assessment process activities can be linked with the various task of the V-Model related to software/security engineering process, as shown in the following figure.

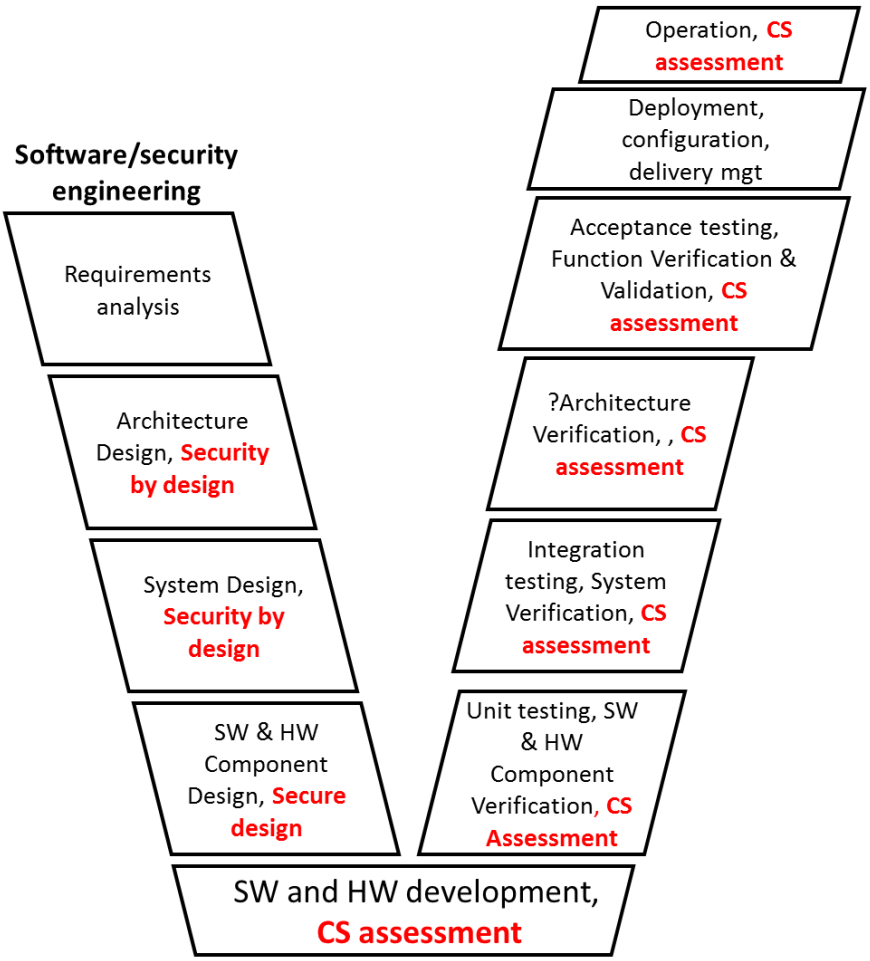


Figure 32: V-Model related to software/security engineering process

After Requirement Analysis, the Architecture Design, and System Design must foresee **Security by Design**, as well as Component SW & HW Design in order to have a **Secure Design.**

After the development of SW & HW components a **Cybersecurity Assessment** (VA/PT) is needed, and this must repeated after the activity of SW & HW Component Test, Integration testing and Function Verification and Validation.

A visual mapping of the assessment process with the activities of the engineering process is shown in the following figure:

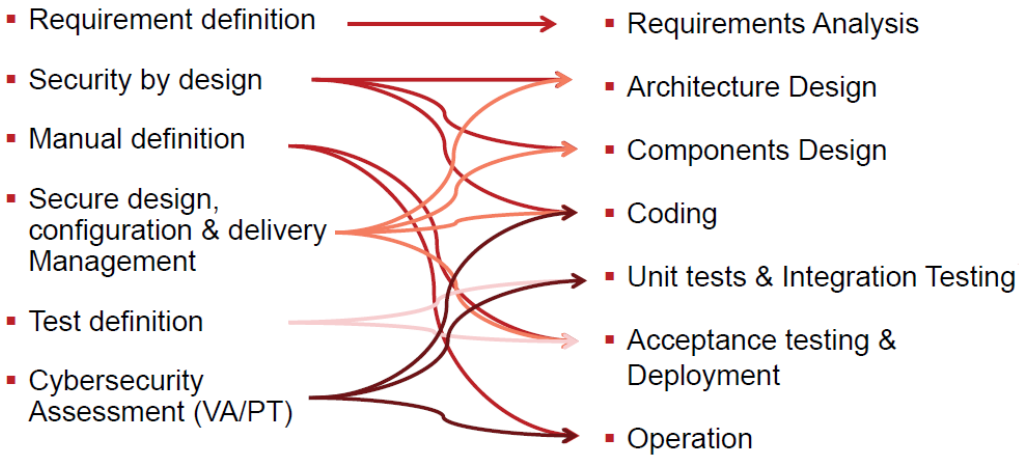


Figure 33: Assessment process (V-Model) mapping

# Chapter 5 Prototypes for Risk discovery, assessment and management for complex systems of systems (T5.3)

## 5.1 Introduction and overview

Table 24 provides an updated overview of tools and contributions related to the security of software and software supply chains. Notable changes compared to deliverable D5.1 are as follows:

- Buildwatch from UBO will be demonstrated in the context of the e-Government vertical instead of the connected cars vertical. Initially it was planned to demonstrate Buildwatch by analysing the autonomous vehicle's platform Apollo Auto. This software, however, is not used within the connected cars vertical of WP5. Hence, a more suitable demonstration based on Shibboleth, leveraged by CINI's SAML IdP, is preferred. This barely influences Buildwatch's development roadmap and planned evaluations as it was planned to be language agnostic from the beginning.
- Project KB from SAP has been added to the list, which encompasses a YAML data format, a dataset with security-related statements about open source projects as well as a tool to facilitate data production and consumption. This work was formerly mentioned as part of Eclipse Steady, cf. Section 3.1.9 in D5.1, but has been promoted to a separate project/tool, due to the general need for publicly available dataset with code-level security information. As of Nov 20, the dataset comprises statements related to 702 different security vulnerabilities.
- The Package Scanner from SAP has been removed from this list, which was meant to detect malicious packages deployed in the PyPI package repository. Still, a prototype has been developed in collaboration with Cybersecurity for Europe, and its results led to several publications.

SafeCommit was originally meant to identify both vulnerability-introducing and vulnerability-fixing commits. However, to avoid overlap with SAP's work on Commit2Vec (cf. Section 5.2.2.2 in D5.2), which also identifies fix-commits, it was decided to focus SafeCommit on vulnerability-introducing only.

| Partner | Contribution | Section | Technologies Covered | Use-case |
|---|---|---|---|---|
| UBO | Buildwatch | 3.2.3 | Agnostic | E-government |
| CEA | Frama-C | 3.2.4 | C | Connected cars |
| CINI | Approver | 3.2.1 | Java (Android) | E-government |
| SAP | Steady | 3.2.15 | Java, Python | E-government |
| SAP | Project KB | 3.2.10 | Agnostic | E-government |
| UNILU | Logic Bomb Detection | 3.2.6 | Java (Android) | E-government |
| UNILU | SafeCommit | 3.2.13 | C/C++ | Connected cars |
| UKON | Supply chain visualization | 3.2.18 | Java, Python | E-government |

Table 24: Overview about tools extended/developed in the context of task 5.3

Beyond actual tools, task 5.3 produces several additional contributions in the form of datasets, data formats and models/techniques/concepts:

- KEMEA contributes a model to determine the attractiveness of open source projects from the viewpoint of attackers (cf. Section 5.2.3.1 in D5.2).
- UBO and SAP contribute a dataset with malicious packages that have been used in past, real-world attacks against the npm, Ruby and Python ecosystems (cf. Section 5.2.3.2 in D5.2). As of December 7th, the dataset comprises information about 1083 malicious packages.

The synergies and integration of all the above contributions is described in the following section, while in-depth descriptions of each tool can be found in Chapter 3 of this deliverable and Section 3 of deliverable D5.2.

## 5.2 Synergies and integration of individual contributions

This section describes synergies and integrations of the various SPARTA tools and contributions related to the security of software and software supply chains. Figure 34 provides a comprehensive overview of those contributions, their nature, e.g., tool or dataset, and selected dataflows.

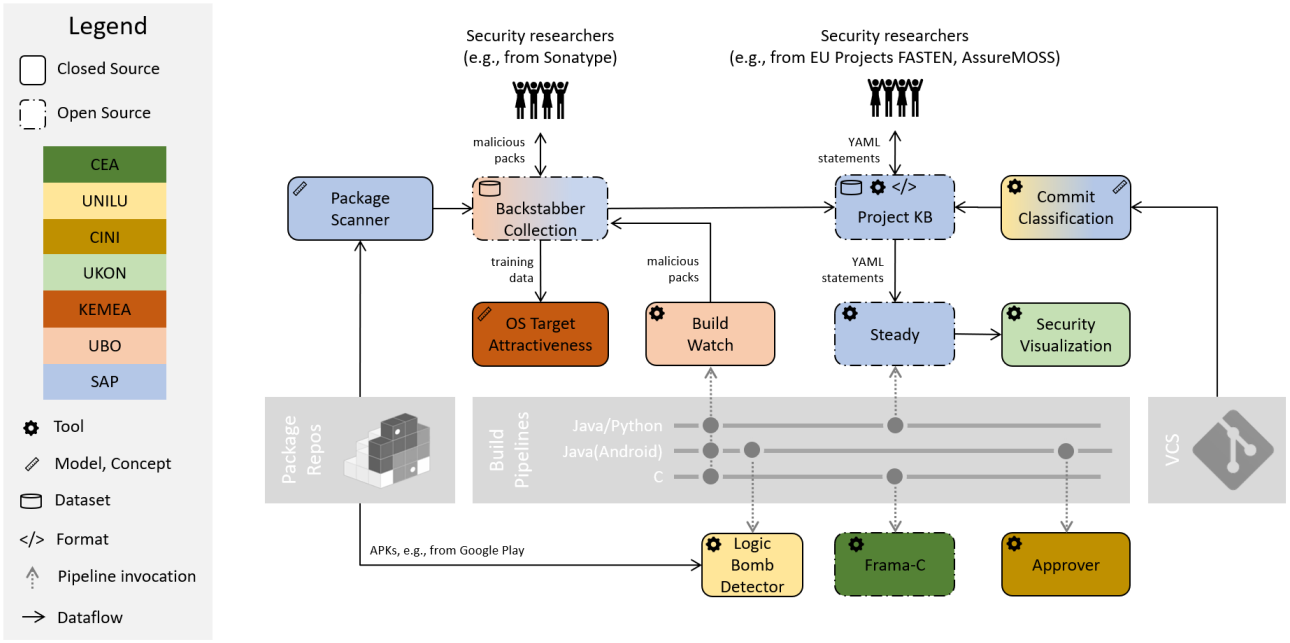


Figure 34: Synergies and integration of task 5.3 contributions

The primary targets of all contributions are (1) package repositories, e.g., PyPI or npm, which contain pre-built, binary packages of open source projects, (2) build pipelines, which automate the compilation, packaging and deployment process of given software projects and (3) versioning control systems, which support the versioning of source code and other development artifacts of software projects. Some tools can be applied in multiple contexts, e.g., the Logic Bomb Detector, which can be used to check both arbitrary Android packages published on app stores such as Google Play, as well as Android packages built for a given project during its pipeline runs.

The general guideline towards tool integration was to focus on Linux-based build pipelines as integration technology, and – where possible – to avoid any direct communication channels.

This design choice provides a great degree of freedom to tool end-users, who can assemble their pipelines according to risk profile and appetite. Of course, tool interaction is also needless when different technologies are addressed, e.g., Frama-C targets C/C++ while Approver targets mobile

applications for the Android operating system. Furthermore, the focus on pipelines also provides freedom to tool developers, who can develop using arbitrary technologies and programming languages as long as their tooling can be invoked through the command line of Linux-based pipeline executions.

Accordingly, Figure 34 illustrates the invocation of different tools during three different build pipelines. BuildWatch is agnostic to the technology of the development project in question, hence, can be invoked no matter the programming language used. In contrast, Frama-C, Approver, Steady and Logic Bomb Detector target specific technologies.

However, a direct communication channel was established in the case of the Security Visualization tool and Eclipse Steady. Its purpose being to provide organizations with holistic views on open source consumption and security, it seemed appropriate to connect to Steady, which – due to its centralized architecture – holds current and historical scan results of potentially all development projects of an organization.

Two dataset contributions of task T5.3, the Backstabber's Knife Collection of malicious open source packages used in real-world attacks, and Project KB with code-level information about vulnerabilities in open source projects, play an important role in terms of conceptual integration and community building.

The motivation for both datasets is to provide high-quality data to developers and researchers within and outside of SPARTA. Consequently, both projects have been open-sourced on GitHub, and already enjoyed contributions from 3rd parties, including commercial software providers and EU research projects. They both fill a gap, as corresponding datasets were not available in the past. In case of Project KB, for instance, the code-level information about fix commits and vulnerable open source projects is more precise and actionable than corresponding databases like the CVE/NVD.

The consumption and population of those datasets within SPARTA is illustrated by dataflows in Figure 34. Dataset contributions typically involve a certain degree of manual review in order to maintain the dataset quality, while the consumption happens mostly automated.

One example flow goes from the Backstabber's Knife Collection to Project KB, and from there to Eclipse Steady. In this example, information about malicious packages is exported to Project KB using its YAML format, and from their automatically imported into Steady's internal database. As a result, users of Steady will be notified if their software project depends on a malicious open source component version.

The Backstabber Collection is also used to train models aiming to determine the attractiveness of open source projects for attackers, and – in general – serve to improve the detection techniques and tools such as Build Watch and Package Scanner. Any new findings of malicious packages will be reviewed and fed into the dataset.

Similarly, the results of our efforts to automatically classify commits in versioning controls systems as security-relevant, either vulnerability introducing or fixing, will be reflected as YAML statements in Project KB.

# Chapter 6 Prototypes for Integration on Demonstration Cases and Validation (T5.4)

## 6.1 Introduction

The concept of evaluation certification in the context of “Cybersecurity” has evolved in the last decades to face what have been the related problems that have arisen with the passage of time and with technological developments.

Evaluation and certification development has proved more difficult in the world of information security, where they have had a reasonably long history in which it is possible to highlight the advantages and disadvantages of its development.

After the end of the Cold War, the European model prevailed creating the Common Criteria for Information Technology Security Evaluation.

The common criteria introduced the concept of evaluation against a model called "protection profile" which specifies the type of threats that must be assumed and the type of protection that must be provided against such threats

Thanks to its flexibility this approach has found fertile ground also in this project. Demonstration is set in Task 5.1 of WP5 in which an ad hoc PP for vertical 1 was defined and in which we have gone even further by inserting in such a format alongside the security requirements those of safety.

## 6.2 Evaluation Process Concepts

Security should be viewed as a process, which should not be static. Moreover, it must be easily modifiable so that any improvement can be implemented and it must cover the entire life cycle of the target (product, system or process) to which security is applied.

One of the problems that arise with a static type certifications is that the validity of the certifications would seem to decay when the first patchless vulnerability is highlighted.

This leads on one hand to push towards the patching of vulnerabilities, but on the other hand it is necessary that such patching, if relative to a certified target, needs the definition of an appropriate procedure that goes to consider the correct actors (those of the certification process) and accompanies this target throughout its life cycle.

But this is only one aspect of one of the phases of what we can introduce with the name of "Security Process".

Let's then consider what the approach to safety, adopted in the CAPE program of this project, was and in particular in deliverable D5.1

In this context, security (in general) has been approached as an iterated process, and it has been shown that the identified process is applicable to different frameworks of interest.

In particular, the same process has been adapted to the following contexts:

- Common Criteria Evaluation Process
- Safety engineering
- Security engineering

using the V-model as shown in the following image.

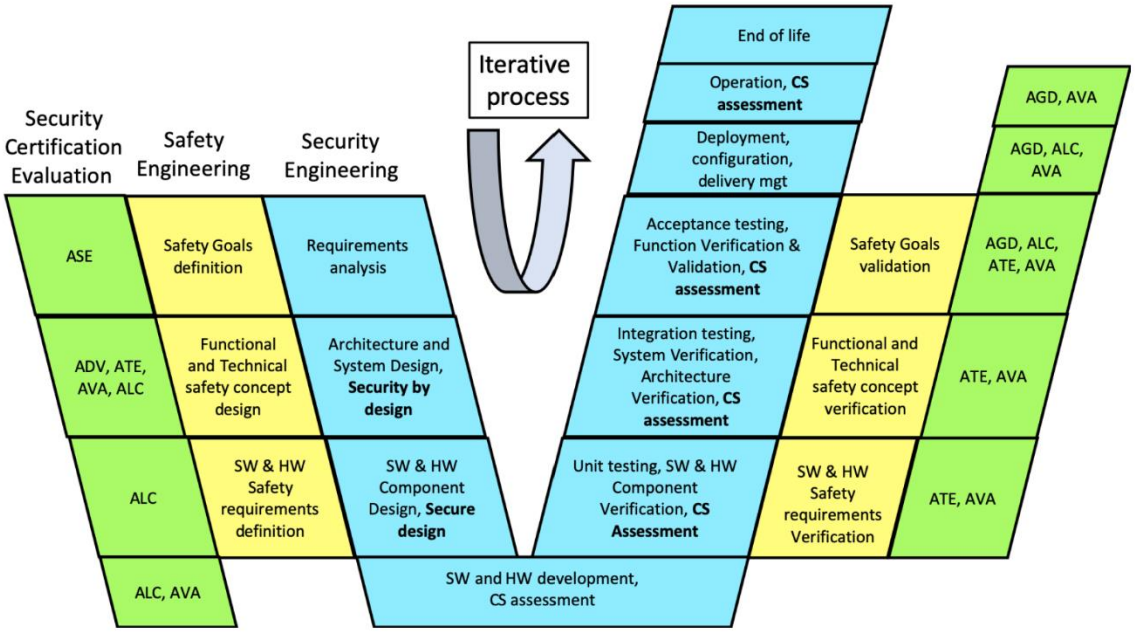


Figure 35: Different frameworks V-model appliance

All of these, however, can be traced back to a flow process relating to the security assessment that can generally be applicable to a product, a system (of any complexity) or a process, during its entire life cycle

## 6.3 Introduction to verticals evaluability

The correlation between the assurance classes and the Cybersecurity process phases can be described as follows:

- ASE (Security Target Evaluation): this class deals with the evaluation of the consistency of the ”Security Target” which also contains the definition of the security requirements of the TOE, therefore it is closely linked to the security requirements management phase.
- ADV (Development): this class deals with the evaluation of the six families of requirement for structuring and representing the security functionality realized by the target of evaluation (TOE) at various levels and varying forms of abstraction that the developer must produce during the product development phase, naturally it is linked to the features of the Secure by design processes adopted by the supplier.
- AGD (Guidance Documentation): this class takes care of the evaluation of the manuals that are delivered to the customer. These manuals contain both the secure configuration process of the TOE in its user environment and its safe use methods for each category of defined end-user.
- ALC (Life-cycle support): this is a very important class that evaluates all aspects of the management of the TOE during its life cycle: in the development phase in which it is under the responsibility of the developer, during the transitional phase of transport in its final operating environment and of course the management in the operating environment under the responsibility of the customer and the developer, in the hypothesis of maintaining the certification (security patch management).

