SPARTA

D5.1

Assessment specifications and roadmap

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 / D5.1 / V1.0
Work package contributing to the deliverable	WP5
Due date	January 2020 – M12
Actual submission date	31st January, 2020

Responsible organisation	CETIC
Editor	Sébastien Dupont
Dissemination level	PU
Revision	V1.0

Abstract	This deliverable describes the scenarios and use cases addressed in the CAPE program, defines a generic assessment framework, positions the tools of the partners within this framework, addressing continuous cybersecurity certification needs, and concludes with the program roadmap.
Keywords	assessment, certification, safety, security, connected cars, e-government

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

Editor

Sébastien Dupont

Contributors (ordered according to beneficiary numbers)

Maroneze André (CEA)
Massonnet Philippe, Dupont Sébastien (CETIC)
Nigam Vivek (FTS)
Plate Henrik (SAP)
Sykosch Arnold (UBO)
Cakmak Eren (UKON)
Thanasis Sfetsos (NCSR)
Jimenez Victor(EUT)
Amparan Estibaliz, Martínez Cristina, López Angel (TEC)
Garcia-Alfaro Joaquin, Segovia Mariana, Rubio-Hernan Jose, Blanc Gregory, Debar Hervé (IMT)
Carbone Roberto, Ranise Silvio, Verderame Luca (CINI)
Spaziani Brunella Marco (CNIT)
Yautsiukhin Artsiom (CNR)
Morgagni Andrea (LEO)
Klein Jacques, Bissyande Tegawende, Samhi Jordan (UNILU)

Reviewers (ordered according to beneficiary numbers)

Jan Hajny (BUT)
Evaldas Bruze (L3CE)

Disclaimer

Executive Summary

Deliverable 5.1, CAPE Assessment Specifications and Roadmap, is the first 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 one hand, complex software systems of systems on the other hand.

The main highlights of the CAPE program at almost one year of execution are threefold. First, the deliverable describes a universal V-model of the software development lifecycle, that we have extended to meet our needs, and that enables a common description of all our tools despite their diversity in purpose and scope. Second, it describes a set of new features for software assessment, that will constitute advances to the state of the art with respect to software security and safety. Third, it specifies new controls that can be embedded into software, with the specific goal of meeting the security-safety co-design objective of the CAPE program.

The deliverable describes two use cases associated with these environments, the connected vehicle for security and safety co-design, and the e-government for complex systems of systems. It briefly describes the assessment demonstration scenarios foreseen in the verticals context: small-scale vehicle infrastructures to simulate behavior of real vehicles in the connected vehicles vertical, and mobile/desktop authentications solutions in the e-government vertical. It also provides a description of a generic assessment framework and positions the tools provided or being developed by the partners within this assessment framework.

The role of the framework is to describe in which phase of the security engineering process each of the assessment tools can be used. The framework includes safety engineering and cybersecurity certification evaluation processes in order to explain how each of the assessment tools could also be useful in these processes. The framework approach provides a loose coupling approach between tools that have been designed and developed independently: simulation-based fault injection, formal verification, software verification, penetration testing, security analysis, etc. Original assessment methods will be produced and demonstrated: for example vulnerability assessment based on abstract code representation to increase the level of assurance in e-government authentication services, and model-driven/simulation-based fault injection for early dependability evaluation of safety-critical systems such as connected cars. The deliverable describes plans for making the assessment tools available in a continuous integration and DevSecOps approach for the verticals in the next years of the project.

The deliverable concludes with a roadmap for the next years of the research program. Roadmaps are given for each of the tasks, detailing objectives, activities and expected results that will be reported in the next deliverables D5.2, D5.3 and D5.4.

Table of Content

List of Figures

List of Tables

Chapter 1 Introduction

The CAPE program, listed in the description of action as workpackage 5 (WP5), 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 addresses the issue of assessing cybersecurity performance of our environments. It targets two specific environments, cybersecurity and safety convergence on one hand, complex systems of systems on the other hand.

Cybersecurity and safety convergence stems from the fact that industrial systems, typically cyber-physical systems, are increasingly controlled by programs. For example, cars are probably one of the most computerized objects available today, with over 100 onboard controllers and computers, and more code inside than an airplane. While we have addressed safety of cyber-physical systems for some time now, there is a need to provide methods and tools for including cybersecurity properties in safety reasoning, and to jointly specify cybersecurity and safety goals, to ensure that one would not disable the other.

Complex software systems are increasingly driven by the DevOps paradigm, where development and runtime are intimately mixed to put in production advanced versions of software. This is reinforced by the fact that software vendors rely on large numbers of libraries, which also need to be risk-assessed and incorporated into the cybersecurity assessment process.

The CAPE program addresses these two aspects through tasks 5.2 and 5.3 respectively, each of these tasks focusing also on a use case. Deliverable 5.1, *Assessment specifications and roadmap*, is the first deliverable in the CAPE program. Its role is thus to set the scene of the program activities for the duration of the project, leading to the definition, development and demonstration of assessment tools linked to the two environments described before.

The assessment tools being developed or extended in the CAPE research program are presented in the form of a cybersecurity assessment framework. The role of the framework is to describe in which phase of the security engineering process each of the assessment tools can be used. The framework also takes into account safety engineering and cybersecurity certification evaluation processes in order to explain how each of the assessment tools could also be useful in these processes.

D5.1 is organized as follows. Chapter 2 describes these use cases, the connected vehicle and platooning for cybersecurity and safety, e-government for complex systems of systems. Chapter 3 provides high-level assessment specifications, whose goal is to describe the assessment lifecycle as we understand it, so that it fits the needs of the use cases. Chapter 3 also describes the tools made available or developed by the partners in the context of CAPE. Chapter 4 concludes with our development roadmap.

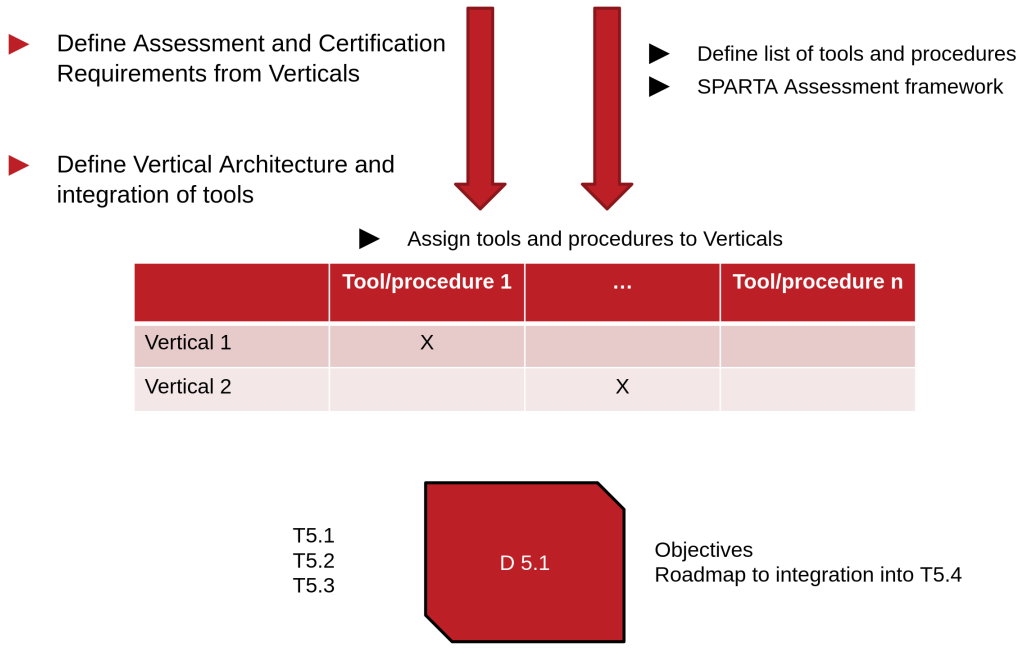


Figure 1: Process for construction of D5.1

Chapter 2 Assessment and Certification Requirements for the Verticals

This section provides an overview of assessment and certification requirements for the two verticals addressed by the CAPE program. For each vertical, the objectives of the case studies will be described and their relevance to assessment and certification will be explained. Architectures for the various case studies are then presented to demonstrate assessment methods, tools, standards and certifications.

- the **Connected and Cooperative Cars Vertical** (section 2.1) aims at advancing the cyber-security of connected vehicles in the context of a vehicle platooning scenario. The vertical will produce 3 demonstrators that will show how continuous assessment can improve security and safety [1] using tools and techniques built in the CAPE program: simulation-based fault injection, Security/Safety by design [2] through formal verification, software verification & validation and penetration testing.
- the **e-Government Services vertical** (section 2.2) has as goal to improve the cyber-security of authentication solutions based on the Italian national identity card. Through 2 case studies (desktop RF and mobile NFC), several assessment methods will be demonstrated: protocol security verification, software verification methods, vulnerability and risk assessment.

Note that 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 (cf. Section 3.3.2.1) as well as the detection of so-called supply chain attacks (cf. Section 3.3.2.2). 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.

2.1 Vertical 1: Demonstration of converging tools for assessing Connected and Cooperative Car Cybersecurity (CCCC) in the context of Euro NCAP

The past years have witnessed a technological revolution interconnecting systems and people. This revolution is leading to new exciting services and business models. The automotive industry is also profiting from this revolution, as in the near future, vehicles will reach high levels of autonomy, enabled by many technologies and in particular, by the increased integration between vehicles and between vehicle and the available infrastructure.

However, the increased interconnectivity and increased level of autonomy increase both the attack surface of these systems and also the degree of damage that intruders can cause. Indeed, cyber-attacks can exploit vulnerabilities in the available communication channels to cause catastrophic events, i.e., great human and material loss. For example, it has been shown that safety mechanisms can be deactivated remotely by intruders [5, 8], thus placing its passengers in danger.

[1] In this deliverable, concepts such as security, risk, safety, threats, ... are used according to the ENISA overview of cybersecurity and related terminology https://www.enisa.europa.eu/publications/enisa-position-papers-and-opinions/enisa-overview-of-cybersecurity-and-related-terminology, they will be defined in the following sections of the deliverable as needed

[2] Note that work on the related concepts "Privacy by Design" and "Privacy by Default" is achieved in the SPARTA work package 2

This vertical has as goal to advance the cyber-security of connected vehicles. In the next sections, we will discuss the specific case study we will investigate, namely platooning, the available infrastructure for demonstrating the effectiveness of our goal, and related assessment and certification requirements.

2.1.1 Case Study Objectives, Description and Relevance

Vehicle Platooning is a vehicle mode where vehicles, normally trucks, travel in a sequence formation, as illustrated by Figure 2. The platoon is led by a vehicle, called *leader*, and the following vehicles are called *followers*.

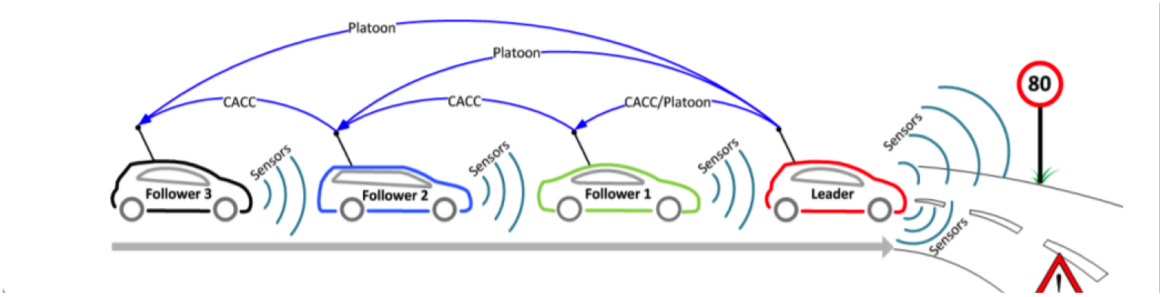


Figure 2: Illustration of a platooning formation.

The vehicles try to keep a minimum gap between the vehicles to reduce fuel consumption by profiting from the wind shadow generated by the following vehicle. To keep such a short gap and still be safe, vehicles possess distance sensors and rely on communication channels between the vehicles. For example, whenever a vehicle has to activate its emergency brakes, the whole platoon is informed, thus decreasing the reaction time and avoiding accidents.

Besides the improved fuel consumption, platooning also improves the traffic safety, as the leader car is connected to the infrastructure that informs the leader about, for example, possible hazards in the road, so that the leader can react adequately maintaining the platoon in safety.

From a security perspective, one needs to answer the following question:

How can one continuously guarantee that cyber-attacks cannot exploit vulnerabilities in, for example, the communication channels and cause accidents?

This research question can be broken down into the following two sub-objectives:

- **Objective 1**: To demonstrate in the platooning case study safety and security co-engineering, including techniques for co-analysis, co-verification, and assessment.
- **Objective 2**: To demonstrate how safety and security standards relevant for connected cars can be integrated and how they can impact the assessment process.

In order to achieve these objectives, we will develop and apply techniques from the following domains:

- **Cyber-Physical Systems:** Connected cars contain embedded software (cyber) that interacts with the physical world. For example, the software embedded in the platoon vehicles takes decision on accelerating or not depending on the information gathered by sensors and the information sent by the other vehicles.
- **Automotive:** As the vertical involves vehicles, it is also subject to the existing certification and assessment standards.
- **Safety and Security:** As described above, cyber-attacks can cause catastrophic events, e.g., vehicle accidents. This means that there is a clear impact of security in safety. Methods as safe/secure by design will be deployed to carry out co-analysis between safety and security.

- **Network Communication:** The network communication between the vehicles has to have a high level of trust and thus security, so that vehicles can base their decisions on the information transmitted between the vehicles.
- **Model-Based Engineering:** The automotive industry widely applies Model-Based Engineering approaches for the development of vehicles. Instead of using document-based development, Model-Based Engineering approaches builds models that can be reasoned over, such as for co-analysis of safety and security [71, 81].

The vehicles in the platoon have the following **main functions**:

- **Platoon formation**: A vehicle may join a platoon by using a vehicle merge maneuver, or a vehicle that is already member of a platoon may leave its platoon by using a vehicle split maneuver. A vehicle performing a merge maneuver may join the platoon at the back of the platoon, or can also use a more complicated maneuver and join in the middle of the platoon, in which case, the platoon vehicles form a larger gap so that the merging vehicle can join the platoon. These functions require that the merging/splitting vehicle and the leader of the platoon coordinate by sending data, such as their speed, location, and number of vehicles in the platoon.
- **Lane Keeping**: The vehicles when in a platoon (or in the platoon mode) keep within the road lane in an automated fashion. This means that the vehicles can identify the lane in which they are and they can follow the lane even when the road has curves.
- **Sensing functions**: Vehicles contain a number of sensors that provide information about the vehicle and its environment. For example, LIDAR can inform the distance to the vehicle in front. GPS signals inform the vehicle's localization. Cameras can be used together with AI elements to detect the types of objects in the road.
- **Vehicle to Vehicle (V2V) communication**: Vehicles can communicate within each other. This is necessary, for example, to negotiate when a vehicle wants to join a platoon, and coordinate among the vehicles so that an accepted vehicle joins the platoon, and to inform vehicles whenever there is an emergency brake maneuver.
- **Infra-structure to Vehicle (I2V) communication**: Vehicles can also communicate with the available infrastructure. For example, the infrastructure can inform vehicles whenever there is a hazard ahead. Similarly, a vehicle can inform the infrastructure about the road conditions.
- **Adaptive Cruise Control (ACC) or Cooperative Adaptive Cruise Control (CACC)**: To maintain the gap to the next vehicle, a vehicle has to control its speed in an automated fashion. The gap has to be kept within a safe distance, even vehicles activate their emergency brake. It may do so by relying in its sensors, e.g., distance sensors, without relying on the information communicated by the other vehicles, called Adaptive Cruise Control, or they can use the information sent by the other vehicles in the platoon, called Cooperative Awareness Basic Service. Vehicles that use CACC can be informed, for example, whenever a vehicle ahead in the platoon has activate the emergency brake and take preventing actions. However, this communication can also, in principle, be exploited by intruders.