- ATE (Tests): it is the class that takes into consideration all the tests that demonstrate that security functionalities operate according to its design descriptions, both the functional ones proposed by the developer and the independent ones proposed by the evaluators.
- AVA (Vulnerability Assessment): this class takes care of vulnerability assessment activity to analyze vulnerabilities in the development and operation of the TOE. Development vulnerabilities are those introduced during its development and these can be minimized with the adoption by the developer of ”security by design” processes. Operational vulnerabilities are those that could exploit the weaknesses of non-technical countermeasures to violate the TOE security functionality. This analysis is carried out by the evaluators during TOE evaluation deliverables analysis or from the classic vulnerability analysis performed also adopting automatic tools.

Of course, in such a process, compared to a normal CC certification process, to maintain the status of the target obtained with the certification, some of these phases must be repeated throughout its life cycle.

### 6.3.1 Vertical 1 - Connected and Cooperative Car Cybersecurity (CCCC) in the context of Euro NCAP

In the context of the Platooning scenario (Vertical 1 Chapter 7), the safety and security requirements of the scenario have been defined in the same Common Criteria Protection Profile.

The Target Of Evaluation (TOE) is the Safety and Security Platooning Management Module (SafSecPMM), that is used to ensure the safe and secure operation of vehicle platoons. The TOE has an interface towards the Vehicle Communication Device (VCS), the Hardware Security Modules (HSM), if HSM is available and is not directly integrated in VCS, and the Vehicle Control Module (VCM).

The PP of SafSecPMM is included in Deliverable D5.2 Annex B. It incorporates security countermeasures and other security features to increase the robustness of the platooning behaviour, provides accountability information for this behaviour, and contains security measures to protect its own assets.

### 6.3.2 Vertical 2 - Complex System Assessment Including Large Software and Open Source Environments, Targeting e-Government Services

A different approach was followed for Vertical 2 (Chapter 8) evaluation instead of CC Protection Profile.

The security evaluation of the CIE ID app in the context of Vertical 2 involves the verification of a set of security requirements that enable the assessment of the security posture of the mobile app.

The following security requirements has been identified, using OWASP Mobile Application Security Verification Standard:

| ID | OWASP MOBILE TOP 10 | Description |
|---|---|---|
| SecR1 | M3 | Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app. |
| SecR2 | M7 | The app is signed and provisioned with a valid certificate, of which the private key is properly protected. |
| SecR3 | M7 | The app has been built in release mode, with settings appropriate for a release build (e.g., non-debuggable). |

| ID | OWASP MOBILE TOP 10 | Description |
|---|---|---|
| SecR4 | M7 | Debugging code and developer assistance code (e.g., test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages. |
| SecR5 | M7 | Debugging symbols have been removed from native binaries. |
| SecR6 | M7 | The app only requests the minimum set of permissions necessary. |
| SecR7 | M2<br>M5<br>M7 | The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices. |
| SecR8 | M2 | The app removes sensitive data from views when moved to the background. |
| SecR9 | M7 | The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app. |
| SecR10 | M3 | The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards. |
| SecR11 | M7 | JavaScript is disabled in WebViews unless explicitly required. |
| SecR12 | M3<br>M7 | The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted. |
| SecR13 | M3<br>M7 | The app only depends on up-to-date connectivity and security libraries. |
| SecR14 | M2<br>M7 | No sensitive data is shared with third parties unless it is a necessary part of the architecture. |
| SecR15 | M2<br>M7 | No sensitive data should be stored outside of the app container or system credential storage facilities. |
| SecR16 | M7 | The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected. |
| SecR17 | M7 | No sensitive data is included in backups generated by the mobile operating system. |
| SecR18 | M7 | A WebView's cache, storage, and loaded resources (JavaScript, etc.) should be cleared before the WebView is destroyed. |
| SecR19 | M3 | The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA. |
| SecR20 | M7<br>M9 | Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis. |
| SecR21 | M3 | A WebView's cache, storage, and loaded resources (JavaScript, etc.) should be cleared before the WebView is destroyed. |

Table 25: Security Requirements for Vertical 2 in T5.4

The aforementioned security requirements will be evaluated, using the tools **Approver** and **TSOpen** by implementing the DevSecOps pipeline.

Eclipse **Steady** will be used to assess the presence of known security vulnerabilities affecting any of the dependencies of the specific version of Shibboleth integrated in the scenario. In addition, Eclipse Steady will be used to detect whether the code implemented to customize the solution depends on open-source components with known vulnerabilities, to collect evidence regarding the execution of vulnerable code, and to provide updated recommendations.

# Chapter 7 Vertical 1 – Prototypes for Connected and Cooperative Car Cybersecurity (CCCC)

## 7.1 Introduction

This chapter describes how the prototypes described in previous chapters have been applied to the cooperative car cybersecurity vertical (a.k.a. Connected Car Vertical), in particular to the scenarios described in D5.2. For more details on such scenarios, we refer the interested reader to D5.2.

For each scenario, we detail in the following sections the activities carried out till the time of the deliverable. More specifically, Section 7.2 describes how prototypes are used in the basic scenario. The next sections increment the basic scenario with new features or new methodologies to evaluate the basic scenario. Section 7.3 describes how the firewall can improve the security of the basic scenario. Section 7.4 describes the evaluation process for validating the basic scenario with penetration testing. Section 7.5 assesses both safety and security compliance and certification with the OpenCert tool. Finally, Section 7.6 describes fault-injection and analysis of faulty scenarios by means of simulations.

## 7.2 Scenario 1: Basic Scenario

The basic scenario's goal is to evaluate the process, from security analysis, requirements to implementation and verification and validation, for increasing the security of vehicle platooning when assuming a malicious intruder that can manipulate the communication channels.

The goal of our intruder is to cause a crash between two legitimate vehicles. To this end, the intruder injects false messages into the CACC communication channels. To ensure that injected messages are valid, we assume that the intruder can obtain encryption keys from any vehicle in the platoon.

Finally, the scenario is complemented by the incorporation of a dashboard, a web page that will allow checking the status of the platoon. In addition, this dashboard is used to inject false speed messages from one car to another.

### 7.2.1 Modelling and Implementation

The implementation of the Connected Car basic scenario has started from the safety and security requirements described in detail in the Safety and Security Platooning Management Module (SafSecPMM) Protection Profile (see D5.2 Appendix B [2]). This set of requirements support the implementation of the two iterations of the Basic Scenario, having the first one being achieved during the second year of the SPARTA project.

Table 26 shows the requirements that have been defined for the first iteration of the Basic Scenario, both for the TEC demonstrator and the FTS demonstrator. The open requirements in the table (yellow or red cells) will be tackled in the next iteration of the Basic Scenario.

| Req. Id | Short Description | TEC | FTS |
|---|---|---|---|
| **PMM_IF.1.1** | Maintain heart-beat data (vehicle identifier, speed, direction, geo-position, timestamp) to VCS | **Solved** | **Solved** |
| **PMM_IF.2.1** | Maintain heart-beat data from VCS | **Solved** | **Solved** |
| **PMM_IF.3.1** | Maintain incoming emergency brake | **Solved** | **Solved** |
| **PMM_IF.4.1** | Maintain incoming emergency brake | **Solved** | **Solved** |

| Req. Id | Short Description | TEC | FTS |
|---|---|---|---|
| **PMM_IF.5.1** | Maintain data from VCM | **Solved** | **Solved** |
| **PMM_IF.6.1** | Maintain data to VCM | **Solved** | **Solved** |
| **PMM_PC.1.1** | Data passes all VCS plausibility checks | **Solved** | **Solved** |
| **PMM_PC.3.1** | Inform on Failed Plausibility Checks | **Solved** | |
| **PMM_VCS-HPC.1.1** | Maintain heart-beat data history | **Solved** | **Solved** |
| **PMM_VCS-HPC.2.1** | Heart-beat message consistent to the history | **Solved** | **Solved** |
| **PMM_VCS-SPC.1.1** | Maintain distances history | **Solved** | |
| **PMM_VCS-TPC.1.1** | Consult the TOE vehicle internal clock | **Solved** | |
| **PMM_VCM-HPC.1.1** | Maintain sensor data history | **Solved** | **Solved** |
| **PMM_VCM-TPC.1.1** | Consult the TOE vehicle internal clock | **Solved** | |
| **PMM_VCM-TPC.2.1** | Message freshness | **Solved** | |
| **FPT_ITC.1.1** | Inter-TSF confidentiality during transmission | **Solved** | |
| **FPT_ITI.1.2** | Inter-TSF verify integrity | **Solved** | |

Table 26: Requirements covered by each demonstrator for the first iteration of the Basic Scenario

It is important to emphasize that during the second year of the SPARTA project FTS and TEC have consolidated a close collaboration between their laboratories. This collaboration has been mainly reflected in four ways: periodical meetings, common set of requirements, a shared MS Excel sheet to overview implementation progress, and a common Git repository for code.

- **Fortnightly meetings** have been held between FTS and TEC to inform about the implementation status, share ideas, have a better understanding of our Rovers, help each other to solve problems, discover good candidate parts to be re-used, etc. In some of them, other CAPE partners, Leonardo and Eurecat, have joined to support the elaboration of a Protection Profile according to the ISO/IEC 15408 Common Criteria standard.
- The implementation of the two demonstrators has started from a **common set of requirements**, defined jointly in the Protection Profile document. As a result, FTS and TEC laboratories have produced similar results in the implementation, but with some specificities that are described in Section 7.2.1.1 and 7.2.1.2 respectively.
- FTS and TEC have also worked collaboratively by using an **implementation status sheet** that keeps track of the basic scenario development (see Figure 36). This file, that is available at the SPARTA project SVN, has allowed TEC and FTS to track the implementation progress of the requirements. For each requirement, we indicate its identification, short description, detailed description, iteration, whether the requirement will be covered in the TEC and/or Fortis demonstrator, current status (pending, solved, partially solved, cancelled) and comments to it.
- Finally, a **private git repository** has been created by TEC to share some code that would be used in both demonstrators and therefore, to save implementation effort. More specifically, the CACC implemented by FTS has been integrated into the TEC Rovers (see Figure 41). In the future, FTS also plans to integrate the TEC Dashboard (see Figure 42) into the FTS demonstrator, thus becoming another code-reuse sample. As an illustration of the Git Repository we have included the Figure 37, showing the high usage of the repo with 240 commits so far, and the Figure 38 showing the AutoFOCUS3 Project for the CACC and generated code.

| ID | Short Description | Description | TEC Rovers Demonstrator | FTS Rovers Demonstrator | TEC Rovers Status | FTS Rovers Status |
|---|---|---|---|---|---|---|
| PMM_IF.1.1 | Maintain heart-beat data to VCS | The TOE shall maintain an outgoing heart-beat data flow with other platooning vehicles as specified below:<br>• From TOE to VCS (and then to another vehicle TOE)<br>• Messages transmitted shall contain the following data computed from the TOE vehicle sensors/algorithms:<br>◦Vehicle unique identifier<br>◦Vehicle speed<br>◦Direction<br>◦Geo-Position<br>◦Timestamp | X | X | Solved | Solved |
| PMM_IF.2.1 | Maintain heart-beat data from VCS | The TOE shall maintain an incomming heart-beat data flow with other platooning vehicles as specified below:<br>• From (another vehicle TOE to vehicle) VCS to TOE<br>• Messages transmitted shall contain the following data collected by the VCS from the links to other vehicles<br>◦Unique identifier of the vehicle to which the data corresponds to<br>◦Vehicle speed<br>◦Direction<br>◦Geo-Position<br>◦Timestamp<br>◦Digitally signed certificates | X | X | Solved | Solved |
| PMM_IF.3.1 | Maintain incoming emergency brake | The TOE shall maintain an incomming flow with other vehicles informing the TOE vehicle about emergency brake maneouvers as specified below:<br>• From (another vehicle TOE to vehicle) VCS to TOE<br>• Messages transmitted shall contain the following data:<br>◦Unique identifier of the vehicle to which the emergeny brake has been issued<br>◦Emergency brake identifier<br>◦Timestamp<br>◦Digitally signed certificates | X | X | Solved | Solved |
| PMM_IF.4.1 | Maintain incomming emergency brake | The TOE shall maintain an outgoing information flow with other vehicles informing the other vehicles about emergency brake maneouver performed by the TOE vehicle as specified below:<br>• From TOE to VCS (and then to another vehicle TOE)<br>• Messages transmitted shall contain the following data:<br>◦Unique idenfier of the TOE vehicle<br>◦Emergency brake identifier<br>◦Timestamp | X | X | Solved | Solved |
| PMM_IF.5.1 | Maintain data from VCM | The TOE shall maitain an incomming information from the TOE vehicle VCM and the TOE as specified below:<br>• From VCM to TOE<br>• Messages transmitted shall contain the following data:<br>◦Speed<br>◦Direction<br>◦Geo-Position<br>◦Gap to the next vehicle<br>◦Distance to the edges of the lane | X | X | Solved | Solved |
| PMM_IF.6.1 | Maintain data to VCM | The TOE shall maitain an outgoing information from the TOE to the the TOE vehicle VCM as specified below:<br>• From TOE to VCM<br>• Messages transmitted shall contain the following data:<br>◦Speed<br>◦Direction | X | X | Solved | Solved |
| PMM_PC.1.1 | Data passes all VCS plausilibity checks | The TOE shall accept data incomming from the VCS only if the data passes all plausilibity checks defined. | X | X | Solved | Solved |
| PMM_PC.2.1 | Data passes all VCM plausilibity checks | The TOE shall accept data incomming from the VCM only if the data passes all the plausilibity checks defined. | | | | |
| PMM_PC.3.1 | Inform on Failed Plausibility Checks | The TOE shall inform the other vehicles in the platoon, the TOE vehicle driver and the authorities whenever a sequence of M rounds of incomming data fails at least N plausibility checks. [M,N to be defined by the application] | X | X | Solved | |

Figure 36: MS Excel sheet showing the implementation status of a subset of the requirements from the PP

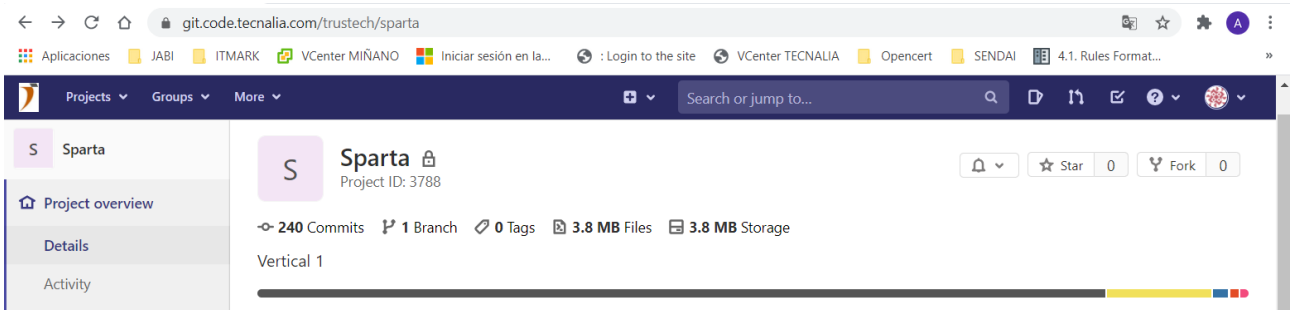


Figure 37: Git Repo for collaboration in the Platooning Basic Scenario

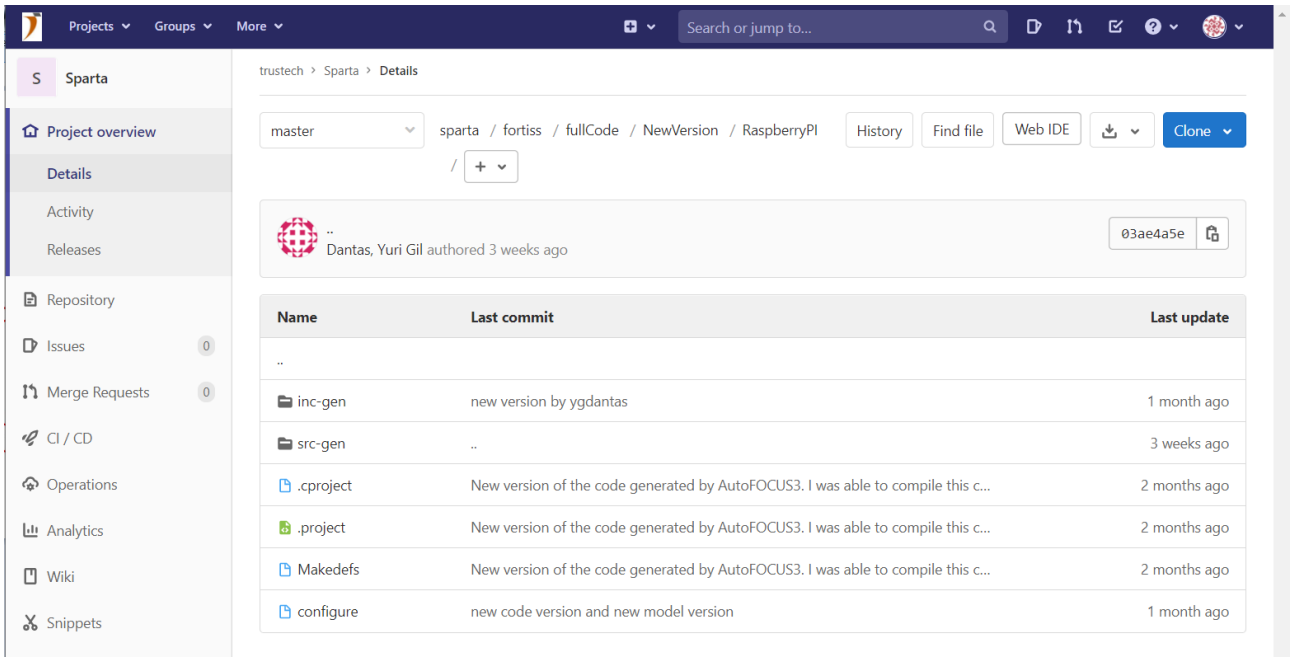


Figure 38: FTS CACC code in Git

### 7.2.1.1 Specificities for FTS Rovers (FTS)

We developed our platoon control system on AutoFOCUS3. In this section, we describe the communication process between platoon members, the Cooperative Cruise Control (CACC) system, and the plausibility check to mitigate injection attacks. The developed platoon control system has been deployed on the FTS rovers.