Vertical Scenario Conditions For the vertical, we are considering a scenario where the platoon(s) are on the Highway in a dedicated lane for platoons only. Moreover, there are no intersections nor joint points. Initially, we are going to assess the security of when the platoon is formed and uses CACC so that vehicles maintain a safe distance between each other. Then we will also consider more complicated maneouvers, such as when a new vehicle wants to merge with an existing platoon or when a vehicle wants to split from the platoon that his part of.

2.1.2 Architecture and Technology of the Case Study

The CCCC vertical will have some demonstrators from different companies where several of the tools and methodologies developed in the CAPE workpackage will be applied. Demonstrators will be composed of a small-scale vehicle infrastructure simulating the behaviour of real vehicles.

2.1.2.1 Demo 1: Tecnalia Case Study

Tecnalia will provide an infrastructure of three Veloxcar vehicles as the one illustrated by Figure 3. These Veloxcars will be part of the demonstration of safety and security co-engineering, including techniques for co-analysis, co-verification, and assessment. The size of each vehicle is 1:8.

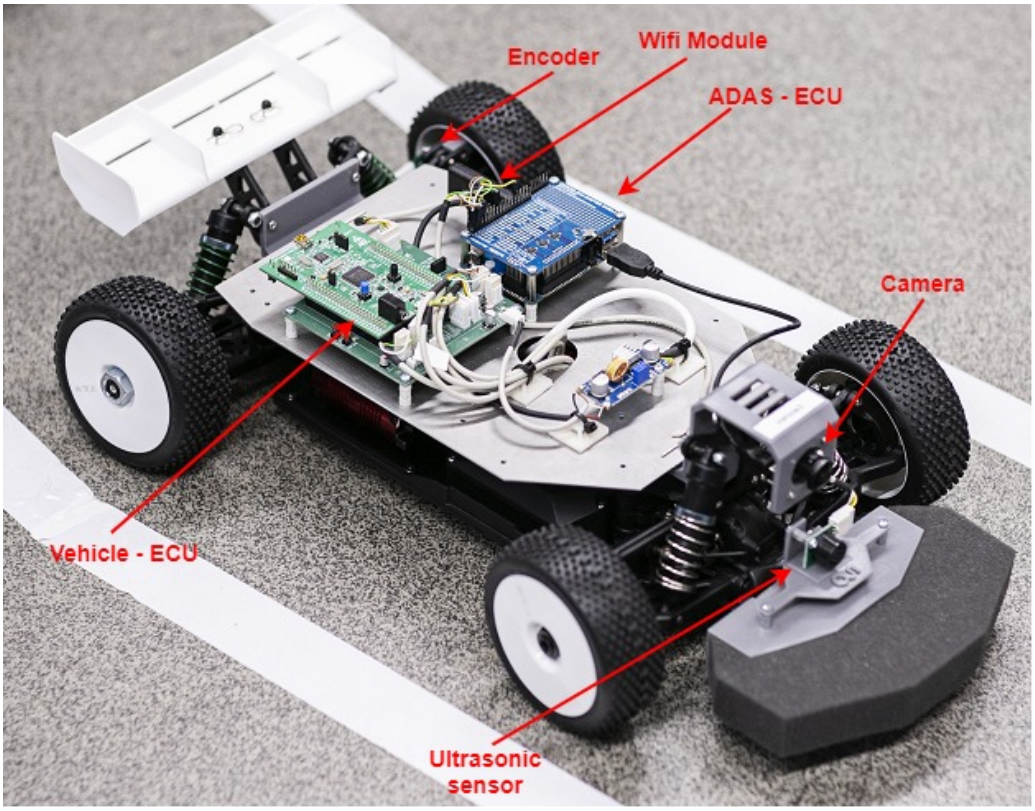


Figure 3: Veloxcar vehicle.

The Figure 4 shows an overview of the top-level system architecture of a Veloxcar vehicle. Each Veloxcar has two Engine Control Unit (ECU)s. The first ECU, called *Vehicle-ECU*, contains the software to operate the basic communication and motion-systems needed to run the vehicle actuators, whereas the second ECU, called *Advanced Driver-Assistance System (ADAS)-ECU*, contains the necessary software to operate all systems related to the autonomous-driving-functions, such as Lane Keeping or Adaptive Cruise Control. This ECU has the Robot Operating System (ROS) as Middleware running on a Linux-Distribution.

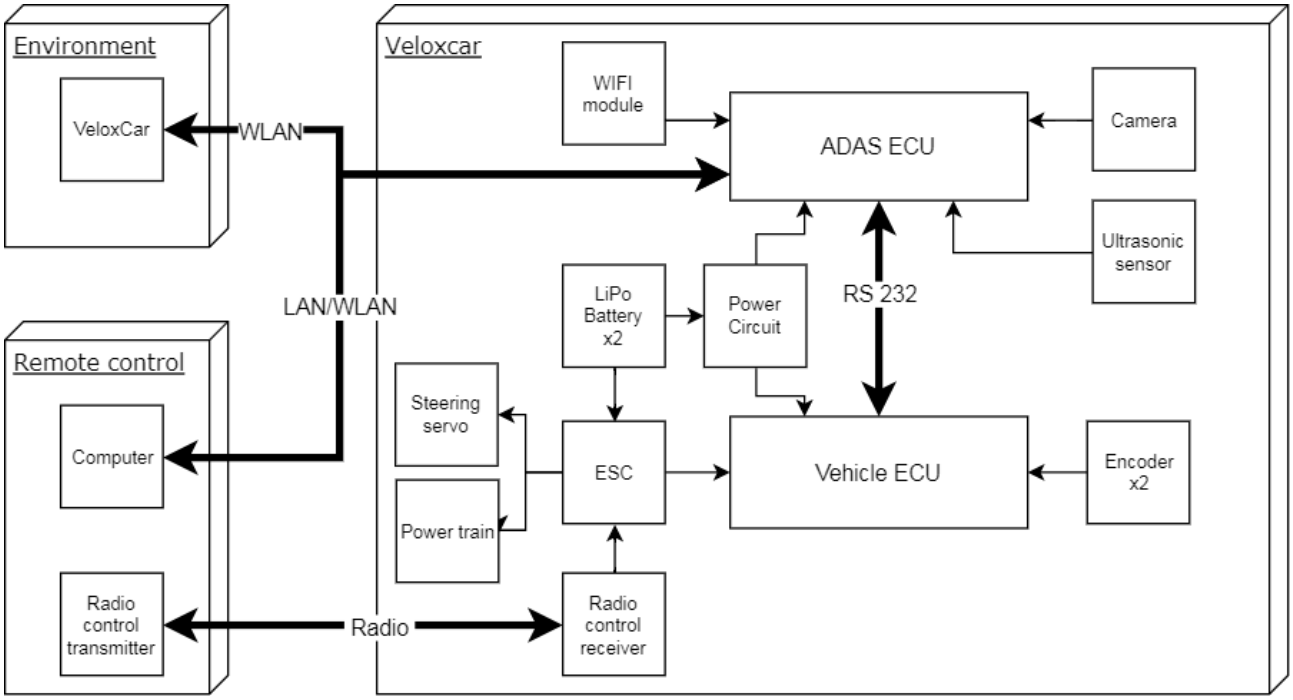


Figure 4: Veloxcar architecture.

Vehicle-ECU is a STM32F4 Discovery board that manages all motor control (longitudinal and lateral) and contains vehicle model odometry calculation from the encoder sensor data.

- **Incremental Encoder**: The vehicle has two encoders installed in the back wheels to measure the current wheel speed.
- **Steering servomotor**: Servomotor that performs the angle of the steering.
- **DC Motor**: Brushless Direct Current (DC) motor which deploys the speed on the vehicle.

ADAS-ECU is an Odroid XU4 board which can run various Linux distributions, in our case Ubuntu Mate 16.04. By implementing the eMMC 5.0, USB 3.0 and Ethernet interfaces, the ODROID-XU4 has a high data transfer speed, which is a feature that is increasingly required to support the processing of the autonomous-driving-functions. To perform these functions, some inputs coming from the following sensors are needed.

- **Camera**: A camera is installed in front of the vehicle to capture a lane on the road. The camera is a DFM 22BUC03-ML USB 2.0 color board camera with 1/3 Micron CMOS Sensor, 0,4MP (744x480) resolution and up to 76 fps.
- **Ultrasonic sensor**: The ultrasonic sensor installed in the front of the vehicule is a SRF02 sensor with a range of 20cm to 6m. a frecuency of 40KHz and both a i2c and serial interface. This sensor emits an ultrasonic wave and receives the wave reflected back from any object in the wave path. The ultrasonic sensor returns the distance from the vehicle to the object, measuring the time between the emission and the reception.

The Veloxcar has several communication systems implemented. Firstly, to reach the communication between the two ECUs, a RS232 serial communication channel is used. Secondly, to set parameters and/or control the vehicle from a computer, Local Area Network (LAN) (Ethernet) or Wireless Local Area Network (WLAN)(WiFi) are used. Finally, wireless communication between vehicles is achieved via WiFi. All the communication between the vehicles is not encrypted.

In the CCCC vertical, the intelligence of the vehicles will be improved to be used in a **Cooperative Adaptive Cruise Control (CACC)/Platooning scenario**, which is an improvement of the Adaptive Cruise Control (ACC) advanced driver assistance system. In this scenario one of the vehicles will be the *leader*, and using the data coming from the odometry, camera and ultrasonic sensors will be kept circulating within a circuit that has been traced with white lines. The leader will send its speed via

WIFI to other two vehicles (*CACC followers*) to adopt it. The followers must also maintain a safety distance between them.

Along the demostration, the leader will be modifying its velocity (accelerating or decelerating) along the route and the followers will be adapting their velocity in order to maintain a safe gap to the vehicle in the front.

Several hazardous situations can arise that affect the safety of the system (cf. Section 3.2.1.1.1). Besides that, several threats on the communication channels could also affect that safety (cf. Section 2.1.3.5). Both safety hazards and security threats could cause collisions between the platooning members, affecting the safety of the vehicules and the passengers integrity. Therefore, some safety and security mechanisms should be added to mitigate these risks (cf. Section 2.1.3.6).

2.1.2.2 Demo 2: Fortiss Case Study

Fortiss will make available its rovers to demonstrate safety and security co-engineering techniques, developed in Task 5.2, and for the demonstration of the software security verification tools, developed in Task 5.3. Figure 5 depicts the types of rovers available at Fortiss. As the rovers were mounted within Fortiss, they can be customized with different set of sensors. For example, some rovers include LIDAR sensors that have been used to implement automated parking functions.

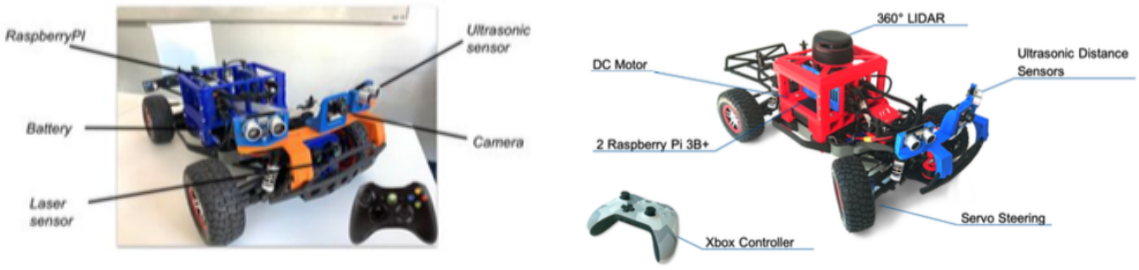


Figure 5: Illustration of Fortiss rovers.

The basic architecture of the Fortiss rovers is composed by the following components:

- Raspberry PI, containing the embedded software with the logic of how the rovers shall behave.
- pi camera used as a sensor for detecting, for example, which vehicle is ahead. This functionality is provided by using the open source OpenCV software https://opencv.org/.
- FOCBOX (VESC) motor controller together with its embedded proprietary software, that controls the motor actuators.
- Tinkerforge Sensor system, containing a basic set of sensors.

The software embedded in the rovers is partly developed using a Model-Based Engineering development process, using the tool AutoFOCUS3 [2], using a Model-Based Engineering approach.

Figure 6 depicts a screenshot of part of the AutoFOCUS3 model. AutoFOCUS3 generates C code from the model, which is then embedded in the available Rasberry Pis. Moreover, some C libraries have been implemented by Fortiss for integrating the code generated by AutoFOCUS3 and the remaining libraries embedded in the rover.

Currently, there has not been any emphasis on the security issues, but only on demonstrating that it is possible to program Platoon solutions using Model-Based Engineering approaches. For example, all the communication between vehicles is not encrypted.

Therefore, we strongly believe that the Fortiss demonstrator could profit from the assessment tooling available/developed by CAPE.

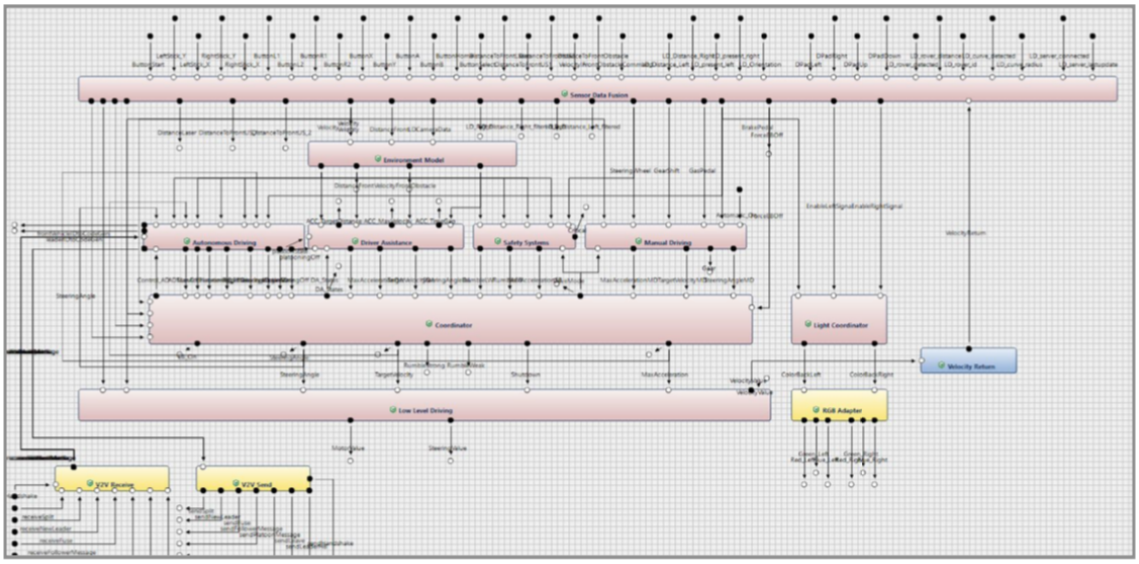
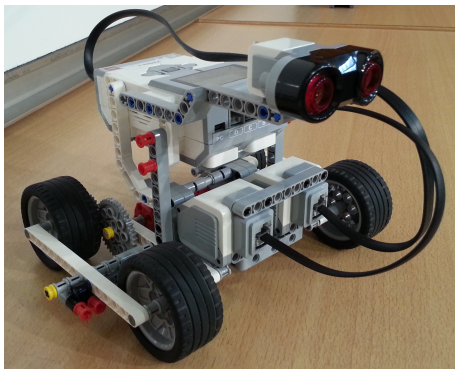
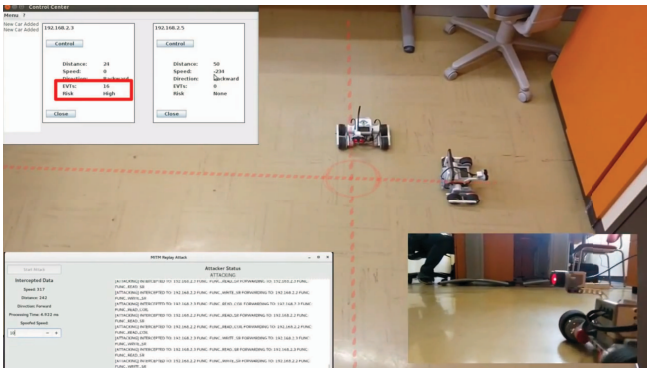