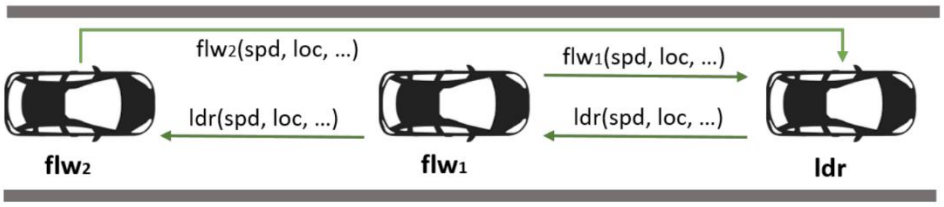


Figure 39: Communication process between platoon members

Figure 39 illustrates a platoon scenario that makes explicit the communication process between platoon members. This platoon scenario is composed of three vehicles: a leader (ldr), and two followers ($flw_1$ and $flw_2$, respectively). The green arrows illustrate the messages exchanged between vehicles. These messages are exchanged periodically. Each follower sends a message with information such as speed and location to the platoon leader. The leader gathers the received information and broadcasts it to each follower in the platoon. Each follower adapts its state based on the received message (as explained next). Technically, these messages are sent via UDP sockets. The developed communication process allows each follower to know relevant information about other vehicles. This information may be used to implement plausibility checks to mitigate injection attacks.

We developed a CACC system to enable followers to adapt to their state, such as speed and position, based on information exchanged between vehicles. The developed CACC system is both fuel-efficient and safe. That is, if the distance to the preceding vehicle is too great, the fuel consumption concern kicks in and attempts to reduce it by accelerating. Similarly, if the distance is dangerously short, then the safety concern kicks in and attempts to increase it by decelerating. To this end, our CACC system implements functions for fuel-efficient and safe.

- safe(id,min,max) denotes that the distance to the preceding vehicle of id is considered safe if it is between the values min and max;
- fuel(id,min,max) denotes that the distance to the preceding vehicle of id is considered fuel efficient if it is between the values min and max;

We attempt to satisfy both concerns, safety and fuel-saving, by searching for interceptions between safe and fuel values. If this is not possible, then safety is given priority over fuel-saving.

We developed a plausibility countermeasure to mitigate injection attacks that send false speed values. This plausibility countermeasure considers the speed of the preceding vehicle. To this end, we implemented a list on each follower to store the last n speed values from the preceding vehicle (a.k.a. history). The plausibility check works as follows. Whenever a follower receives a message with the speed of the preceding vehicle, the countermeasure checks it against the local history. The countermeasure is triggered if the incoming speed value deviates from 30% w.r.t. the average of the last n speed values received by the vehicle.

Through AutoFOCUS3, we automatically generated C code from the specified platoon control system, including the communication process, CACC and plausibility check. The generated code has been deployed on the FTS rovers. Figure E illustrates two FTS rovers: a leader (yellow rover) and a follower (blue rover) running the generated code from AutoFOCUS3.

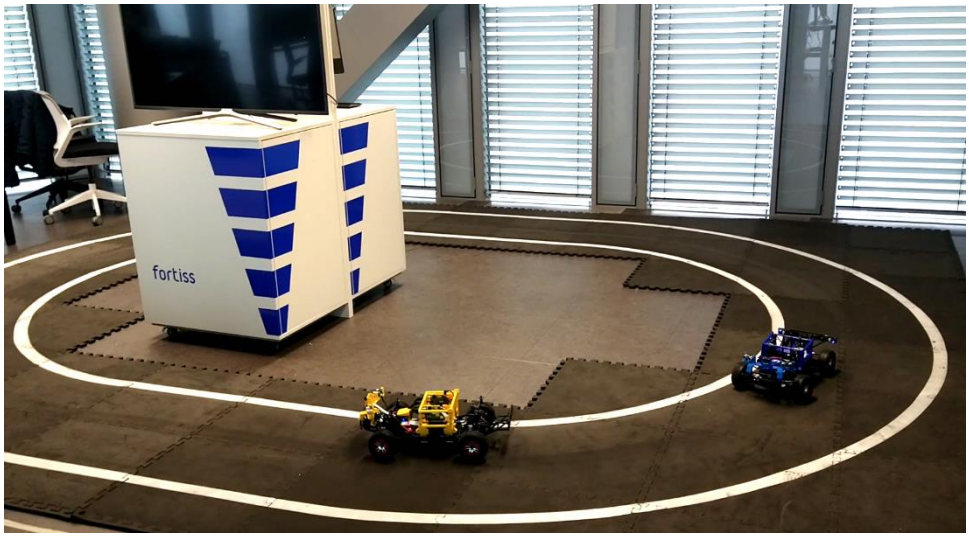


Figure 40: FTS rovers

#### 7.2.1.2 Specificities for TEC Rovers

Figure 41 shows the different software parts installed in TEC Rovers (the three rovers have exactly the same software with different configurations), the specific parts responsible for interchanging messages between cars, and the parties responsible for serving the dashboard page to a web client. For connectivity between cars and between cars and the computer displaying the dashboard, we use the same private WiFi network.

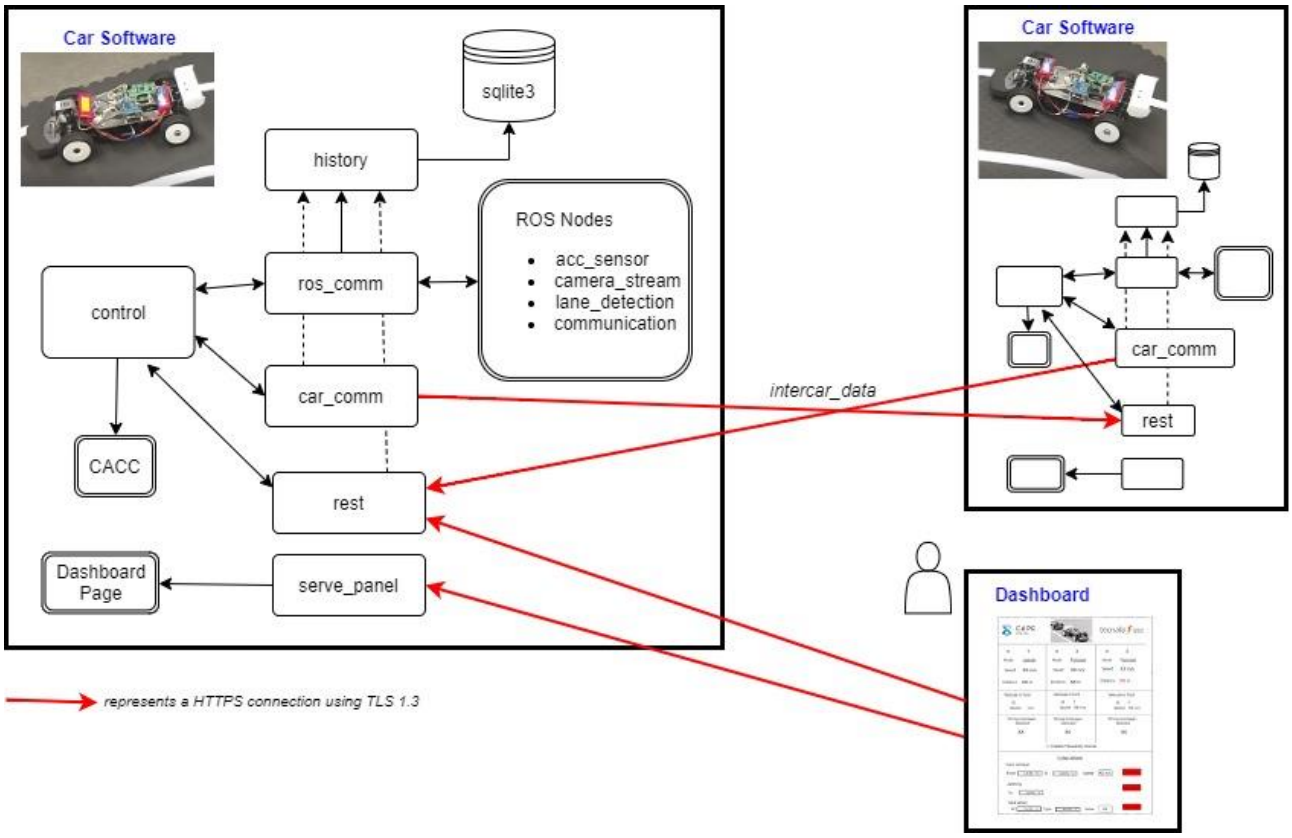


Figure 41: TEC Rovers Software architecture

The TEC Rovers autonomous driving is composed of different programs, written in C++, modelled as **ROS Nodes** that communicate using the ROS bus message system to interchange commands between the nodes. Each node takes on different tasks:

- **acc_sensor**: ultrasonic sensor node that provides the distance with the object in front of the car, usually a preceding car.
- **camera_stream**: get the images and pass them to the lane detection program.
- **lane_detection**: program in charge of analysing the camera images and calculating the trajectory to keep the car in the lane delimited by white lanes in the ground.
- **communication**: program that controls the steering and motion motors using a custom board. It accepts commands for setting the speed and steering angle and gives back the current speed.

The software implemented for the project (see Figure 41) fulfils the requirements specified in the protection profile. It is mainly a program composed of multiple threads that allow communication between the cars. All the software is written in python 2.7, except a CACC module provided by AutoFOCUS3 that is written in C but that has been compiled as a shared library (.so) and called from the python program. In addition, an NTP server has been installed in the leader to synchronize the clocks of the three cars.

The software uses Flask[7] as the framework to serve the pages and respond to the REST calls, the python module Requests[8] to create the REST HTTPS calls as a client and the rospy library[9] to communicate with the **ROS Nodes** in C++.

[7] https://flask.palletsprojects.com/en/1.1.x/

[8] https://requests.readthedocs.io/en/master/

[9] http://wiki.ros.org/rospy

For each thread, there is a corresponding class that manages all the communication with the related entity. The following is a list of the most important classes:

- **ros_comm**: is in charge of receiving all the ROS messages provided by other ROS nodes to acquire the different variables that are needed to know the car status: current speed, distance with the previous car and the steering angle. It also provides a method to send ROS messages for setting the leader speed.
- **car_comm:** is in charge of sending the **intercar_data** every 0.5 seconds to the other two cars in the platoon. The class establishes a secure TLS connection with each of the remaining cars and sends the same message to both via REST calls. The intercar_data interchanged is the following:
    - timestamp: message time
    - source: car sender id
    - destination: car receiver id
    - speed: sender car speed
    - distance: distance to previous car
    - direction: steering angle
    - emergency_brake: boolean indicating if the car is in emergency mode
    - emergency_break_id: emergency break id
- **rest**: is a RESTful API in charge of receiving the REST calls coming from the Dashboard and the calls that exchange the **intercar_data** coming from the other cars of the platoon**.**
- **control**: it centralizes the communication storing all the values coming from **ROS Nodes** and **intercar_data.** Besides**,** it integrates the CACC algorithm made by Fortiss that allows the platooning and has been explained before (see Section 7.2.1.1). It is also in charge of implementing the **countermeasures** to validate incoming messages to avoid and detect false messages coming from a malicious car. The countermeasures are triggered if the incoming speed value deviates from 10% the average of the last 3 speed values received by the vehicle. Finally, it also listens to the Dashboard events to disable the countermeasures and to inject false speed messages between cars.
- **history**: it implements the functions in charge of storing all the platooning data (all incoming and outgoing messages and the sensors data) in a local database using **sqlite3.** Each car's local database will have the same data.
- **serve_panel**: provides the Dashboard (see Figure 42).

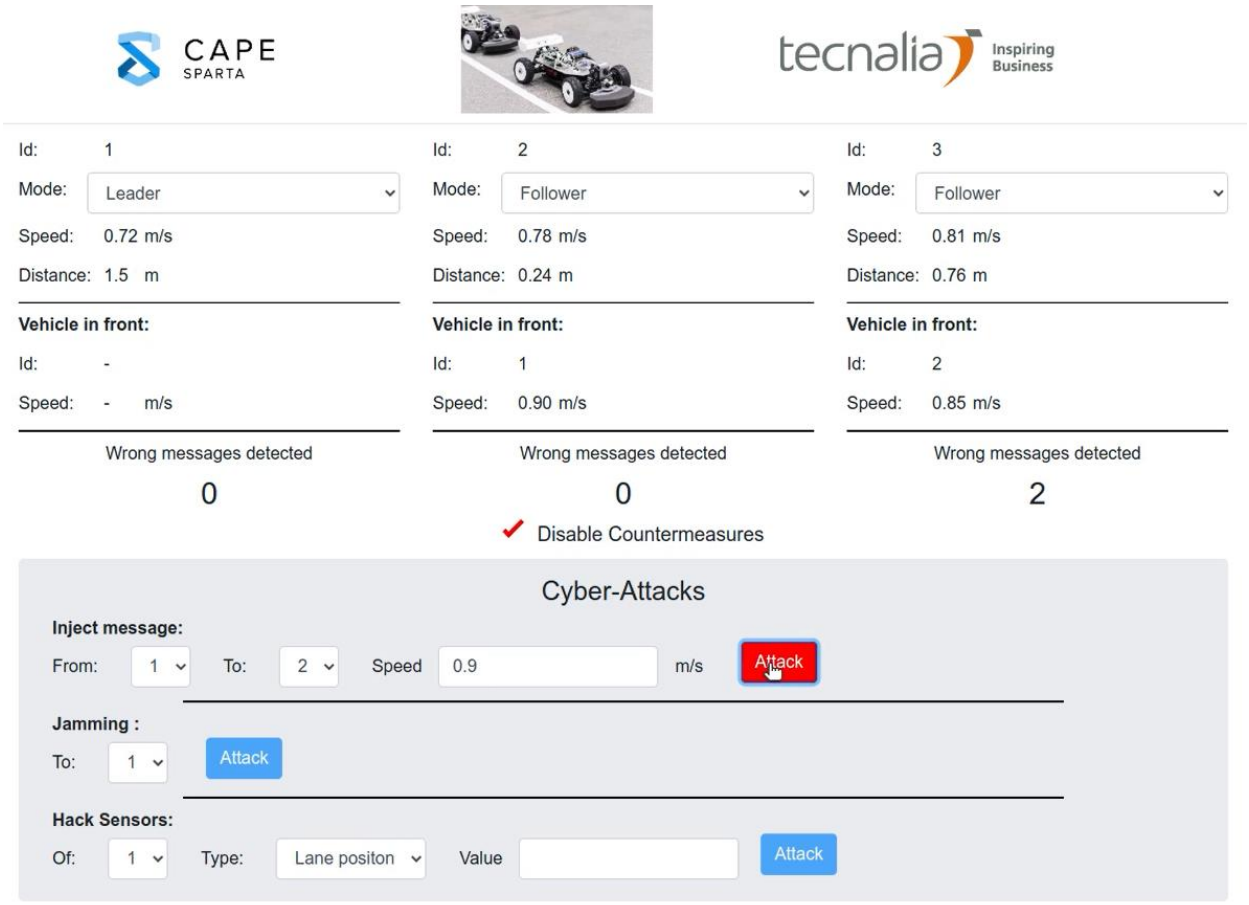


Figure 42: Dashboard

Finally, the Dashboard web application has been developed using Bootstrap framework[10] and jQuery[11], that uses the REST API to interact with the cars to support the following objectives:

- Show relevant indicators (**car_info**):
  - Identification
  - mode
  - speed
  - distance to preceding car
  - id and speed received from preceding car
- Show the number of wrong messages detected by the countermeasures.
- Disable countermeasures, to allow for crashes between attacked cars.
- Simulate a cyberattack by injecting a false speed in the **intercar_data** (note that the jamming and hacking sensor attacks haven't been implemented in the first iteration of the demonstrator)

[10] https://getbootstrap.com/

[11] https://jquery.com

### 7.2.2 Verification and Validation

Both verification and validation of the basic scenario will be handled by D5.4 [3]. This section briefly describes the first steps towards validating our model and implementations.

We have validated our AutoFOCUS3 model by means of simulations using AutoFOCUS3. More specifically, we validated the correctness of our CACC implementation, communication channels and plausibility countermeasures through a set of input values.

As described in Section 4.3.1, we also validated the security of CACC platoon using formal verification. We have formalized a CACC platoon in Maude. The formalized CACC platoon should cover the same functionalities of the CACC platoon modelled on AutoFOCUS3. We have validated the CACC platoon through multiple platoon scenarios such as joining and emergency mode. Security-wise, we validate the effectiveness of multiple attack scenarios and two plausibility countermeasures (all described in Section 4.3.1).

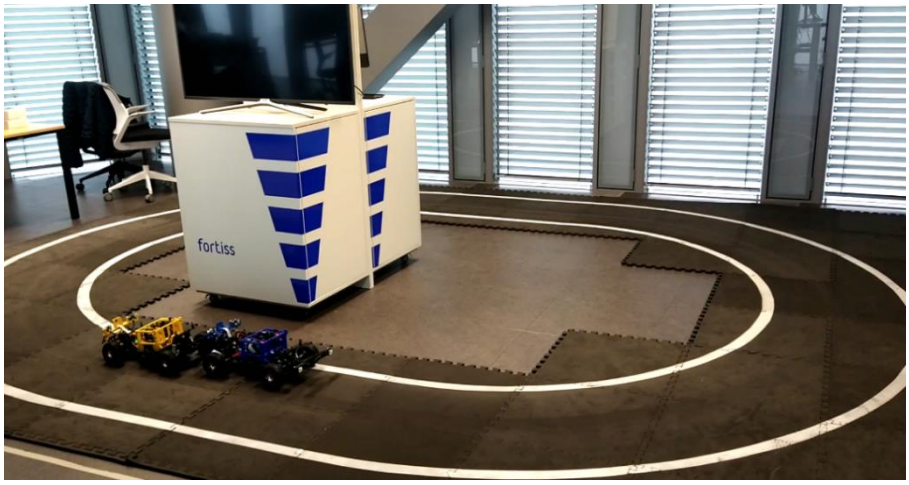


Figure 43: Injection of false speed values leads to a crash between vehicles

Next, we generated C code from AutoFOCUS and deployed the generated code on the FTS rovers. We validated the effectiveness of an injection attack (see attack II-B from Section 4.3.1) against the FTS rovers through actual experiments. That is, we validated whether the injection of false speed values would lead to a crash between two vehicles. Our experimental results confirmed the effectiveness of this attack. Figure 43 illustrates a crash between two FTS rovers, where the yellow rover maliciously sent false speed values to the blue rover.