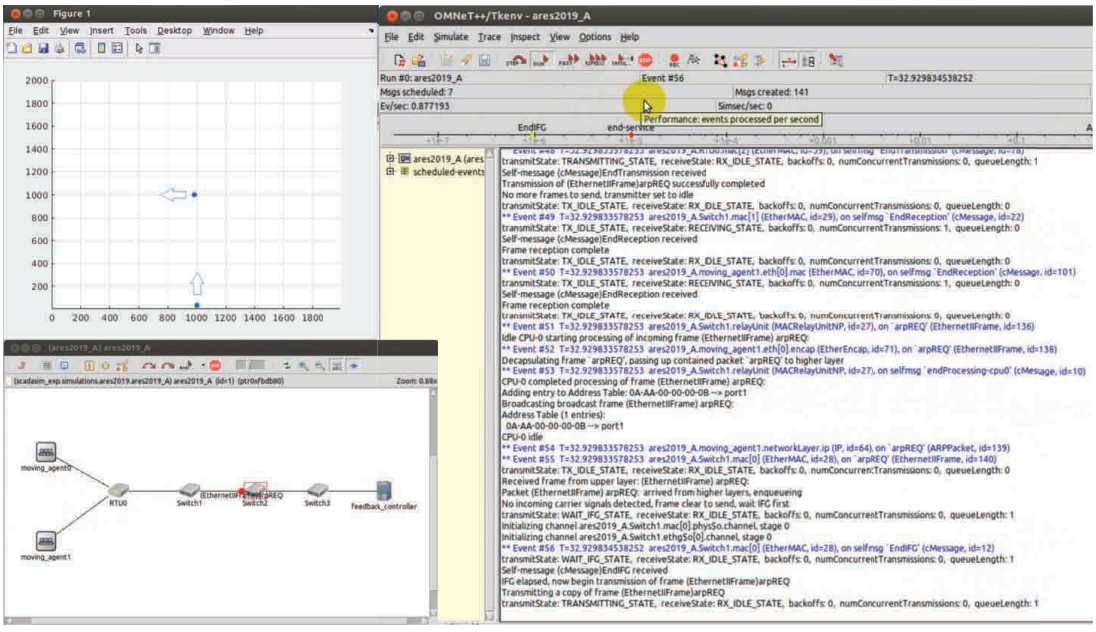
Figure 6: Illustration of the AutoFOCUS3 specification of the software embedded in the fortiss rover.



(a) EV3 Rover agent.



(b) Generation of SCADA traffic.



(c) OMNeT++ co-simulation, using the generated SCADA traffic.

Figure 7: IMT SCADA Testbed (cf. http://j.mp/omnetcps and http://j.mp/legoscada for live demonstration videocaptures and source code). (a,b) Generation of SCADA-driven CPS data (c) OMNeT++ co-simulation.

2.1.2.3 Demo 3: IMT Case Study

The testbed consists of *Lego Mindstorms* EV3 Rover agents [98], complemented by Raspberry Pi [74] units implementing the feedback controllers in charge of leading the sensors (e.g., distance sensors) and actuators (e.g., dynamic speed accelerators) of the EV3 Rover agents. Some pictures of the testbed are shown in Figures 7a and 7b. SCADA traffic is generated from the testbed (EV3 Rover agents and controllers), based on standard Modbus command and control specifications [77]. The testbed implements a kinematics scenario, in which two or more EV3 Rover agents perform a deterministic (cyclic) motion (e.g., backward and forward motion over a bounded square area).

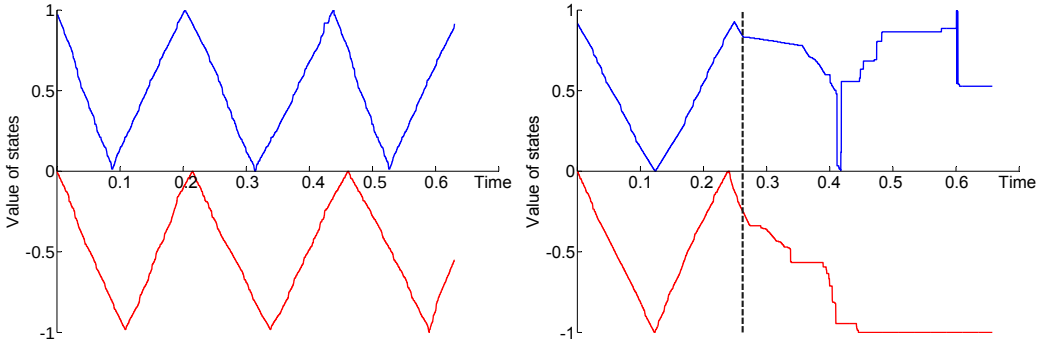
Figure 7c depicts the numeric co-simulation complementing the kinematics scenario, using the collected SCADA traffic to train a cyber-physical programmable simulator using OMNeT++ [112, 118] and some related TCP/IP and Modbus simulation libraries [97, 113]. Each EV3 Rover agent (cf. Figure 7a) have a distance sensor in the frontal part, to measure the relative distance to the boundaries of a unit square area. The distance is transmitted to the feedback controllers, via SCADA commands. The feedback controllers compute the relative velocity of each agent, and the Euclidean distance between, e.g., two agents, in order to guarantee spatial collision-free motion. The goal of the testbed is to experiment an adversarial use case, in which a rogue device perpetrates a cyber-physical attack that affects the control processing of the EV3 Rover agents. If the attack is successful, the agents move to undesirable states, resulting in the physical collision depicted by the dotted red lines in Figure 7b. Figures 8(a–b) show the kinematics during the nominal case (i.e., absence of attacks, left-side); and during the attack (i.e., the moment at which the adversary takes control over the system, right-side). Time is normalized between 0.0 and 1.0, representing the temporal percentage of multiple experimental runs. We can appreciate how the system moves to unstable states, disrupted by the adversary. Some live demonstration videos showing the spatial collision that cause the disruption represented in Figure 8 are available at http://j.mp/legoscada.

During the OMNeT++ co-simulation, we analyze the system behavior in the normal operation mode, under attack and using the attack attenuation approach presented in [106]. In the testbed, the two EV3 Rover agents follow a trajectory of up to two meters. The feedback controllers coordinate the movement of the agents, by sending the relative velocity to the agents, and receiving back the distance of the agents to the spatial boundaries. The feedback controllers send a series of SCADA commands, through a network of traffic programmable forwarders (e.g., software defined switches). The physical process controlling the agents, i.e., their distance sensors and their dynamic speed accelerators, are controlled by the differential equations and automata defined in [100]. The adversary starts the cyber-physical attack by either tampering the controller with fake sensor readings or modifying the control commands sent from the controller. With the OMNeT++ co-simulation, we evaluate the attenuation of a bias injection attack [111], i.e., by forging tampered control commands from the controller to the plant. The focus of the co-simulation is only the physical part of the cyber-physical attack, using the network to damage the system. In other words, it assumes adversaries that already found their way to hack and gain unauthorized remote access to the system, e.g., using cyberattacks exploiting weaknesses associated to the cyber part of the system.