In addition, we validated the effectiveness of a plausibility countermeasure (see COMM countermeasure from Section 4.3.1) against this injection attack. Our experimental results showed that the implemented countermeasure was effective in mitigating the injection attack.

The validation of the basic scenario has been recorded in a video, an outstanding project output focused on dissemination purposes. The video storyline goes through the definition of the Connected Car Platooning scenario, the requirements definition by means of the Protection Profile, the AutoFOCUS modelling process, the formal methods used for security-by-design, and concludes with the two demonstrators of the basic scenario, FTS demonstrator and TEC demonstrator. Figure 43 and Figure 44Figure 50 show some video frames extracted from the video.

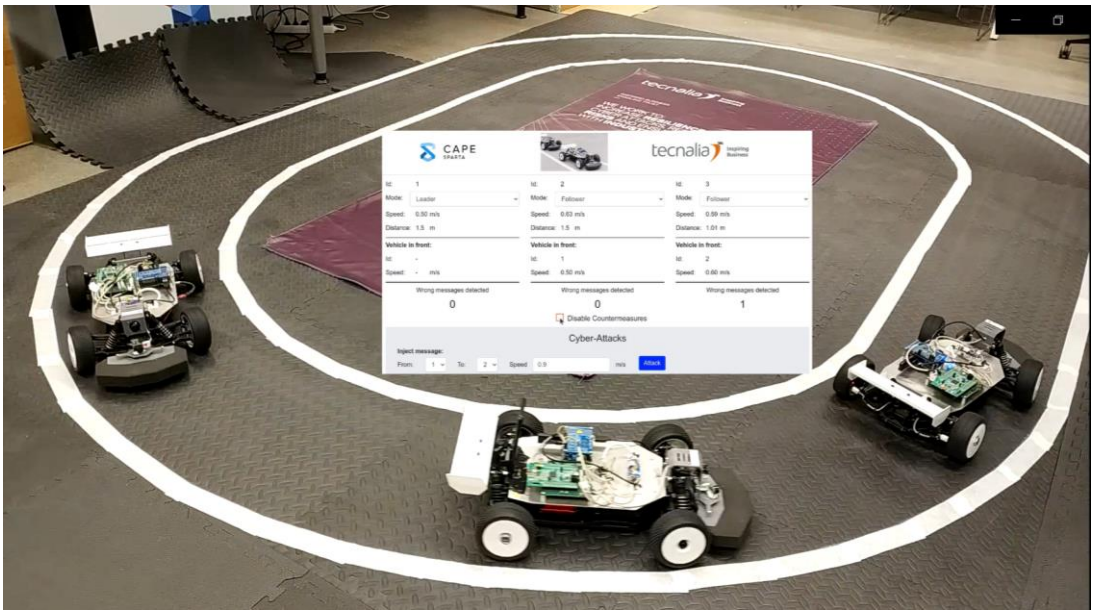


Figure 44: TEC Rovers moving in a platoon

### 7.2.3 Assessment

We assess the adequacy of our implementation based on the requirements defined in the protection profile. The complete list of requirements can be found in the SPARTA repository. A screenshot of a subset of these requirements is shown in Figure 36.

We have implemented multiple requirements from the protection profile, including:

- PMM_IF.1.1/PMM_IF.2.1: Platoon members (e.g., leader) may send heartbeat message to other platoon members (e.g., followers) as well as receive heartbeat messages from other platoon members.
- PMM_IF.3.1/PMM_IF.4.1: Platoon members may send emergency brake messages to other platoon members as well as receive emergency brake messages from other platoon members.
- PMM_VCS-HPC.1.1: Each platoon member keeps a history of the last n received heartbeat messages.
- PMM_VCS-HPC.2.1: Each incoming heartbeat message must be checked against the local history. As a result, only messages that are consistent with local history shall be allowed (plausibility countermeasure).

These requirements were enough to evaluate the basic scenario. They enabled us to evaluate the CACC platoon itself, and communication between platoon members. Moreover, we also evaluated the history-based plausibility countermeasure against an injection attack.

We are yet to implement other requirements such as the one about a plausibility check (PMM_VCS-HPC.3.1) for mitigating attacks injecting false emergency brake messages. We will also implement the requirement about keeping the history of the last n distance values from the preceding vehicle. This will help us to implement plausibility countermeasures that use data from the gap sensor, as described in Section 4.3.1.

We were not able to implement some requirements from the protection profile. For instance, the FTS rovers do not share internal clock. Hence, we could not implement the requirements PMM_VCM-TPC.1.1 and PMM_VCM-TPC.2.1 that aim to check the timestamp of incoming heartbeat messages for message freshness.

## 7.3 Scenario 2: Firewall updates

In this demonstration scenario, we develop an Infrastructure to Vehicle (I2V) case study where continuous compliance can be maintained when security requirements are dynamic. In the security branch of the V-Model, this scenario covers security design through the use of SCAP to describe the security policies, testing with the application of vulnerability scans and operations with the orchestration of the security services.

In a first iteration, we consider the case of a platoon that traverses zones with different security requirements. In a second iteration to be implemented in the first half of 2021 we will enrich the scenario to demonstrate how to apply security requirements to a vehicle joining the platoon and improve the security services registry with for example honeypots.

### 7.3.1 Modelling and Implementation

The demonstration scenario is developed on the CETIC testbed (see Figure 45), it is implemented as follows:

- A Private Cloud infrastructure managed by the Proxmox Virtual Environment[12] open-source server management platform coupled with Kubernetes[13] to provide container orchestration, Kubernetes clusters are managed using the Rancher[14] solution. The cloud is connected to the Edge nodes devices using the KubeEdge[15] system, which provides container orchestration at the Edge, Raspberry Pi devices are used to simulate the traffic infrastructure, providing wifi connectivity and local security zone policies for the platoon. This infrastructure offers scalable on-demand compute resources to support variations in the load of the simulation.
- The platoon consists of Donkey Car[16] rovers. They can use a Raspberry Pi to drive autonomously, but we chose the Jetson Nano Development Board[17] instead to leverage its improved GPU capabilities for data intensive processing at the edge (e.g. real-time image processing). The rovers use a wide-lens camera for lane detection, ultrasonic sensors and a 2D Lidar for distance detection.
- The orchestration of the security services is provided by VaCSIne, it is written as Python microservices. The Security Agent orchestrates the deployment and configuration of the security services (firewalls, vulnerability scanners, honeypots, etc.) using Ansible[18] playbooks and Helm[19] charts.
- Security services orchestration and execution produce traces that can be monitored. VaCSIne produces logs of the orchestration, for instance when a new service is deployed or reconfigured. Security services will also output traces, for example a vulnerability scan with OpenSCAP will produce execution logs and a vulnerability scan report. We use Grafana Loki[20] for the management and visualisation of the logs and security events in the system.

[12] https://proxmox.com/en/proxmox-ve

[13] https://kubernetes.io/

[14] https://rancher.com/

[15] https://kubeedge.io/

[16] https://www.donkeycar.com/

[17] https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[18] https://www.ansible.com/

[19] https://helm.sh/

[20] https://grafana.com/oss/loki/

- The infrastructure continuous integration, deployment and assessment is supported by the GitLab and Foreman platforms.

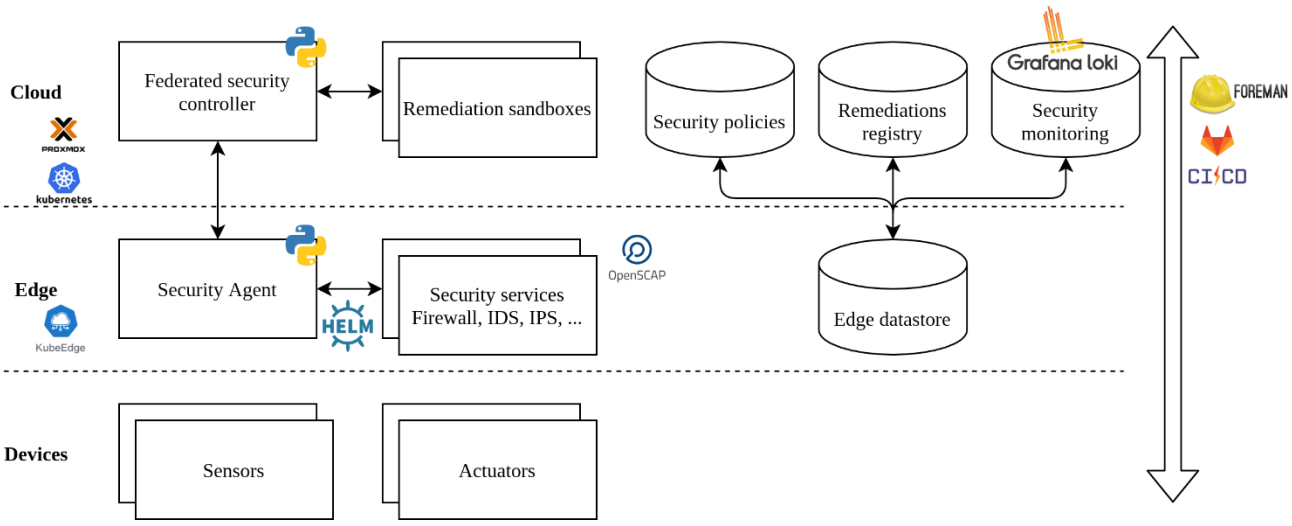


Figure 45: Vertical 1, Scenario 2 - CETIC testbed implementation and deployment view

Figure 46 is a picture of a platoon of rovers in the CETIC testbed, the platoon is composed of a leader and a follower. The platoon can be seen driving on the road between two wifi hotspots the Security Agents will reconfigure the security services to satisfy the new hotspot zone securthat simulate the traffic infrastructure. When the platoon changes from one hotspot to the other (based on signal strength), the Security Agents will reconfigure the security services to satisfy the new hotspot zone security policy.

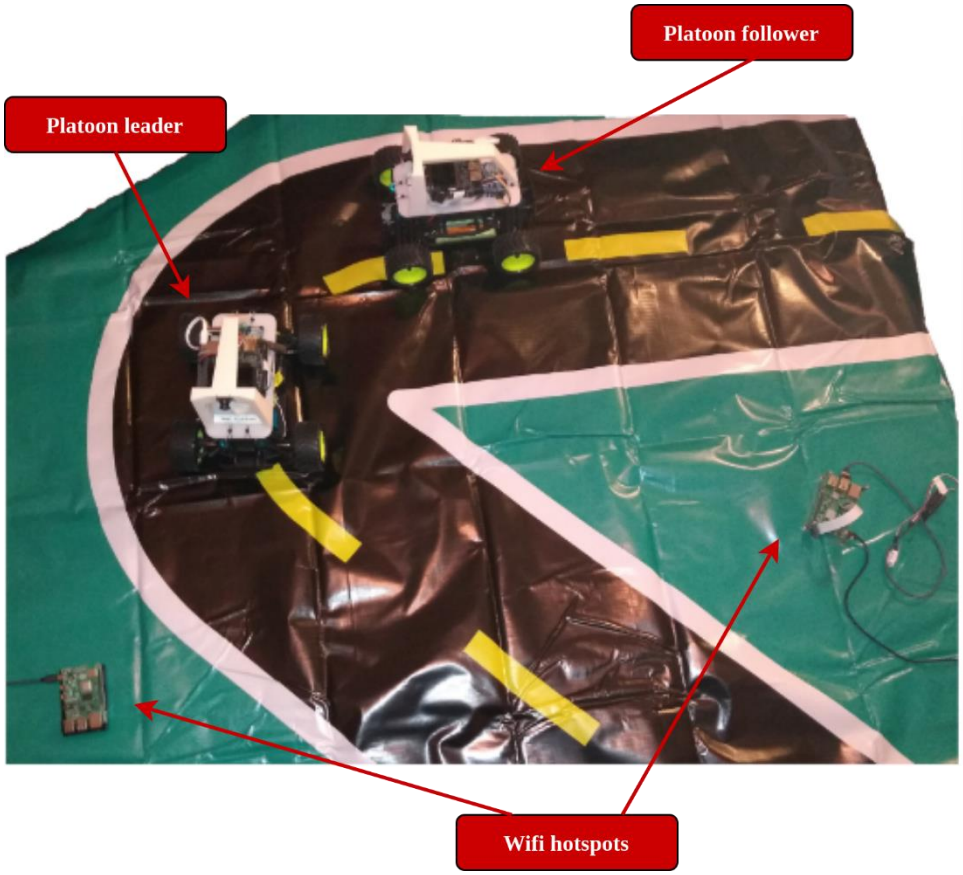


Figure 46: Vertical 1, Scenario 2- CETIC Edge testbed

To monitor the security events in the testbed, we use the Grafana Loki log aggregation system. The results can be viewed in Grafana web interface, where the logs and events can be explored and monitored in dashboards through various visualisations. Figure 47 presents a sample dashboard that shows the timeline of events and corresponding logs when a platoon drives between edges: vulnerability scans and security remediations are applied as needed when the security policy changes.

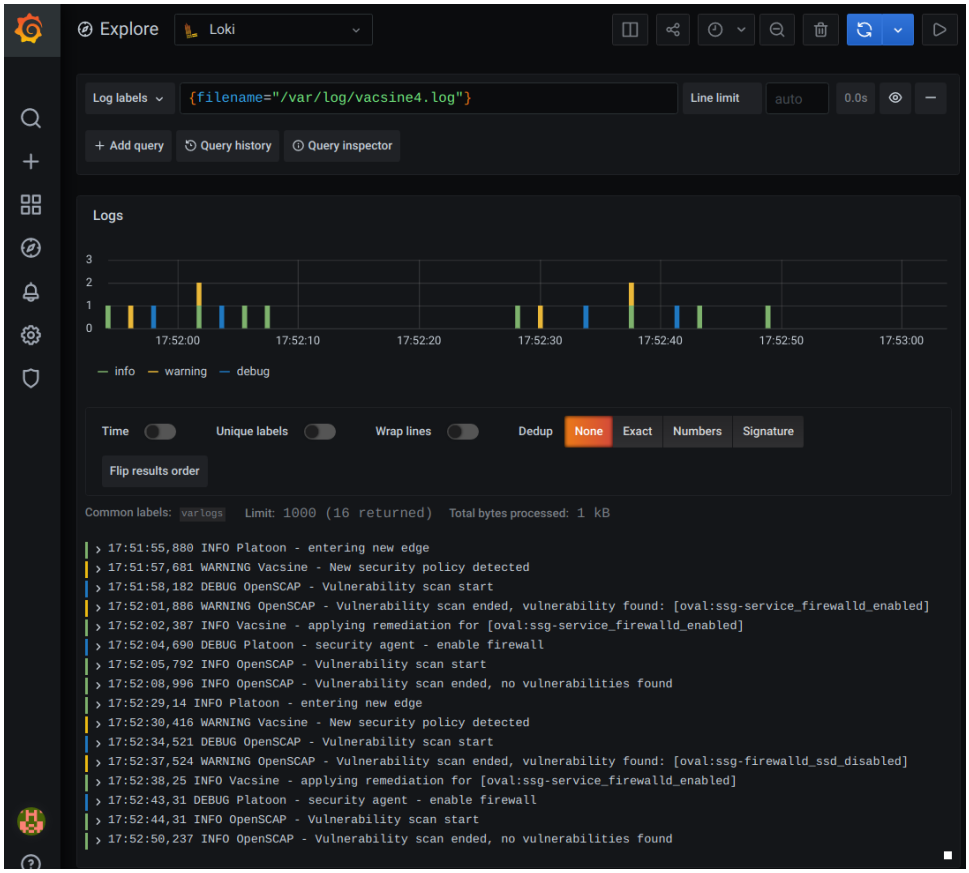


Figure 47: Vertical 1, Scenario 2- Grafana log and event dashboard

### 7.3.2 Verification and Validation

Verification and validation of this scenario will be done in D5.4. We describe briefly here our initial approach to validate the scenario implementation.

In this first iteration, we deployed the platoon on the CETIC testbed with a default security policy and perform vulnerability scans to check the default security policy is verified on the platoon. While the platoon drives between zones with different security policies, the security services (in this first iteration, the firewall) are redeployed and reconfigured. This produced compliance evidence in the form of vulnerability scan reports and logs for the security services orchestration. Figure 48 provides a sample SCAP content that corresponds to a security policy where the firewall needs to be running and the ssh port should be closed.

```
<ns3:definition class="compliance" id="oval:ssg-service_firewalld_enabled:def:1" version="1">
    <ns3:metadata>
    <ns3:title>Service firewalld Enabled</ns3:title>
    <ns3:affected family="unix">
        <ns3:platform>Red Hat Enterprise Linux 7</ns3:platform>
    </ns3:affected>
    <ns3:description>The firewalld service should be enabled if possible.</ns3:description>
    <ns3:reference ref_id="CCE-80998-8" source="CCE"/>
    <ns3:reference ref_id="service_firewalld_enabled" source="ssg"/>
    </ns3:metadata>
    <ns3:criteria comment="package firewalld installed and service firewalld is configured to start" operator="AND">
    <ns3:extend_definition comment="firewalld installed" definition_ref="oval:ssg-package_firewalld_installed:def:1"/>
    <ns3:criteria comment="service firewalld is configured to start and is running" operator="AND">
        <ns3:criterion comment="firewalld is running" test_ref="oval:ssg-test_service_running_firewalld:tst:1"/>
        <ns3:criteria comment="service firewalld is configured to start" operator="OR">
        <ns3:criterion comment="multi-user.target wants firewalld" test_ref="oval:ssg-test_multi_user_wants_firewalld:tst:1"/>
        <ns3:criterion comment="multi-user.target wants firewalld socket" test_ref="oval:ssg-test_multi_user_wants_firewalld_socket:tst:1"/>
        </ns3:criteria>
    </ns3:criteria>
    </ns3:criteria>
</ns3:definition>
<ns3:definition class="compliance" id="oval:ssg-firewalld_sshd_disabled:def:1" version="1">
    <ns3:metadata>
    <ns3:title>Disallow inbound firewall access to the SSH Server port</ns3:title>
    <ns3:affected family="unix">
        <ns3:platform>Red Hat Enterprise Linux 7</ns3:platform>
    </ns3:affected>
    <ns3:description>If inbound SSH access is not needed, the firewall should disallow or reject access to the SSH port (22).</ns3:description>
    <ns3:reference ref_id="CCE-80218-1" source="CCE"/>
    <ns3:reference ref_id="firewalld_sshd_disabled" source="ssg"/>
    </ns3:metadata>
    <ns3:criteria operator="AND">
    <ns3:criterion comment="ssh service is not enabled in services" test_ref="oval:ssg-test_firewalld_service_sshd:tst:1"/>
    <ns3:criterion comment="ssh port is not enabled in services" test_ref="oval:ssg-test_firewalld_service_sshd_port:tst:1"/>
    <ns3:criterion comment="ssh service is not enabled in zones" test_ref="oval:ssg-test_firewalld_zone_sshd:tst:1"/>
    <ns3:criterion comment="ssh port is not enabled in zones" test_ref="oval:ssg-test_firewalld_zone_sshd_port:tst:1"/>
    </ns3:criteria>
</ns3:definition>
```

Figure 48: Sample security policy – Extract of SCAP/OVAL ssg-rhel7

We use Foreman[21] combined with Gitlab-CI to orchestrate the DevSecOps pipeline. Figure 49 shows the Foreman web interface for a non-compliant vehicle, this is the result of a vulnerability scan using OpenSCAP.

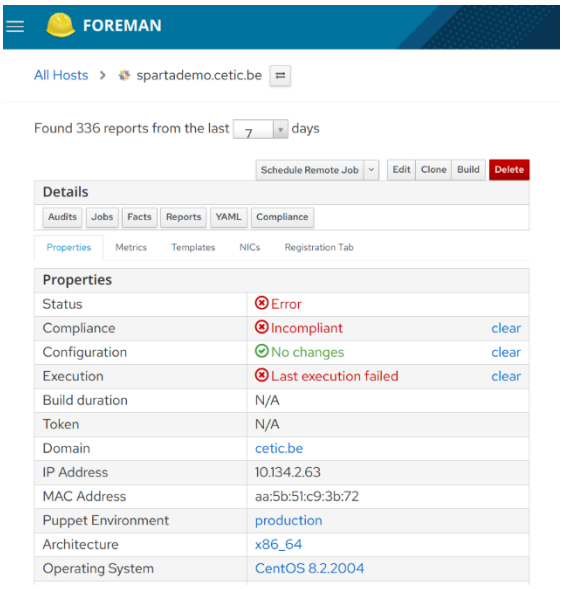


Figure 49: Compliance status in Foreman

In the next iteration, we will validate inside the continuous assessment pipeline that the security requirements stay satisfied by correlating the security orchestration logs and the vulnerability scan reports.

[21] https://www.theforeman.org/

## 7.4 Scenario 3: Verification tooling

### 7.4.1 Modelling and Implementation

As commented in chapter 4.3.2 the tools to perform the penetration testing phase are being prepared.

On one side, there is the collection of information and exploration of already known vulnerabilities. Special focus here are the WiFI attacks as main vector with exploitable results, but also python, TLS, Raspberry and Broadcom chipsets are studied. Here below there is the complete list of CVEs found on the different attack vectors.

**TLS1.3**

| CVE | Description | Applicability |
|---|---|---|
| CVE-2018-19608 | Arm Mbed TLS before 2.14.1, before 2.7.8, and before 2.1.17 allows a local unprivileged attacker to recover the plaintext of RSA decryption, which is used in RSA-without-(EC)DH(E) cipher suites. | Local attacks are not contemplated in this pen-testing. To check if it is possible somehow to access locally via WiFI vector. |
| CVE-2018-12404 | A cached side channel attack during handshakes using RSA encryption could allow for the decryption of encrypted content. This is a variant of the Adaptive Chosen Ciphertext attack (AKA Bleichenbacher attack) and affects all NSS versions prior to NSS 3.41. | Local attacks are not contemplated in this pen-testing. To check if it is possible somehow to access locally via WiFI vector. |
| CVE-2018-16868 | A Bleichenbacher type side-channel based padding oracle attack was found in the way gnutls handles verification of RSA decrypted PKCS#1 v1.5 data. An attacker who is able to run process on the same physical core as the victim process, could use this to extract plaintext or in some cases downgrade any TLS connections to a vulnerable server. | Local attacks are not contemplated in this pen-testing. To check if it is possible somehow to access locally via WiFI vector. |
| CVE-2018-16869 | A Bleichenbacher type side-channel based padding oracle attack was found in the way nettle handles endian conversion of RSA decrypted PKCS#1 v1.5 data. An attacker who is able to run a process on the same physical core as the victim process, could use this flaw extract plaintext or in some cases downgrade any TLS connections to a vulnerable server. | Local attacks are not contemplated in this pen-testing. To check if it is possible somehow to access locally via WiFI vector. |
| CVE-2018-16870 | It was found that wolfssl before 3.15.7 is vulnerable to a new variant of the Bleichenbacher attack to perform downgrade attacks against TLS. This may lead to leakage of sensible data. | Local attacks are not contemplated in this pen-testing. To check if it is possible somehow to access locally via WiFI vector. |
| CVE-2019-6659 | On version 14.0.0-14.1.0.1, BIG-IP virtual servers with TLSv1.3 enabled may experience a denial of service due to undisclosed incoming messages. | |

| CVE | Description | Applicability |
|---|---|---|
| CVE-2020-24613 | wolfSSL before 4.5.0 mishandles TLS 1.3 server data in the WAIT_CERT_CR state, within SanityCheckTls13MsgReceived() in tls13.c. This is an incorrect implementation of the TLS 1.3 client state machine. This allows attackers in a privileged network position to completely impersonate any TLS 1.3 servers, and read or modify potentially sensitive information between clients using the wolfSSL library and these TLS servers. | Local attacks are not contemplated in this pen-testing. To check if it is possible somehow to access locally via WiFI vector. |
| CVE-2020-1968 | The Raccoon attack exploits a flaw in the TLS specification which can lead to an attacker being able to compute the pre-master secret in connections which have used a Diffie-Hellman (DH) based ciphersuite. In such a case this would result in the attacker being able to eavesdrop on all encrypted communications sent over that TLS connection. The attack can only be exploited if an implementation re-uses a DH secret across multiple TLS connections. Note that this issue only impacts DH ciphersuites and not ECDH ciphersuites. This issue affects OpenSSL 1.0.2 which is out of support and no longer receiving public updates. OpenSSL 1.1.1 is not vulnerable to this issue. Fixed in OpenSSL 1.0.2w (Affected 1.0.2-1.0.2v). | If not in the last version of openSSL, it could be exploited. |
| CVE-2020-1967 | Server or client applications that call the SSL_check_chain() function during or after a TLS 1.3 handshake may crash due to a NULL pointer dereference as a result of incorrect handling of the "signature_algorithms_cert" TLS extension. The crash occurs if an invalid or unrecognised signature algorithm is received from the peer. This could be exploited by a malicious peer in a Denial of Service attack. OpenSSL version 1.1.1d, 1.1.1e, and 1.1.1f are affected by this issue. This issue did not affect OpenSSL versions prior to 1.1.1d. Reported by Bernd Edlinger. | It could be exploited if not last version of OPENSSL |
| CVE-2020-24659 | An issue was discovered in GnuTLS before 3.6.15. A server can trigger a NULL pointer dereference in a TLS 1.3 client if a no_renegotiation alert is sent with unexpected timing, and then an invalid second handshake occurs. The crash happens in the application's error handling path, where the gnutls_deinit function is called after detecting a handshake failure. | It could be exploited if GNUTLS is used and not the final version SW |
| CVE-2020-13777 | GnuTLS 3.6.x before 3.6.14 uses incorrect cryptography for encrypting a session ticket (a loss of confidentiality in TLS 1.2, and an authentication bypass in TLS 1.3). The earliest affected version is 3.6.4 (2018-09-24) because of an error in a 2018-09-18 commit. Until the first key rotation, the TLS server always uses wrong data in place of an encryption key derived from an application. | It could be exploited if GNUTLS is used and not the final version SW |

Table 27: TLS selected CVEs

### WiFi 802.11n

| CVE | Description | Applicability |
|---|---|---|
| CVE-2019-15126 (Broadcom)<br>CVE-2020-3702(Qualcomm) | KROOK attacks. An issue was discovered on Broadcom Wi-Fi client devices. Specifically timed and handcrafted traffic can cause internal errors (related to state transitions) in a WLAN device that lead to improper layer 2 Wi-Fi encryption with a consequent possibility of information disclosure over the air for a discrete set of traffic, a different vulnerability than CVE-2019-9500, CVE-2019-9501, CVE-2019-9502, and CVE-2019-9503. | To be checked, as some devices use Broadcom chipsets. |
| CVE-2017-13077, CVE-2017-13078, CVE-2017-13079, CVE-2017-130780, CVE-2017-130781, CVE-2017-13082 CVE-2017-13084, CVE-2017-13086, CVE-2017-13087, CVE-2017-13088 | This CVE are all the representative of the KRAK attacks. Wi-Fi Protected Access (WPA and WPA2) allows reinstallation of the Pairwise Transient Key (PTK) Temporal Key (TK) during the four-way handshake, allowing an attacker within radio range to replay, decrypt, or spoof frames. | To be checked if not yet updated with last FW. |
| Karma attacks | Karmetasploit is a great function within Metasploit, allowing you to fake access points, capture passwords, harvest data, and conduct browser attacks against clients. | |

Table 28: WI-Fi selected CVEs

### Rasbperry Pi (module B+)

| CVE | Description | Applicability |
|---|---|---|
| CVE-2018-18068 | The ARM-based hardware debugging feature on Raspberry Pi 3 module B+ and possibly other devices allows non-secure EL1 code to read/write any EL3 (the highest privilege level in ARMv8) memory/register via inter-processor debugging. With a debug host processor A running in non-secure EL1 and a debug target processor B running in any privilege level, the debugging feature allows A to halt B and promote B to any privilege level. As a debug host, A has full control of B even if B owns a higher privilege level than A. Accordingly, A can read/write any EL3 memory/register via B. Also, with this memory access, A can execute arbitrary code in EL3 | It may require physical access which is out of scope |

Table 29: Raspberry Pi selected CVE

### BROADCOM

| CVE | Description | Applicability |
|---|---|---|
| CVE-2017-9417 | Broadcom BCM43xx Wi-Fi chips allow remote attackers to execute arbitrary code via unspecified vectors, aka the "Broadpwn" issue. | There is a high possibility of success. Check also CVE-2017-11122, CVE-2017-11121, CVE-2017-11120 from similar family. |
| CVE-2018-19860 | Broadcom firmware before summer 2014 on Nexus 5 BCM4335C0 2012-12-11, Raspberry Pi 3 BCM43438A1 2014-06-02, and unspecifed other devices does not properly restrict LMP commnds and executes certain memory contents upon receiving an LMP command, as demonstrated by executing an HCI command | There are possibilities to exploit |

Table 30: Broadcom selected CVEs

### Python 2.7 Server – client

| CVE | Description | Applicability |
|---|---|---|
| CVE-2020-25658 | It was found that python-rsa is vulnerable to Bleichenbacher timing attacks. An attacker can use this flaw via the RSA decryption API to decrypt parts of the cipher text encrypted with RSA. | As there is TLS also, it may be interesting |
| CVE-2020-8492 | Python 2.7 through 2.7.17, 3.5 through 3.5.9, 3.6 through 3.6.10, 3.7 through 3.7.6, and 3.8 through 3.8.1 allows an HTTP server to conduct Regular Expression Denial of Service (ReDoS) attacks against a client because of urllib.request.AbstractBasicAuthHandler catastrophic backtracking. | It could work |
| CVE-2020-27783 | A XSS vulnerability was discovered in python-lxml's clean module. The module's parser didn't properly imitate browsers, which caused different behaviors between the sanitizer and the user's page. A remote attacker could exploit this flaw to run arbitrary HTML/JS code. | It could work |
| CVE-2020-26116 | http.client in Python 3.x before 3.5.10, 3.6.x before 3.6.12, 3.7.x before 3.7.9, and 3.8.x before 3.8.5 allows CRLF injection if the attacker controls the HTTP request method, as demonstrated by inserting CR and LF control characters in the first argument of HTTPConnection.request. | It's python 3.0 but still enough interesting to list it here and maybe test it. |
| CVE-2019-10160 | A security regression of CVE-2019-9636 was discovered in python since commit d537ab0ff9767ef024f26246899728f0116b1ec3 affecting versions 2.7, 3.5, 3.6, 3.7 and from v3.8.0a4 through v3.8.0b1, which still allows an attacker to exploit CVE-2019-9636 by abusing the user and password parts of a URL. When an application parses user-supplied URLs to store cookies, authentication | It could be possibly affected if python is not updated |

| CVE | Description | Applicability |
|---|---|---|
| | credentials, or other kind of information, it is possible for an attacker to provide specially crafted URLs to make the application locate host-related information (e.g. cookies, authentication data) and send them to a different host than where it should, unlike if the URLs had been correctly parsed. The result of an attack may vary based on the application. | |
| CVE-2019-9947 | An issue was discovered in urllib2 in Python 2.x through 2.7.16 and urllib in Python 3.x through 3.7.3. CRLF injection is possible if the attacker controls a url parameter, as demonstrated by the first argument to urllib.request.urlopen with \r\n (specifically in the path component of a URL that lacks a ? character) followed by an HTTP header or a Redis command. This is similar to the CVE-2019-9740 query string issue. | It could be possibly affected if python is not updated |
| CVE-2019-9740 | An issue was discovered in urllib2 in Python 2.x through 2.7.16 and urllib in Python 3.x through 3.7.3. CRLF injection is possible if the attacker controls a url parameter, as demonstrated by the first argument to urllib.request.urlopen with \r\n (specifically in the query string after a ? character) followed by an HTTP header or a Redis command. | It could be possibly affected if python is not updated |
| CVE-2019-5010 | An exploitable denial-of-service vulnerability exists in the X509 certificate parser of Python.org Python 2.7.11 / 3.6.6. A specially crafted X509 certificate can cause a NULL pointer dereference, resulting in a denial of service. An attacker can initiate or accept TLS connections using crafted certificates to trigger this vulnerability.<br>. | CVE-2019-5010 is exploitable with network access, and does not require authorization privledges or user interaction. This vulnerability is considered to have a low attack complexity. It has the highest possible exploitability rating (3.9). The potential impact of an exploit of this vulnerability is considered to have no impact on confidentiality and integrity, and a high impact on availability |
| CVE-2018-20852 | http.cookiejar.DefaultPolicy.domain_return_ok in Lib/http/cookiejar.py in Python before 3.7.3 does not correctly validate the domain: it can be tricked into sending existing cookies to the wrong server. An attacker may abuse this flaw by using a server with a hostname that has another valid hostname as a suffix (e.g., pythonicexample.com to steal cookies for example.com). When a program uses http.cookiejar.DefaultPolicy and tries to do an HTTP connection to an attacker-controlled server, existing cookies can be leaked to the attacker. This affects 2.x through 2.7.16, 3.x before 3.4.10, 3.5.x before 3.5.7, 3.6.x before 3.6.9, and 3.7.x before 3.7.3. | It could be possibly affected if python is not updated |

Table 31: Python 2.7 selected CVEs

On the other side, a setup for the tools composed by a Raspberry PI 4 (which has possibilities to be used in monitor mode thanks to the Nexmon driver tool) is going to be used. Also an ultrasounds transceivers has been chosen and first integration tests with the raspberry have been integrated (see Figure 50)

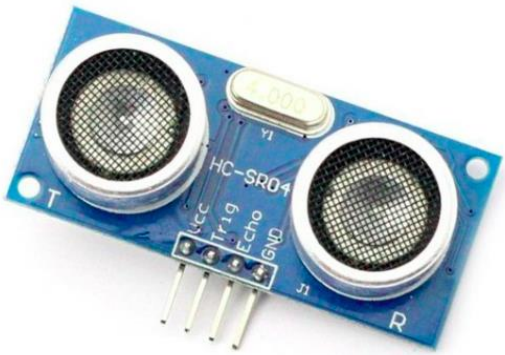


Figure 50: Ultrasound sensor used for pen-testing

The protection profile is also analysed to see if there is any possible fault in the protocol and the outputs of some tools (like AF3) are used in order to get a better overview of the system and understand all the possibilities.

## 7.5 Scenario 4: Safety and Security compliance assessment and certification

### 7.5.1 Modelling and Implementation

Some of the activities described in deliverable 5.2 have been developed in this first iteration of the scenario 4, including the digitalization of the safety and security standards, the creation of two Assurance projects, one per each standard, and the addition of the evidences into the Assurance projects.

First, the digitalization of the standards must be done. As mentioned in D5.1 [1] and D5.2 [2] the standards to be modelled with OpenCert are *ISO 26262 "Functional Safety Road Vehicles"* for functional safety and *SAE J3061 "Cybersecurity Guidebook for Cyber-Physical Vehicle System"* for cybersecurity. They will be created in the Reference Framework folder. Two new files will be created per standard, one file will show the standard modelled in a tree view folder and the other one in a diagram view. Figure 51 shows the diagram view with all the main activities and sub-activities of the ISO 26262 Standard.