Each co-simulation evaluates multiple Monte Carlo different runs (i.e., hundred runs per evaluation test). Simulations consider as well potential sensing errors (e.g., up to 1% distance sensing errors) w.r.t. the real distance values. They also consider network delays, e.g., based on classical distribution delays reported in [39]. Figure 9(a) shows the results obtained for the nominal case (i.e., absence of attack), considering the aforementioned simulation assumptions. The plots depict the average Euclidean distance, with 95% confidence intervals, between the agents, and in function of time. The horizontal axis of the plots in Figures 9(a–d) provides a normalized time between 0.0 and 1.0, representing the temporal percentage prior concluding the simulation runs. The vertical axis of the plots in Figures 9(a–d) provides the Euclidean distance between the two agents, from 0 to 1400 cm. Some further evaluation details are discussed below.

During the perpetration of the attacks, adversaries performs a bias injection of cyber-physical data [111]. Adversaries use the network to modify the exchanged packets between the feedback controllers and the agents. We assume that the adversaries are constantly recording and learn-

(a) Temporal representation of the kinematics scenario, associated to two EV3 Rover agents (left-side, nominal mode; right-side, kinematics during the attack).



(b) Winding graph of a complete kinematics cycle (left-side, nominal case; right-side, attack case.)

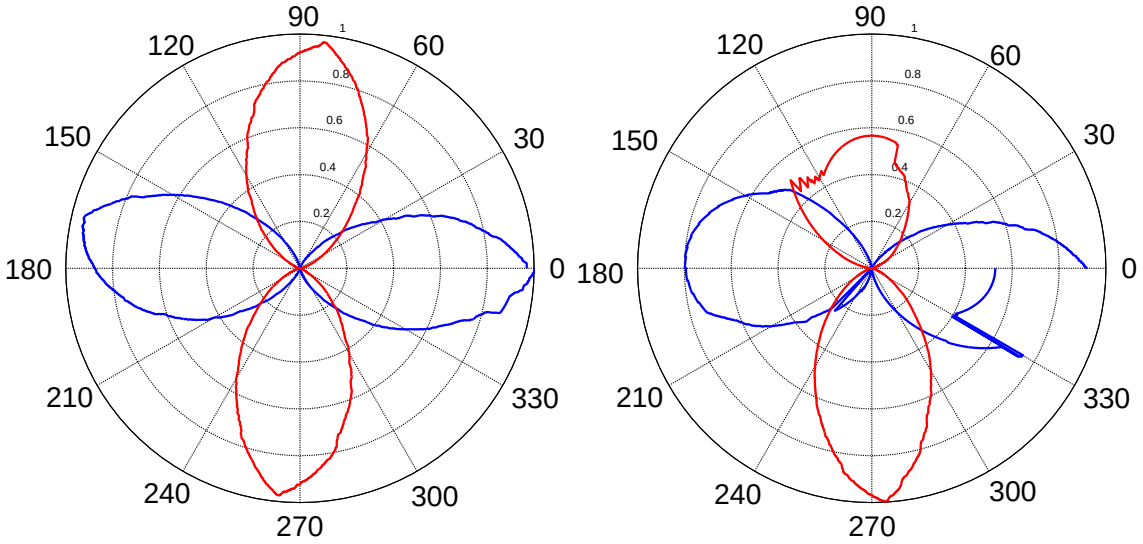


Figure 8: Lego EV3 Rover testbed results (kinematics during nominal and attack modes).

ing the system kinematics by simply accessing the exchanged outputs associated to the commands between sensor, controllers and actuators. Adversaries perform an initial learning phase, to eavesdrop data and infer the system kinematics, i.e., the same ones used by the feedback controller to guarantee the stability of the system, shown as nominal case in Figure 9(a).

Let u_k be a feedback controller command sent to the actuator of one of the agents at time k. Let u_k^{act} be the command received by the actuator at time k, where $0 \leq k \leq T_s$ and T_s be the full duration of each simulation run. The attack interval T_a is limited to the simulation time T_s, as summarized next:

$$u_k^{act} = \begin{cases} u_k & \text{if } k \notin T_a \\ u'_k & \text{if } k \in T_a \end{cases}$$

During the evaluation, we compare two type of adversaries according to the bias injected in the payload of the packets, i.e, according to the difference between the value u'_k injected by the attacker and the real value u_k sent by the controller. We define two adversary models: (1) *noisy adversary* and (2) *stealthy adversary*. The noisy adversary injects in u'_k a bigger difference with respect to the correct command u_k sent by the feedback controller compared to the stealthy adversary. In consequence, a noisy adversary will make the system move faster from its nominal state. Figure 9(b) shows the results obtained for the two adversary models. The feedback controller loses its control over the system, while the adversary forces the spatial collision of the two EV3 Rover agents.

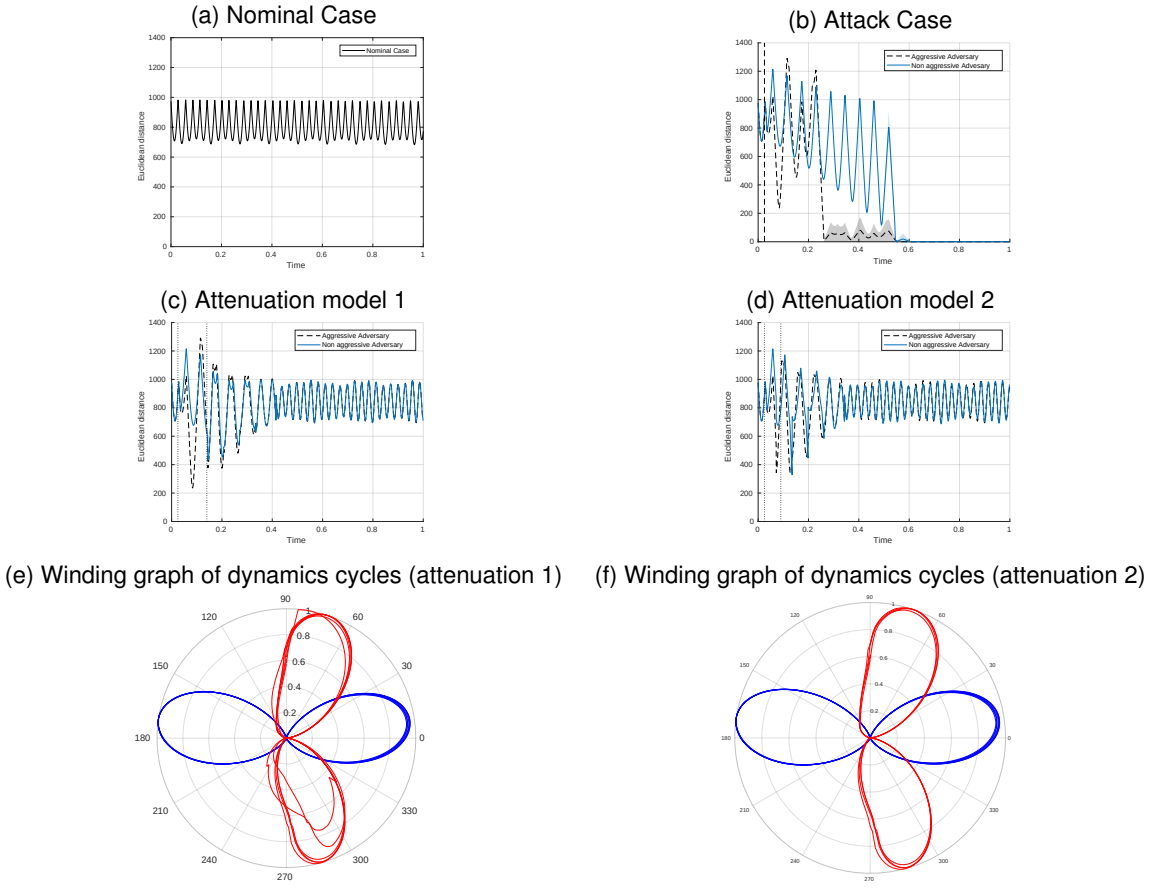


Figure 9: OMNeT++ results. (a–b) Euclidean distance (with 95% confidence intervals), nominal and attack simulations. (c–d) Euclidean distance, attenuation of two different remediation starting time models. (e–f) Winding graphs, same attenuation models.