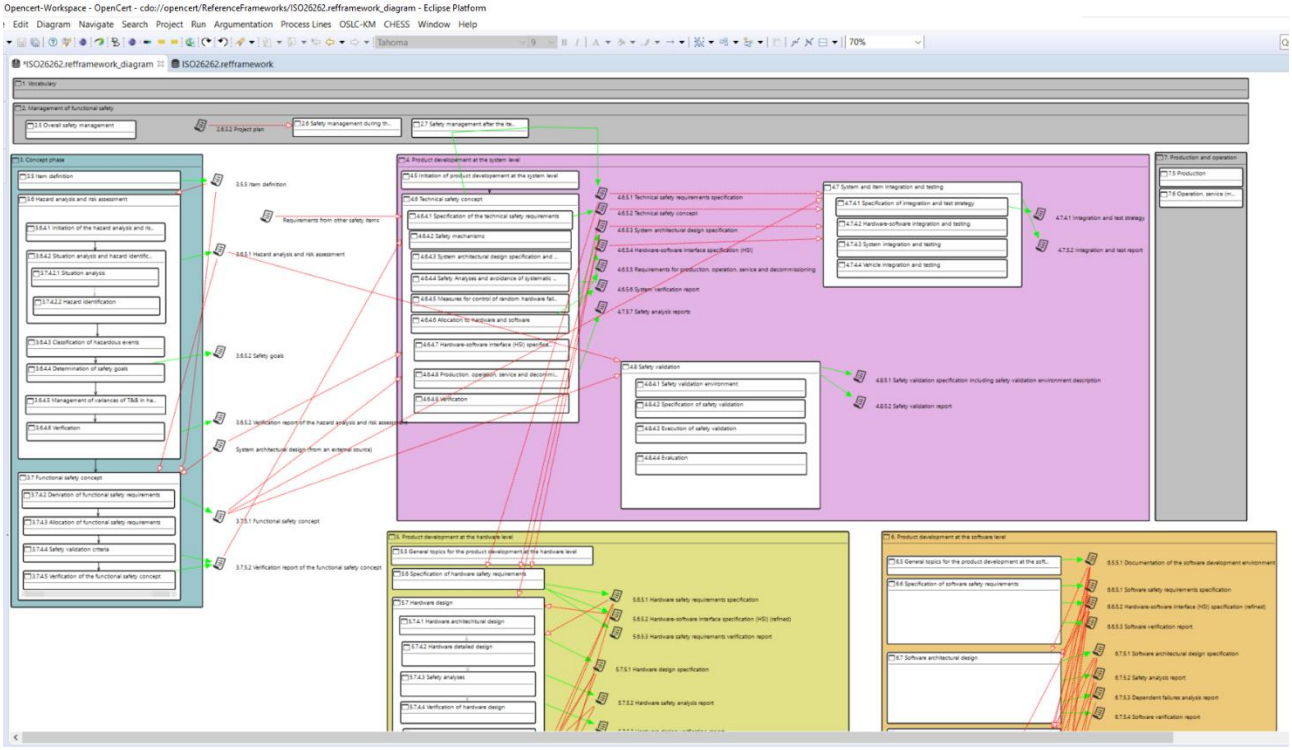

Figure 51: Diagram view of ISO 26262 in OpenCert

The standard is carefully digitalized where all the activities and their requirements are included. If necessary, we could specify for each requirement its applicability (recommended or highly recommended) and its criticality level (A, B, C, D). Figure 52 presents a specific requirement which recommends when different methods/artifacs should be applied depending on the criticality level (A, B, C, D).

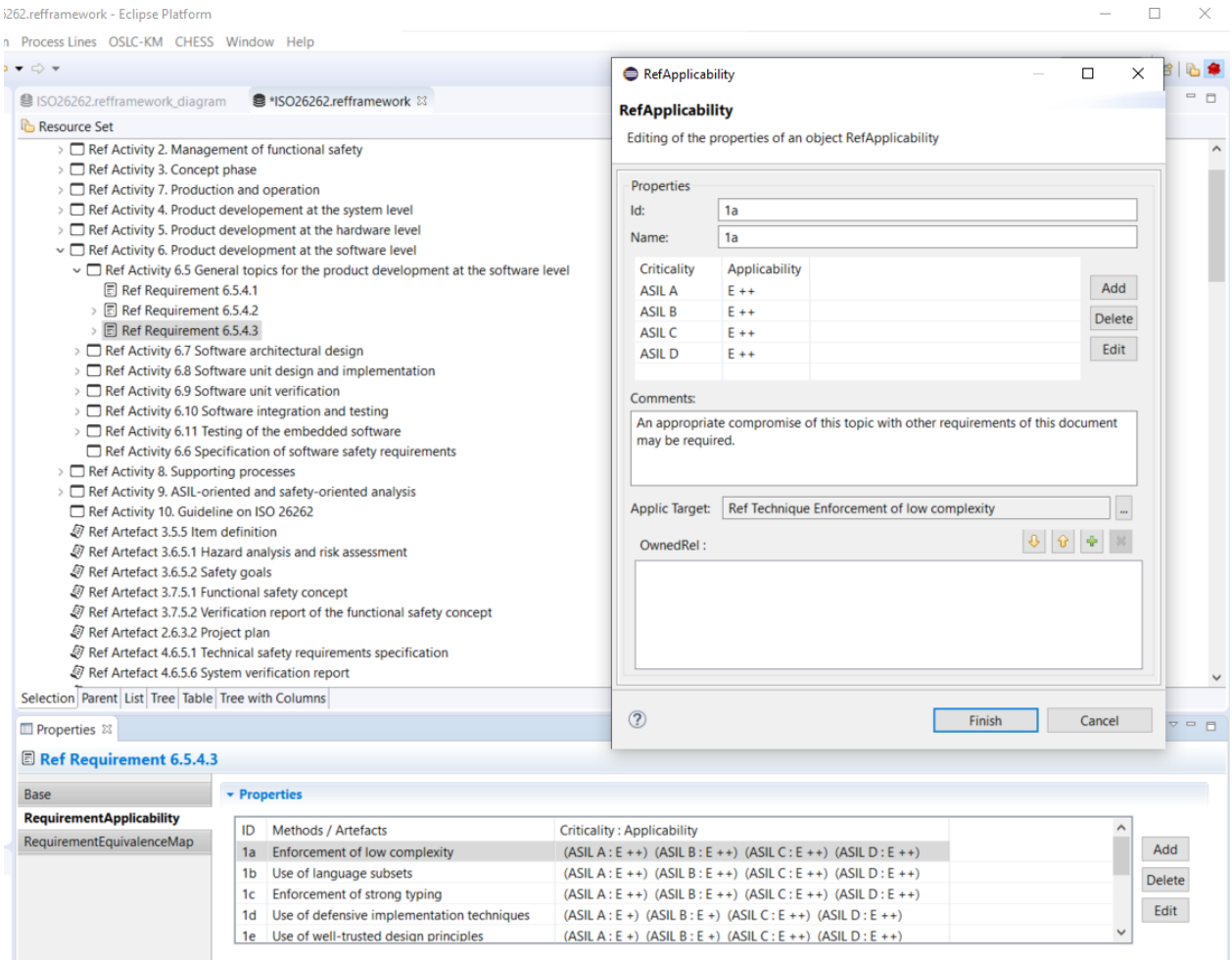


Figure 52: Applicability of methods depending on the criticality level

Figure 53 shows the diagram view of the *SAE J3061 "Cybersecurity Guidebook for Cyber-Physical Vehicle System".*

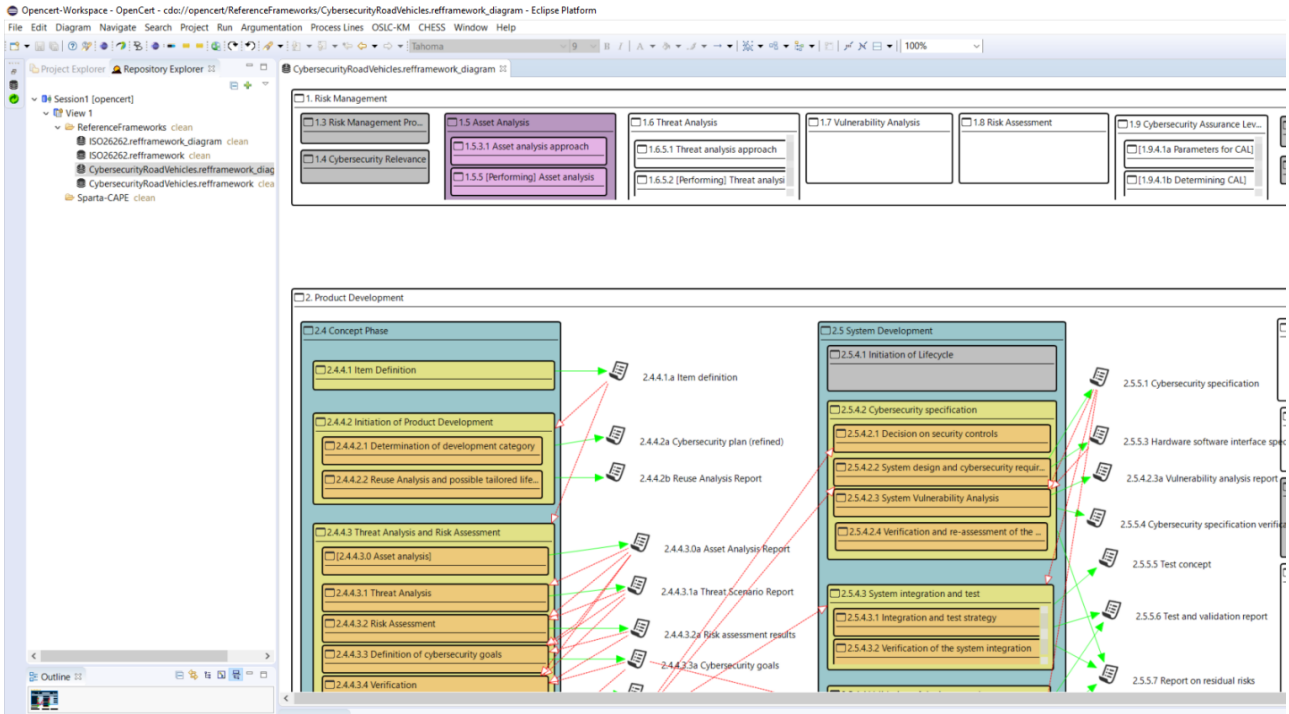


Figure 53: Diagram view of SAE J3061 in OpenCert

After the ISO 26262 and SAE J3061 standards are completely digitalized, we can create the two Assurance projects. When creating an Assurance project, OpenCert helps the engineer to select the activities and requirements to be fulfilled according to the level of criticality of the system.

The Safety Engineer and the Security Engineer follow the steps of both standards in parallel but always having each other into consideration. They must be in constant communication to avoid incongruities where one standard may conflict with the other.

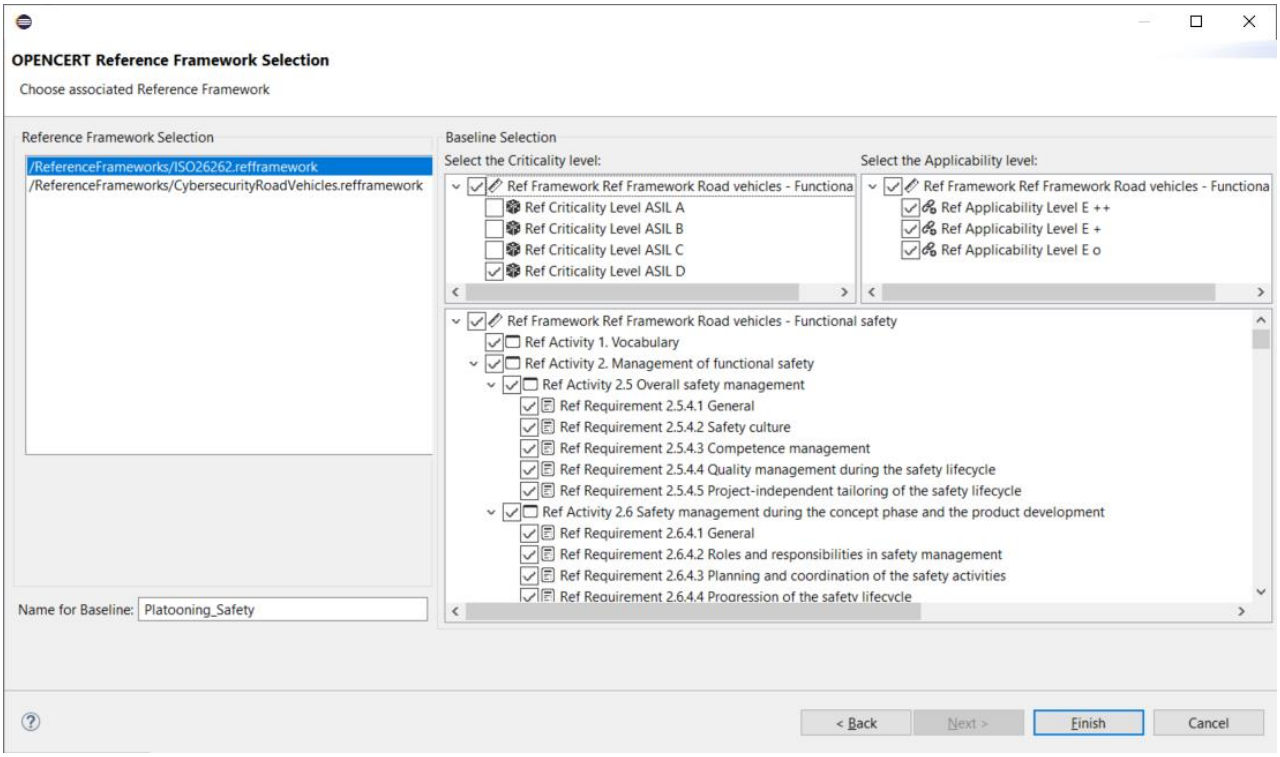


Figure 54: Generation of an Assurance project for ISO 26262

In the first iteration of the scenario we have included several evidences in the Assurance projects of both standards:

- In the ISO 26262 Assurance project, we have added the HAZARD analysis for the Connected Car Vertical that was documented in D5.1, resulting in a criticality level D for the platooning. Furthermore, we have added the FMEA analysis that was documented in D5.2 Annex A. Figure 55 shows the FMEA evidence in OpenCert and how it is linked to its corresponding requirement.
- In the SAE J3061 Assurance project, we have added the Protection Profile document that was documented in D5.2 Annex B as an evidence of the security requirements definition.

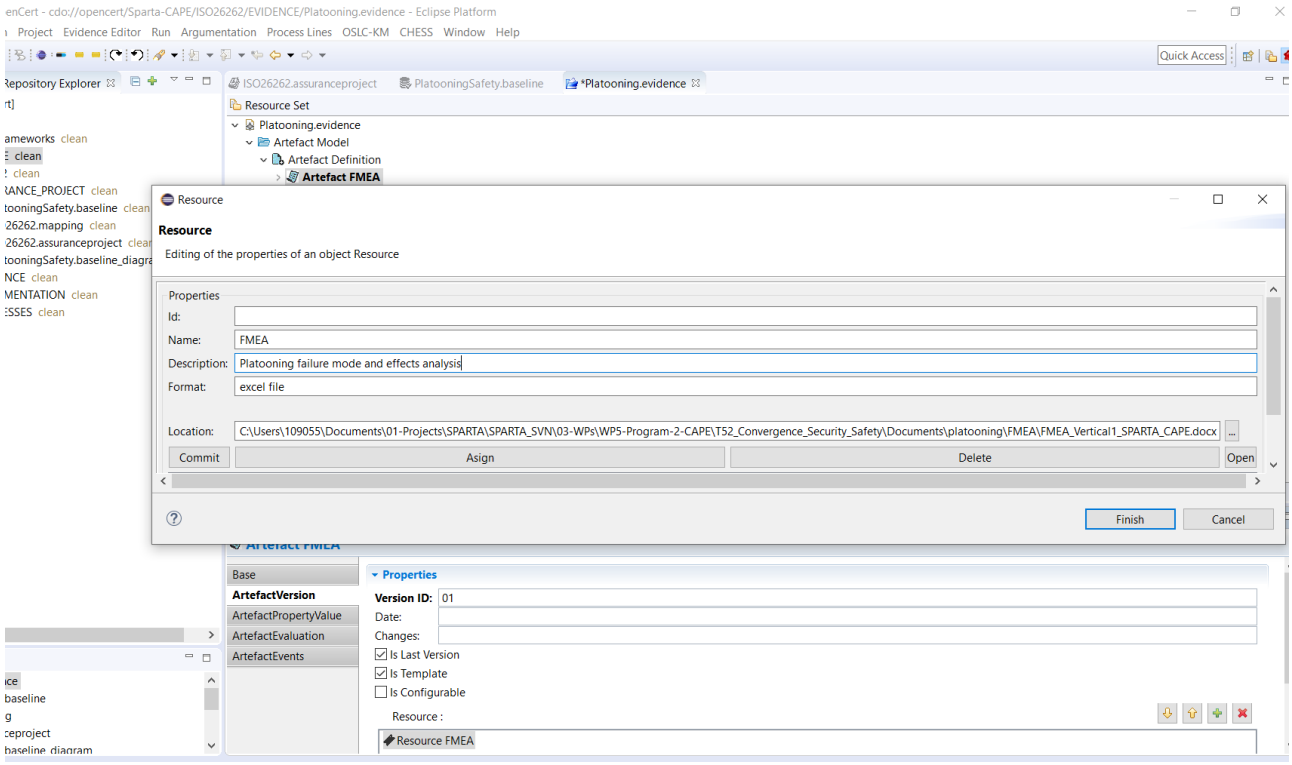


Figure 55: FMEA evidence added in the ISO 26262 Assurance project

In the next iteration of the scenario, we will add more evidences to the Assurance projects, and will make the necessary arguments. These arguments, which will be defined in a structured way called "Assurance Case", are those that allow specifying convincing justification that a system is adequately safety and secure.

### 7.5.2 Verification and Validation

The verification and validation of this scenario will be documented in D5.4.

## 7.6 Scenario 5: Fault-injection and analysis of faulty scenarios

### 7.6.1 Modelling and Implementation

In the first iteration of the scenario, we have implemented a proof of concept with the Sabotage tool to carry out a preliminary analysis of a security mechanism before being integrated in the Connected Car Vertical.

We have modelled a mathematical model in the Matlab/Simulink tool, through the predefined blocks of its library, to detect whether the information (speed of the vehicle in front) received through the WiFi communications of one of the vehicles is false or not. This information is contrasted with the information that this vehicle receives from its proximity sensor (ultrasonic sensor). The mathematical algorithm is called a sensor-based plausibility check. Figure 56 shows some of the types of failures that have been tested with this model.

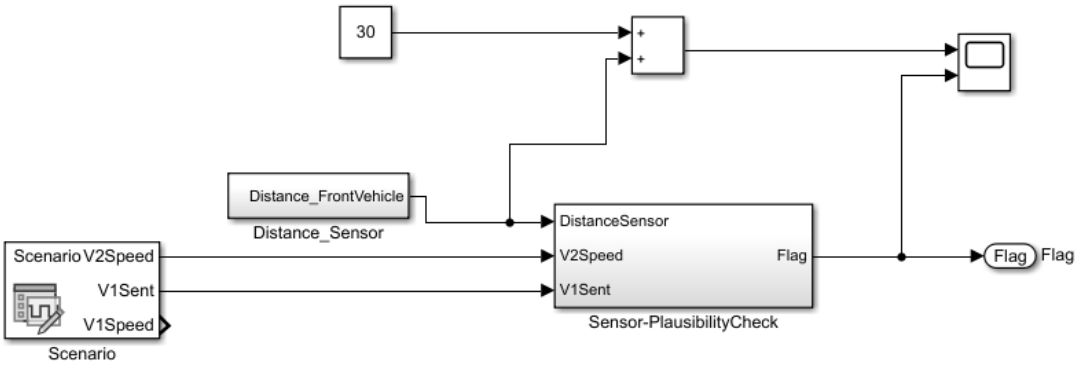


Figure 56: Plausibility check model

Once the model has been created in Simulink, it is time to start testing it to make sure the algorithm works as it should. To know if the algorithm is working correctly, we must inject faults to verify that it really detects when there is a false input value or when it is a true value. By defining what type of fault we are going to inject, where, when, and for how long, we will be able to visualize its response. Figure 57 shows the type of failures that have been tested on this model.