During the attenuation process, the system reacts using reflective programmable networking [106]. Some reflective controllers change the control policies of the system (e.g., conduct a given switch w.r.t. to the programmable controller domains). This reflective change disrupts the adversary (its knowledge about the system) and attenuates the attack. For each of the two defined adversaries, we simulate distinct scenarios using different values for the time the solution starts working. This is a parameter of the simulation that depends mainly on the time required to activate the attack responses, i.e., the time required to set up and coordinate all the components working in the attenuation approach. Figures 9(c)–(d) show how the approach guarantees the controllability property. The first vertical dotted line shows the moment when the attack starts and the second vertical dotted line shows the moment when the technique starts. It is possible to appreciate that the attacker introduces a perturbation in the system. As a consequence, the Euclidean distance between the two EV3 Rover agents starts oscillating out w.r.t. their expected behavior (i.e., w.r.t. Figure 9(a)).

When the attack is detected, new reflective agents will start re-configuring and moving the system control, to attenuate and restore the nominal behavior of the system. Figures 9(e–f) show the winding graph of the EV3 Rover agents under this new approach. The attacked device corresponds to the vertically oriented ellipses. It is possible to observe some perturbations, due to the modifications introduced by the reflective agents when thwarting the adversary actions and recover the stability of the process. As a result, the spatial collision between the two agents is avoided and the system keeps working. Notice that the technique takes control of the physical environment in order to conduct the physical environment from an unstable behavior generated by the attack to a stable and safety behavior, converging to the normal behavior of the physical environment. Figures 9(c–d) show that approach neutralizes the effects of the attack right after a short period of instability. The approach does not eliminate the adversary. However, it contains the effects and reorients the system to the nominal case.

2.1.3 Assessment and Certification Requirements

Even if several safety and security engineering tools and methodologies share common concepts, safety and security are different in nature. Safety stands for freedom from unacceptable risk of harm and functional safety depends on a system or equipment operating correctly in response to its inputs. On the contrary, security is concerned with the protection of assets from threats, where threats are categorised as “the potential for abuse of protected assets”. Fail-safe behaviour is important from a safety perspective while this feature confronts with the security requirement of availability. In the way towards autonomous systems, fault tolerance and availability of functions will be relevant since the solution of switching a system when a failure is detected will simply not be possible anymore. The more fault-tolerant a system is, the more the attack surface increases. This means that whereas redundancy can be good to achieve the required level of reliability, this does not apply to security.

There are many benefits on merging the computing and communication process of Cyber-physical system (CPS)s with the physical process. But because of this, CPSs get a critical problem due to security, caused by the difficulty on protecting the information. This is due to the nature of the CPS architecture design where data, communication, processing and communication channels are combined.

Security in CPSs like vehicles is relatively new and can be defined as the ability to secure the data and operational capabilities of the system from unauthorized access. Confidentiality, integrity, availability and authenticity are the basic properties of any system’s security:

- **Confidentiality** is satisfied if data or objects are not read by an unauthorized party;
- **Integrity** is satisfied if data or objects are not changed (written) or generated by an unauthorized party;
- **Availability** is satisfied if data, objects or services are available;
- **Authenticity** is satisfied if an author of data or an object is who it claims to be.

On the automotive domain, security has become a pressing issue due to the fact that modern vehicles can be attacked from a variety of interfaces, including direct or indirect physical access and, short- or long-range wireless channels. A hacker could exploit vulnerabilities in the vehicle’s electronic systems to generate incidents and even take control of them, which could affect the safety of their occupants, other vehicles on the road or the transport infrastructure itself in the event of a collision. For instance, a hacker could get access to some of the vehicle ECUs and take control of some critical components like brakes or engines, that is why security and safety are closely joined.

All safety-critical systems are cybersecurity-critical since a cyberattack could lead to potential safety losses. Safety is satisfied if an unwanted interference with the on-board vehicle system or communication system has not any impact on the safe operation of the vehicles. A vehicle will never be 100% secure, nevertheless, following a well-defined development process will reduce the likelihood of a successful attack, which can also be related to a reduction of potential hazards.

For the CCCC vertical, where safety and security are closely joined, the connection of the cybersecurity process with the safety process must be applied, keeping both activities separate but performed in conjunction. Therefore, to define the security requirements a Threat Analysis and Risk Assessment (TARA) must be performed but also having into account the Hazard Analysis and Risk Assessment (HARA). It is recommended to elaborate a short assessment of potential safety related threats to determinate if there are any high potential hazards.

The methodology used for inferring security requirements is based on the EVITA project approach [43], and involves the following stages:

- Description of the system under investigation and its environment.
- Description of relevant use cases.
- Identification of the assets to be protected within the described use cases.
- Analysis of the hazards that may affect the safety of the use cases.
- Analysis of the threats that can also affect the safety.

- Identification of the security functional requirements.

2.1.3.1 System under Investigation and its Environment

The system under investigation is an example of a collaborative safety-critical system: a Connected and Cooperative car/Platooning system of several vehicles. The main purpose is to evaluate a joint co-engineering process concerning functional safety and cybersecurity.

A complete description of the system and its architecture has been included in Section 2.1.1 and Section 2.1.2, respectively.

Several **hazards** can arise that affect the safety of the system. For instance,

- the leader changes its speed and the followers do not adapt themselves to the new situation;
- the leader changes its speed and, due to a loss of communication, the followers are not aware of the change;
- the leader changes its speed, but the value is corrupted during the communication to the followers.

Besides that, several **threats** can also affect the safety of the system. For instance,

- a hacker, using a Man in the Middle (MiM) cyber-attack on the WiFi communication, could alter the speed transmitted by the leader before being received by the followers as illustrated by Figure 10,
- a hacker, using a Denial of Service (DoS) cyber-attack to the leader's WiFi Access Point, could prevent the leader from transmitting its speed to the followers.

All these safety hazards and security threats could cause collisions between the platooning members, affecting the safety of the vehicules and the passengers integrity. Some possible safety mechanisms to mitigate these risks would be, for instance, adding a proximity sensor to each vehicle which ensures a minimum safety distance between the platooning members, or comparing the current speed received with the previous values in order to detect an abnormal input, for example, receiving a 80 kph speed when the previous one was 20 kph.

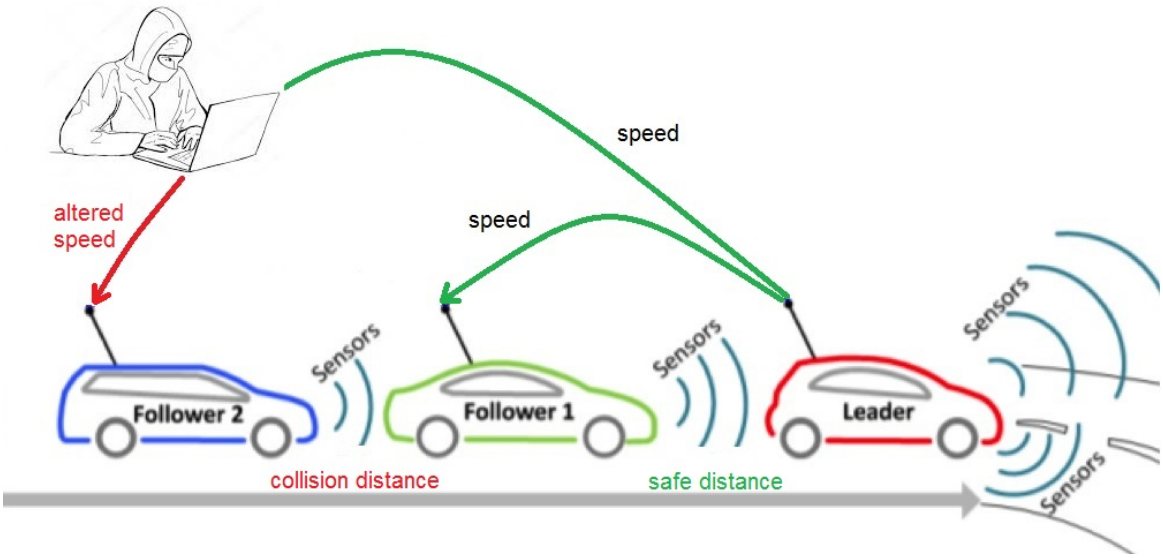


Figure 10: Man in the Middle attack to platooning increasing a follower's speed.