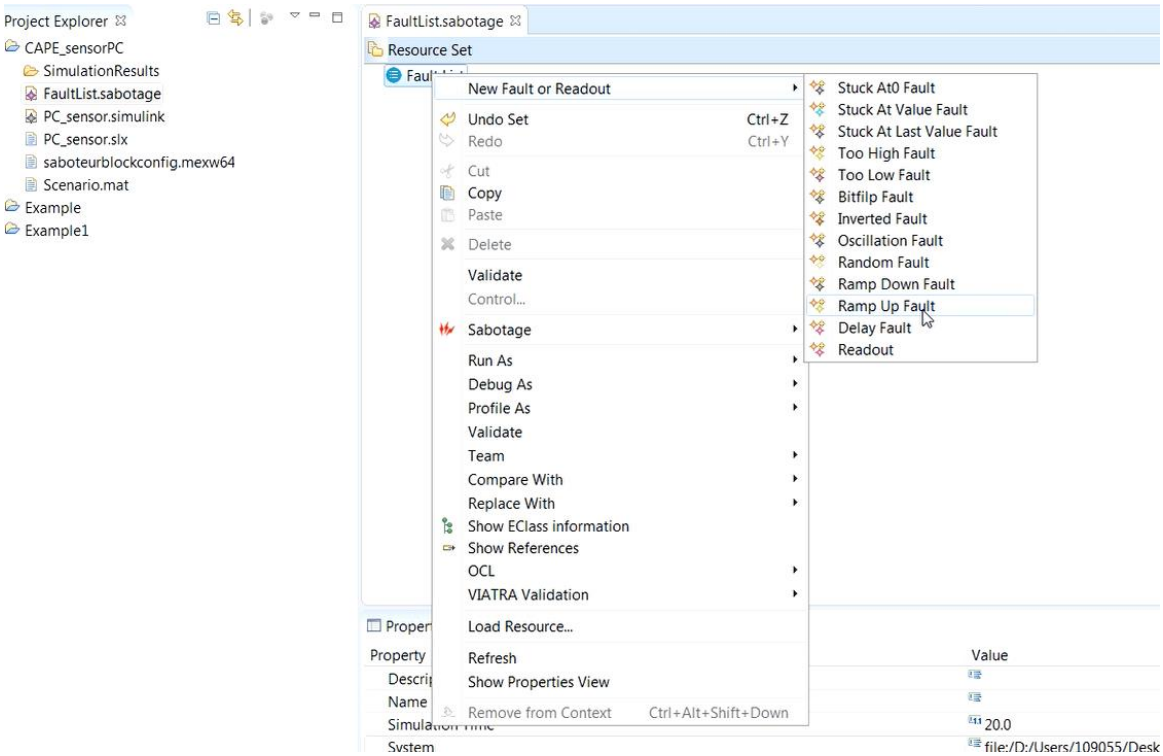


Figure 57: Fault types

All these simulations will help to know if the engineer has to configure better the fault tolerance of the algorithm or, if on the contrary, there is some kind of fault that s/he is unable to detect. If that is the case the engineer should redefine the algorithm again. Once the algorithm has been redefined, all the previous tests must be carried out again to ensure that this time they have been detected. In Figure 58 there is an example of a test with a “ramp up” fault and we can be seen that the algorithm works properly.

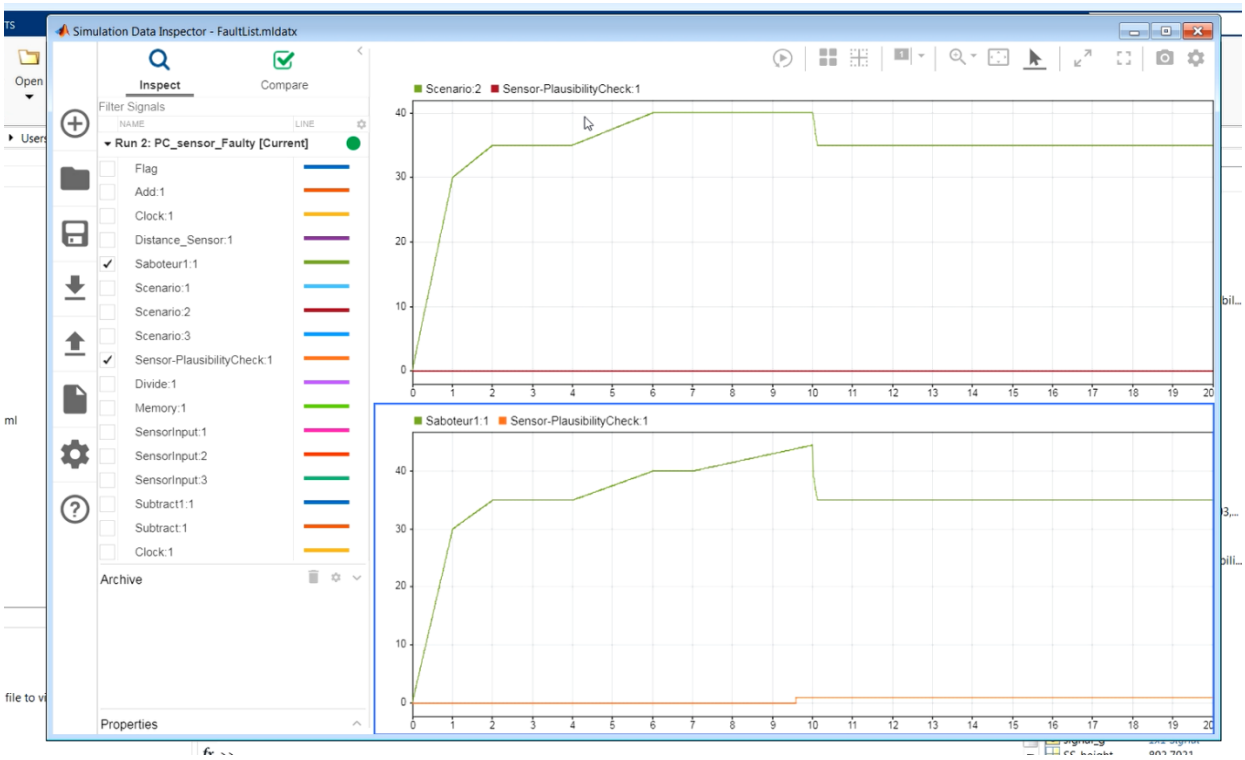


Figure 58: Example of simulation results

In the next iteration of the scenario, we will perform more simulations to inject other types of failures into the algorithm, and the required corresponding modifications will be done before including them into the vehicle.

### 7.6.2 Verification and Validation

The verification and validation of this scenario will be documented in D5.4.

# Chapter 8 Vertical 2 - Prototypes for e-Government Services

## 8.1 Introduction

This chapter illustrates the prototypes for demonstration of the "Complex System Assessment Including Large Software and Open-Source Environments, Targeting e-Government Services" vertical (a.k.a. e-Government services vertical), having as goal to improve the cyber-security of the innovative authentication solutions based on the usage of the Italian national electronic identity card (CIE). The main use case is based on the use of an app on the smartphone to interact with the CIE (through the NFC interface) as an authentication tool to gain access to the services offered by the Public Administration.

In Section 8.2 we present the technical aspects of the selected scenarios, by describing the environment on which the demonstration will take place.

In Section 8.3 we describe the demonstrator prototypes we developed aiming to show how the CAPE tools contribute to secure the CIE selected scenarios.

## 8.2 Scenarios

As mentioned in D5.2, for the demonstration of vertical 2 we identified two main components, as depicted in Figure 59:

- CIE ID APP, and
- SAML IdP on the CIE ID SERVER


Figure 59: Components in the scope of the demonstrations

For each component we defined a demonstration scenario. In D5.2 we mainly focused on the functional and security aspects of these scenarios, while in the following subsections we provide more details concerning the used technology of each scenario. The aim is to specify the environment on which the demonstration will take place.

### 8.2.1 Scenario for the CIE ID APP

CieID is the app developed by the Italian National Mint and Printing House for access, leveraging the electronic identity card (CIE 3.0), to the services of the Italian Public Administrations and the

services provided by the member states of the European Union under the regulation EU 910/2014 eIDAS.

The app has been developed in Kotlin programming language, and it required Android 6.0 and later (API level > 23) equipped with NFC interface. The app integrates third-party libraries to provide specific functionalities and allows the users to make the most of the potential offered by the app. Two examples of the third-party libraries used in the app are okhttp, a library in charge of handling all HTTP calls, and Firebase Crashlytics, a lightweight real-time crash reporter that helps to track, prioritize, and fix stability issues that erode the app quality. The app is available in the official Google Play Store and currently has more than 100.000 installations.

In the context of the SPARTA project, CINI has extended the Gitlab environment in such a way to use the continuous integration functionalities offered by Gitlab to automatically build the APK file after each commit in the repository.

### 8.2.2 Scenario for the SAML IdP

SAML IdP is based on Shibboleth, a standard-based, open-source software package for Single Sign-on (SSO) system across or within organizational boundaries. It allows sites to make informed authorization decisions for individual access of protected online resources in a privacy-preserving manner. SAML IdP is responsible for supplying information about users at a domain to relying parties protected by service providers leveraging the information contains in the CIE.

The Shibboleth's IdP workflow forecasts that when an HTTP request arrives at a request dispatcher, it inspects the request and, based on its properties, sends it to a profile manager. A profile manager, as the name suggests, is designed to handle a particular protocol profile request (for example, SAML 1 Attribute Query, SAML 2 Single Sign-On). In this case, the functions of sending requests and managing profiles are actually implemented by Spring Web Flow, which is located on top of the Spring MVC layer. Each request is mapped to a specific profile flow, the higher-level units of processing in the software. Each profile flow is composed of a set of actions that perform part of the overall process required to generate the appropriate response. Requests go from action to action until the response is complete and then return to the requester. At a high level, the actions are, for example, to authenticate the user, perform the resolution of attributes, sign the responses, etc. When an action does its job, it can, in turn, make calls to one or more IdP services. Each communication is through an API REST via HTTPS. Shibboleth is mostly a set of software components made using the Spring framework based on Java programming language and built with Apache Maven.

The SAML IdP is indeed a custom implementation of Shibboleth IdP deployed on an Azure Virtual Machine with two virtual CPUs and 8Gb of RAM. The OSx on the virtual machine is Ubuntu 20.04 with Java JDK 1.8 and Apache Tomcat version 9.0.40.

FBK has recently extended the Gitlab environment in such a way to use the continuous integration functionalities offered by Gitlab. Every time a git commit is pushed on the repository, the source code is automatically built and deployed (using Apache Maven) on an Azure virtual machine. It provides a running version of CIE ID SERVER, which is accessible for functional and security testing.

The workflow of the pipeline for the CI of the SAML IdP is composed of the following Stages, also depicted in Figure 60:

- **Build maven**: during this stage, a shell script is executed by the Gitlab runner, hosted on the Azure Virtual Machine, and make the build of the SAML IdP via Apache Maven.
- **Deploy artifact**: this stage is in charge of deploying the SAML IdP on the server where it is running. Moreover, it stops the Apache Tomcat service, copies the artifact from the previous stage, and then starts the service again.


Figure 60: Example of the status of a GitLab Stage during code submission

## 8.3 Demonstrators prototypes

In this section, we will describe the demonstrators prototypes we developed aiming to show how the CAPE tools contribute to secure the CIE ID APP and SAML IdP scenarios.

In particular, we will show how:

- we integrated in the development process of the CIE ID APP and SAML IdP (some of) the continuous integration techniques developed in the context of Task 5.3, and
- the tools perform a security assessment of the CIE ID APP and SAML IdP, providing a security report to the security analyst.

The demonstration scenarios of the vertical 2 involve the development and testing environments, where the preliminary versions of the components are developed and tested (before being migrated on the Italian Ministry of the Interior servers). They consist of a Gitlab platform hosted by FBK, and cloud-hosted Azure virtual machines.

Gitlab provides:

- a version control system (Git-repository), storing the source code of CIE ID APP and SAML IdP;
- Issues tracking and continuous integration and deployment pipeline.

The Azure virtual machines, running Linux distributions (Ubuntu) and supporting the Docker technology, are used to:

- host the deployed services for functional and security testing purposes, and
- run some of the CAPE tools.

The virtual machines hosting the tool must install the GitLab Runner application, which works with GitLab CI/CD to run jobs in a pipeline.

In the next sections we provide the technical details on how we integrated each CAPE tool in the demonstrator prototype, by following a continuous integration and DevSecOps approach.

### 8.3.1 Prototype for the DevSecOps pipeline of the CIE ID APP

To assess the security of the CIE ID APP, we deployed the DevSecOps scenario depicted in Figure 61. As described in Deliverable 5.2, the DevSecOps pipeline relies on a set of tools, namely APPROVER (CINI) and TSOpen (UNILU), to evaluate the security and risk requirements for the CIE ID mobile app.

The pipeline has been designed as a set of integration scripts that are attached to the GitLab repository hosting the mobile app source code. The different tools are either hosted in the Azure VM environment or available as remote services (SaaS) exposing REST APIs.

In the following of this section, we detail the current status of each of the involved tools' integration process, and we discuss the next steps to finalize the prototype. A detailed description of the achieved results will be included in Deliverable 5.4.

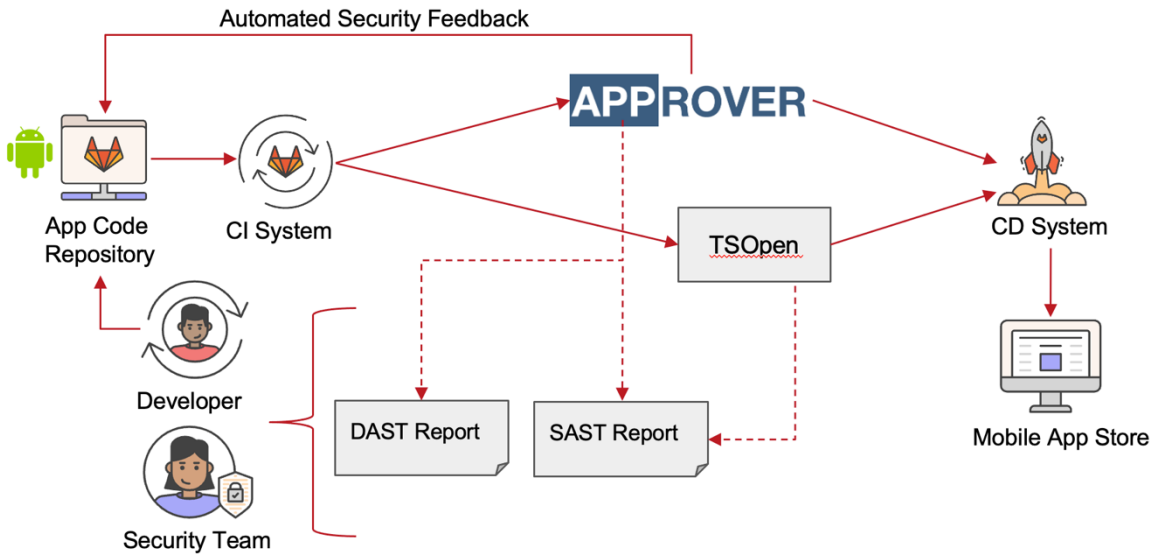


Figure 61: DevSecOps scenario for the CIE ID App

### Approver

The Approver tool, as discussed in Section 3.2.1.2, can be either used as a SaaS service or an on-premise solution.

For this prototype, we specifically developed a DevOps plugin compatible with GitLab (SR1.1) to integrate the usage of the Approver SaaS instance with the GitLab CI/CD system. Then we added a GitLab Script to invoke the Approver plugin in order to perform the security evaluation on the CIE ID app. The tool, as described in Section 3.2.1.4, is able to generate i) a SAST and DAST report – accessible using the APPROVER web interface and ii) a list of vulnerability issues – that are automatically reported as GitLab Issues.

The workflow of the DevSecOps pipeline for the Approver tool is composed of the following Stages:

- **Build-application**: this stage is in charge of building the CIE ID APP and creates an APK file, i.e., the bundle containing the compiled mobile app for Android environments.

- **Test with Approver**: during this stage, the APK is sent to the APPROVER service to perform the analysis. The tool automatically opens for every discovered security vulnerability – thanks to the DevOps-plugin – a GitLab Issue containing the description of the vulnerability, its severity, and a description of possible technical remediation. At the end of the stage, it reports the completion as shown in Figure 62.


Figure 62: DevSecOps - passed stage in the CI/CD process

### TSOpen

TSOpen is used to detect the presence of logic bombs in the APK and its dependencies (also present in the apk file). TSOpen can be used as a standalone tool or as a SaaS service. To be compliant with a DevSecOps pipeline, similarly to Approver, we specifically developed a DevOps plugin compatible with GitLab (SR1.1) to integrate the usage of the Approver SaaS instance with the GitLab CI/CD system. In particular, when TSOpen detects a security issue (i.e., in this case, the presence of a logic bomb), TSOpen will interact with GitLab to automatically open a security issue.

### 8.3.2 Prototype for the DevSecOps pipeline of the SAML IdP Server

To assess the security of the SAML IdP, we also deployed a DevSecOps scenario, as depicted in Figure 63. As described in Deliverable 5.2, the DevSecOps pipeline for the server software relies on a set of tools, namely Eclipse Steady and Project KB (SAP), SafeCommit (UNILU), Buildwatch (UBO), NeSSoS (CNR), and VI (UKON) to evaluate the security and risk requirements.

The pipeline has been designed as a set of integration scripts that are attached to the GitLab repository hosting the mobile app source code. The different tools are either hosted in the Azure VM environment or available as remote services (SaaS) exposing REST APIs.

In the following of this section, we detail the current status of each of the involved tools' integration process, and we discuss the next steps to finalize the prototype. A detailed description of the achieved results will be included in Deliverable 5.4.

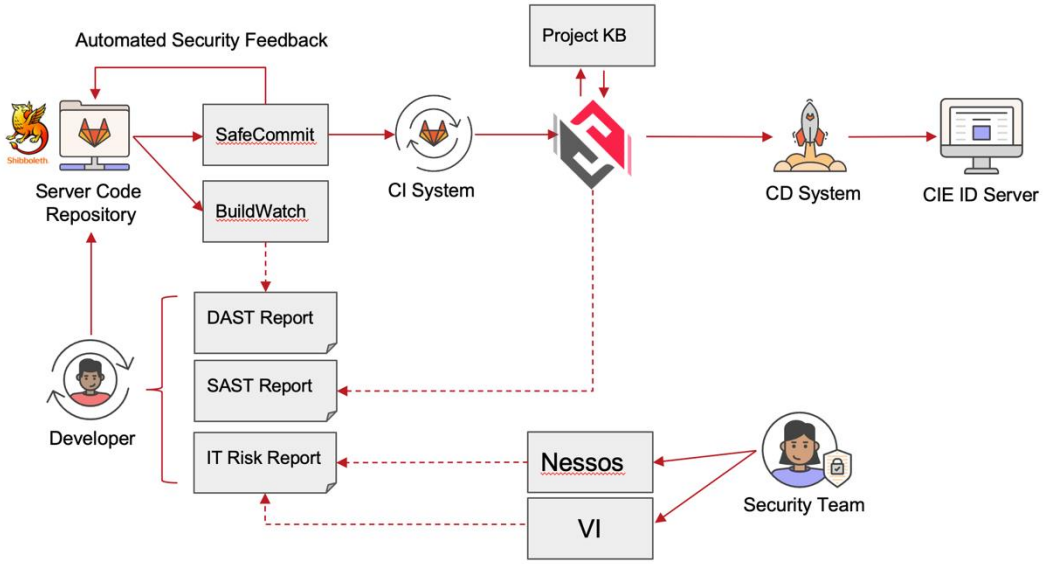


Figure 63: DevSecOps scenario for the SAML IdP Server

#### Steady

A dockerized version of Eclipse Steady runs on an Azure virtual machine (VM) managed by FBK (CINI). In the Azure VM, a Gitlab runner, an application that works with GitLab CI/CD to run jobs in a pipeline, is installed and connected to the Gitlab repository. At the end of the analysis made by Steady, a report is generated and available in the Gitlab repository where the SAML IdP source code is stored.

Stages:

- **Build-maven**: during this stage a shell script is executed by the Gitlab runner, hosted on the Azure Virtual Machine and make the build of the SAML IdP with Apache Maven.
- **Steady Analysis**: in this stage, the APP analysis goal of Steady's Maven plugin is executed in order to inspect and upload dependency information using the RESTful interface of Steady's server-side components. On the server, information about methods of open source dependencies is compared with methods known to be vulnerable. Additional goals such as A2C or T2C can be executed in order to collect reachability information.
- **Steady Report**: in this stage, the REPORT goal of Steady's Maven plugin is executed in order to download and process the vulnerabilities identified in the previous stage, and create a result report in HTML, JSON and XML format. Additionally, the plugin can be configured to break the stage in case of vulnerabilities, such that it appears as "failed" (cf. Figure 64). This allows the user to spot the issues and take the proper actions in order to solve them. When the report has been generated, the artifact is available on the Gitlab repository.


Figure 64: DevSecOps - failed stage in the CI/CD process of Steady

Steady offers a web GUI to navigate the report. This can be accessed by connecting in VPN to the Azure Virtual Machine in which Steady is running.

**SafeCommit**

SafeCommit is used to check whether a given commit performed by a developer introduces a vulnerability in the code base (hosted in the code repository). This tool will be integrated into Gitlab in a way that a security alert will be raised when a developer proposes a commit that could introduce a vulnerability.

**Buildwatch**

A dedicated machine is used to host Buildwatch. In order to trigger dynamic analysis, a Gitlab runner is leveraged. Buildwatch offers a simple REST API to which the runner may send requests for new analysis and corresponding poll results.

Stages:

- Gitlab runner requests a new analysis for a given commit.
- Buildwatch uses Cuckoo to boot an appropriate virtual machine.
- The software is build/tested according to the provided instructions.
- Cuckoo records all system calls.
- Based on the system calls, forensic artifacts are extracted.
- Known, benign, or whitelisted artifacts are removed from the extracted artifacts.
- Buildwatch provides the newly found forensic artifacts via the API to the Gitlab runner.
- Results may be displayed as results of the Gitlab pipeline job.

**NeSSoS**

In scope of this scenario, we are going to use NeSSoS to evaluate the security practices of the network where the server is running to ensure that it is managed securely. As the scenario is focused on a single server, the “network” under evaluation is limited to one server and the assets this sever contains. Many of the questions (inputs for NeSSoS tool) relate to the more generic practices applied in the company managing the server in question (e.g., involvement of the top management in security process, awareness level of the personnel, physical security of the area, security policies and culture enforced in the organisation, contingency plans, etc.), but they must be answered as well as they have direct impact on security management of the server.

NeSSoS tool is available on-line as a service and is free of charge. As for the process, security managers, responsible for security of the organisation (or the system in which the server is installed) should answer the questionnaire of NeSSoS in order to report the security practices applied in the organisation. Also, the NeSSoS tool will ask for the main assets managed by the server, its approximate quantity and possible impact in case of a security breach. Once the input is provided, the tool will provide the resulting risks and a report about estimated coverage of the basic security controls (as they are defined in ISO 27001/2:2013). The tool also has a capability to help the security analyst to improve security in a cost-efficient way. We would like to note that this capability should be used only if the server in question is indeed the only piece of the network. If the organisation has other IT facilities (e.g., other servers, valuable terminals, core business processes), the analysis should include them as well, otherwise, the tool will not be able to perform the cost-benefit analysis

correctly. The tool also has a functionality to use the inputs from other tools, which analyse concrete implementation of specific security features (e.g., vulnerability analysis). The possibility for other CAPE's tools to provide such information and their integration with NeSSoS will be analysed and implemented in the future.

**VI**

In the scenario's scope, we use the visual investigation (VI) of security information to examine software projects' exposure. The vulnerability explorer enables assessing the number of vulnerabilities in projects from an organization's point of view. The demonstrator displays the exposures in open-source components on the level of a whole software development organization. For instance, the interface allows discovering the most/least-used open-source components, the most/least-vulnerable open-source components, the most/least-vulnerable applications, or most/least-relevant vulnerabilities. The demonstrator utilizes the API of Eclipse Steady from SAP. A dockerized version of the VI demonstrator will be provided to display the scanned packages.

# Chapter 9 Roadmap and future work

As stated in D5.1, the roadmap for the CAPE program is based on the demonstration use cases for the Connected Car and e-government verticals as well as the various independent use-cases and is realised in a timeframe aligned with the deliverables. The next and last deliverable will be the D5.4.

- **D5.4 Demonstrators evaluation, M36, LEO:** This deliverable provides the demonstration of the two verticals described in T5.4. It is supported by an evaluation document.

A high-level development roadmap considering the deadlines of WP5 deliverables D5.2, D5.3 and D5.4 is as follows:

- January 2021 (M23): **Early prototypes** are available in D5.3. For each tool, a demo specification exists and has been included in D5.2.
- January 2022 (M35): **Final prototypes** have been evaluated in the respective demonstration scenarios; its summary is ready to be included in D5.4.

The timeline for the final prototypes of the T5.1 tools will be guided by the demonstration scenarios of the verticals and will focus on the integration on them:

- M25-M26: **refined design** of the tools in order to reach the final version of the prototypes
- M27-M28: **implementation** of the final prototype version of the tools
- M27-M35: **verification** and validation of the updated prototypes
- M29-M35: **integration** and evaluation of the framework tools on the various demonstration scenarios

The timeline for the final prototypes of the Connected Car vertical (in Task 5.2) is as follows:

- M25-M26: **modelling** of the requirements that have not been covered in the first iteration of the scenarios
- M27-M28: **verification** of the new developments
- M29: **update** analysis
- M30: **assessment** activities

The timeline for the evaluation activities in T5.4 is as follows:

- M18-M21: **Prototypes evaluability definition**
- M22-M25: **Prototypes evaluability materials production**
- M26-M35: **Prototype evaluability verification**

The following figure presents a timeline of the various activities that will be performed in the next steps of the CAPE tasks.

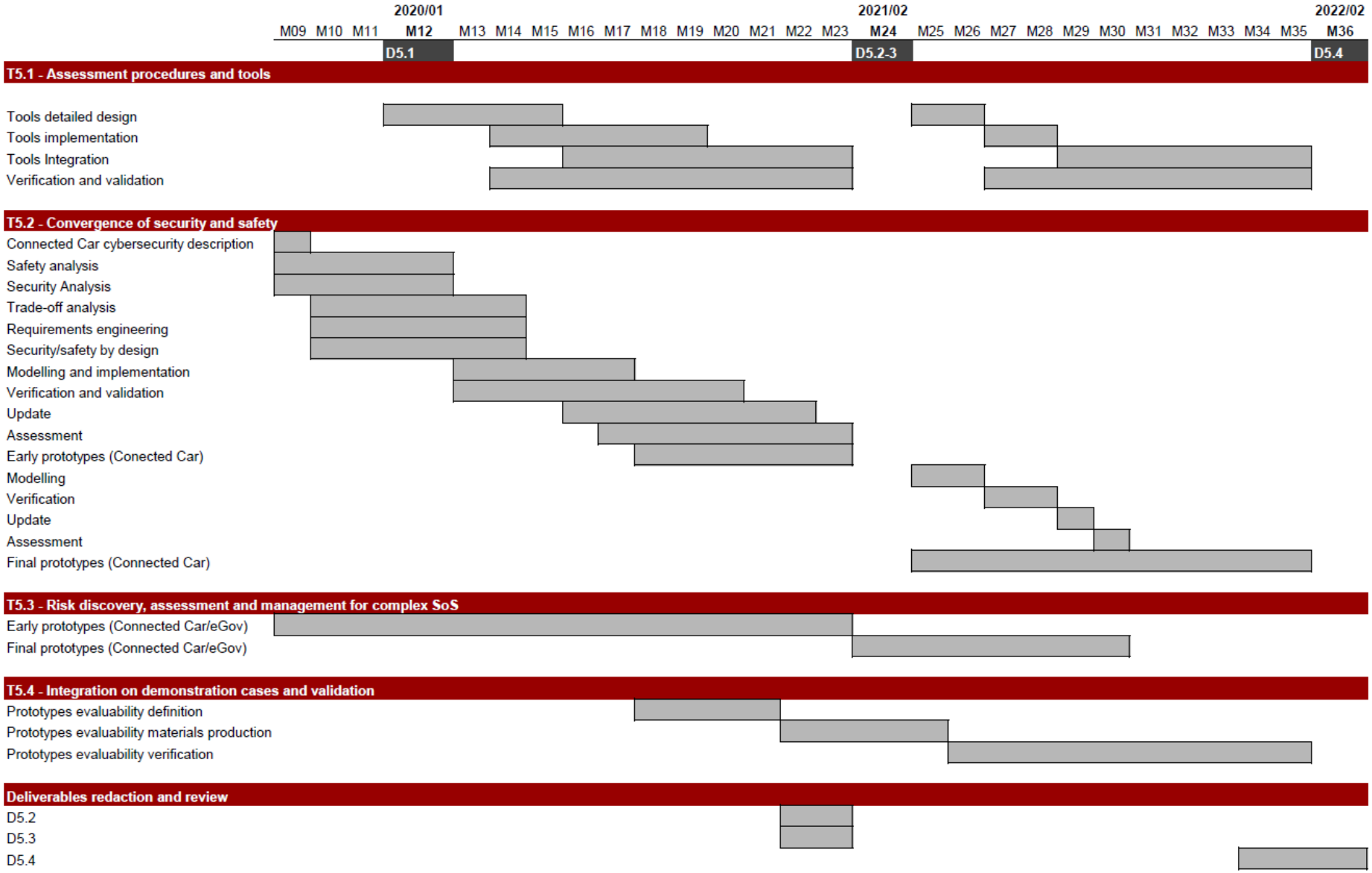


Figure 65: CAPE task roadmap planning

# Chapter 10 Summary and Conclusion

This deliverable contains contributions for each task and vertical in the context of the CAPE program.

In the first contribution, we analysed the prototypes in the context of T5.1 associated with each CAPE tool. We described the prototypes, their objectives, their technical characteristics, security features, components and constraints. At the end, we detailed the experimental tests, test environments and test results for each prototype.

For the second contribution, we contributed with methods and tools for the safety and security co-analysis of the Connected Car Vertical in the context of T5.2. We wrote a protection profile for a Safety and Security Platooning Management Module. We implemented a formal security assessment framework for Platooning scenarios that can verify whether intruders can cause harm in an automated fashion. We identified three new attacks not reported in the literature. Moreover, we constructed executable models with proposed defenses based on plausibility checks.

The third contribution covered the perspective of Task 5.3, describing the various interactions, data flows and synergies of tools, datasets and techniques developed in the context of Task 5.3. Particularly, it underlined the importance of publicly available datasets for security research. Two of such datasets are build and maintained in the context of the CAPE program, supporting the collaboration of academic and industrial research partners within and beyond the scope of SPARTA.

For the fourth contribution, in T5.4 we described evaluation process concepts and correspondence between Common Criteria Evaluation Process and safety/security engineering process (based on the V-mode) has been proposed. Furthermore, an introduction to verticals evaluability has also been described to give an idea of how verticals will be evaluated in task 5.4

The last contribution concerns the use cases “Connected and Cooperative Car Cybersecurity” and “Complex System Assessment including large software and open-source environments, targeting e-Government services”.

For the first use case, Vertical 1, we contributed with CACC platoon prototypes and evaluations of CACC platoons. We developed a collaboration between two labs, one at FTS and another at TEC, each with sets of rovers. We deployed code on the rovers that include the shared code between FTS and TEC. We carried out experiments with the rovers in both labs (FTS and TEC) thus validating our safety and security analysis (T5.2). We prepared two platoon demos with shared code between FTS and TEC labs. Experiments focusing on adaptive security for CACC have also been carried out in the CETIC lab.

For the second use case, Vertical 2, we illustrated the two prototypes for evaluating the mobile app and the IdP server of the e-Government Services vertical based on the Italian national electronic identity card (CIE). In detail, in Section 8.2, we presented the technical aspects of the selected scenarios (namely, "mobile" and "server"). Furthermore, in Section 8.3, we described the two DevSecOps pipelines composed of a selection of SPARTA security tools to evaluate the security posture of the two components and discussed the future steps to complete the integration process.

In terms of governance, we remark that the deliverable provides an example of how CAPE partners have successfully been able to collaborate towards integrated research and validation workflow. We stress that this is particularly important as evaluation and validation is the conclusion and an extremely important part of the research. It often is extremely expensive for individual researchers. The mutual exchange and joint elaboration of validation tools and processes is thus an important lessons-learned from CAPE.

# Chapter 11 List of Abbreviations

| Abbreviation | Translation |
|---|---|
| ACC | Adaptive Cruise Control |
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| APK | Android Package |
| AST | Abstract Syntax Tree |
| C-ACC | Cooperative Adaptive Cruise Control |
| CI/CD | Continuous Integration / Continuous Development |
| CIE | Italian national electronic identity card |
| CPU | Central Processing Unit |
| CVE | Common Vulnerabilities and Exposures |
| CVSS | Common Vulnerability Scoring System |
| DAST | Dynamic Application Security Testing |
| FMEA | Failure Modes and Effects Analysis |
| FTA | Fault Tree Analysis |
| FTP | File Transfer Protocol |
| GSN | Goal Structure Notation |
| GUI | Graphical User Interface |
| HARA | Hazard Analysis and Risk Assessment |
| HSM | Hardware Security Models |
| HTML | HyperText Markup Language |
| HTTP(S) | Hypertext Transfer Protocol Secure |
| IdP | Identity Provider |
| IDS | Intrusion Detection System |
| IMAP | Internet Message Access Protocol |
| ISO | International Organization for Standardization |
| KAOS | Keep All Objectives Satisfied |
| NFC | Near Field Communication |
| NVD | National Vulnerability Database |
| OWASP | Open Web Application Security Project |
| PP | Protection Profile |
| PURL | Persistent URL |
| SaaS | Software as a Service |
| SafSecPMM | Safety and Security Platooning Management Module |

| Abbreviation | Translation |
|---|---|
| SAML | Security Assertion Markup Language |
| SARIF | Static Analysis Results Interchange Format |
| SAST | Static Application Security Testing |
| SIEM | Security Information and Event Management |
| SMTP | Simple Mail Transfer Protocol |
| SoS | System of Systems |
| SR | Software Requirement |
| SSO | Single Sign On |
| SW | Software |
| SysML | Systems Modelling Language |
| TARA | Threat Analysis and Risk Assessment |
| TOE | Target Of Evaluation |
| UC | Use Case |
| UML | Unified Modelling Language |
| UR | User Requirements |
| URL | Uniform Resource Locator |
| VCS | Vehicle Communication Device |
| VCM | Vehicle Control Module |
| VM | Virtual Machine |
| XML | Extensible Markup Language |
| YAML | Yet Another Markup Language |

# Chapter 12 Bibliography

[1] SPARTA CAPE D5.1 “Assessment specifications and roadmap”, 31st January 2020 https://www.sparta.eu/assets/deliverables/SPARTA-D5.1-Assessment-specifications-and-roadmap-PU-M12.pdf

[2] SPARTA CAPE D5.2 “Demonstrator specifications”, January 2021

[3] SPARTA CAPE D5.4 “Demonstrators evaluation”, January 2022

[4] C. Talcott, V. Nigam, F. Arbab, and T. Kappe Formal specification and analysis of robust adaptive distributed cyber-physical systems. In M. Bernardo, R. D. Nicola, and J. Hillston, editors, SFM. 2016

[5] M.Clavel,F.Dura´n,S.Eker,P.Lincoln,N.Mart´ı-Oliet,J.Meseguer,and C. Talcott. All About Maude: A High-Performance Logical Framework, volume 4350 of LNCS. Springer, 2007

[6] AutoFOCUS 3 - Model-based development of embedded systems https://www.fortiss.org/en/publications/software/autofocus-3. Accessed: 08/12/2020

[7] AutoFOCUS 3 - Focus on the System https://download.fortiss.org/public/projects/af3/help/index.html. Accessed: 08/12/2020

[8] AutoFOCUS 3 - tutorial on how to deploy the generate C code into systems https://git.fortiss.org/ff1/lightrunner. Accessed: 08/12/2020

[9] https://www.cybersecurityosservatorio.it

[10] IBM Cost of Data Breach, 2019

[11] Wirpo State of Cybersecurity Report, 2018