2.1.3.2 Use Cases

The use cases described in this section are intended to identify a range of specific platooning functions that could have possible security implications.

Identifier UC1

Name A follower does not adopt the leader's speed due to some failure in the communication channel.

Description To maintain a safety distance to the next vehicle, each follower must control its speed based on the distance to obstacles in the path in an automated fashion. A failure in the communication channel between the leader and a follower would cause collisions between the platooning members.

Actors

- leader
- follower n.1
- follower n.2

Basic flow The leader detects an obstacle and reduces its speed. The new speed should be transmitted to the followers, but a WiFi communication failure (no message or corrupted message) prevents the reception of the signal by the follower(s). A safety mechanism must be added in the scenario to avoid the collision between the platooning members.

Identifier UC2

Name A follower does not adopt the leader's speed due to an attack in the communication channel.

Description To maintain a safety distance to the next vehicle, each vehicle has to control its speed in an automated fashion. A hacker's attack in the communication between the leader and the follower(s) would cause collisions between the platooning members. A safety mechanism must be added in the scenario to avoid the collision between the platooning members.

Actors

- leader
- follower n.1
- follower n.2
- hacker

Basic flow The leader detects an obstacle and reduces its speed. The new speed should be transmitted to the followers, but a hacker, using a cyber-attack on the WiFi communication, increases the velocity value transmitted by the leader before being received by the followers, causing collisions between the platooning members.

Identifier UC3

Name Checking the speed value that has been received from the leader.

Description To maintain a safety distance to the next follower, a follower must control the coherence of the speed value received from the leader with respect to the previous one that has been received.

Actors

- leader
- follower n.1
- follower n.2

Basic flow The leader communicates its current speed to the followers, which compare it with the previous one received in order to detect big differences between them. In case of detecting an abnormal value, the new speed will be ignored by the followers.

Identifier UC4

Name A Follower does not receive any speed value during a certain interval of time, for instance 3 seconds.

Description A hacker connects to the leader flooding the WiFi with superfluous requests in an attempt to overload the system and to prevent sending a legitimate speed to the followers.

Actors

- leader
- follower n.1
- follower n.2
- hacker

Basic flow The leader is hacked and cannot transmit its speed for a long time. The followers should stop.

2.1.3.3 System Assets

The main assets are the integrity of the vehicles and their passengers. To achieve them there are several surfaces of attack on the CCCC vertical that may become target of attacks:

- Wireless communication: Providing access to 3rd parties to the vehicle data through the wireless communication system poses serious security and safety risks to the vehicle, passengers and other road users. It is very important to protect against to corrupted or fake messages attacks and to enable continuous availability, avoiding jamming.
- Safety critical and non-safety critical functions: Endangering safety-critical functions such as braking could produce hazardous events. In addition, non-safety-critical functions like the radio can be another safety risk in terms of driver distraction, drawing its attention away from the road. These functions could also be affected by an unauthorized user having access to sensors, actuators, inside communication systems or ECUs.

2.1.3.4 Hazard Identification

In the Connected & Cooperative Car Cybersecurity (CCCC) vertical, a safety violation can also be caused by security problems, so a Hazard Analysis and Risk Assessment (HARA) analysis has been developed in terms of security. The output of the HARA are the rated Hazards that will serve as root events for safety analysis and the Safety Goals that will be refined into more detailed safety requirements. The result of this HARA analysis for the CCCC vertical, that is illustrated in Figure 11, contains valuable knowledge to use for threat modelling and therefore, also for penetration testing. Detailed information on the development of the HARA analysis can be found in the section 3.2.1.1.1.

We also are using a Model-Based approach to carry out HARA analysis. In particular, we have represented the HARA using the tool AutoFOCUS3 [2] as a Goal Structuring Notation (GSN) model. The whole GSN tree for the CCCC vertical is depicted in Figure 12. Further information can be found in the section 3.2.1.1.1.

There are some advantages of using this notation. It breaks down different analysis into branches. Moreover, it is possible to extract TARA analysis from these trees as detailed in Section 3.2. The TARA analysis is shown in the section 2.1.3.5.

ID	Malfunction behaviour	Hazard	Location	Enviromental conditions	Platooning mode	Severity (S)	Rational S	Exposure (E)	Rational E	Controllability (C)	Rational C	ASIL	Safety Goal
1	Unintended full braking in platooning at cruise speed	Unintended longitudinal deceleration	Highway or Country road	Any weather conditions	Follower or Leader	S3	Every hazard that can lead to **collision** will be considered S3	E4	**High probability** of exposure for a Highway or Country road	C3	It is **uncontrollable** by driver or external people	D	**Avoid collision due to unintended full braking.** Vehicles in the platoon must have detection sensors to detect unintentional braking at highspeed of the vehicle ahead and start braking.
2	Lack/loss of braking	Insufficient logitudinal deceleration	Highway or Country road	Any weather conditions	Follower or Leader	S3	Every hazard that can lead to **collision** will be considered S3	E4	**High probability** of exposure for a Highway or Country road	C3	It is **uncontrollable** by driver or external people	C	**Avoid collision due to lack of braking**
3	Unintended acceleration	Unintended longitudinal acceleration	Highway or Country road	Any weather conditions	Leader	S2	Every hazard that can lead to **collision** will be considered S2	E4	**High probability** of exposure for a Highway or Country road	C3	It is **uncontrollable** by driver or external people	C	**Avoid long distances between the Leader and Followers**
4					Follower	S3	Every hazard that can lead to **collision** will be considered S3	E4	**High probability** of exposure for a Highway or Country road	C3	It is **uncontrollable** by driver or external people. **No reaction time**	D	**Avoid Collisions due to unintended acceleration.** Vehicles in the platoon must have detection sensors to detect unintentional accereration of the vehicle ahead and start braking.
5	Unintended steering	Unintended lateral steering	Highway or Country road	Any weather conditions	Follower or Leader	S3	Every hazard that can lead to **collision** will be considered S3	E4	**High probability** of exposure for a Highway or Country road	C3	It is **uncontrollable** by driver or external people	D	**Avoid possible collision due to unattended steering**
6	Loss of communication	The Leader does not send a speed value to the Follower. The Follower does not receive a speed value from the Leader	Highway or Country road	Any weather conditions	Follower or Leader	S3	Every hazard that can lead to **collision** will be considered S3	E4	**High probability** of exposure for a Highway or Country road	C3	It is **uncontrollable** by driver or external people	D	**Avoid possible collision due to loss of V2V information.** Vehicles in the platoon must have detection sensors to monitor the distance to the vehicle ahead and adjust the speed accordingly.
7	Erroneous communication	The Follower receives an erroneous speed value from the Leader	Highway or Country road	Any weather conditions	Follower	S3	Every hazard that can lead to **collision** will be considered S3	E4	**High probability** of exposure for a Highway or Country road	C3	It is **uncontrollable** by driver or external people	D	**Avoid possible collision due to tampering of V2V information.** Vehicles in the platoon must have detection sensors to monitor the distance to the vehicle ahead and adjust the speed accordingly.

Figure 11: HARA Analysis of Vertical 1 (Platooning).

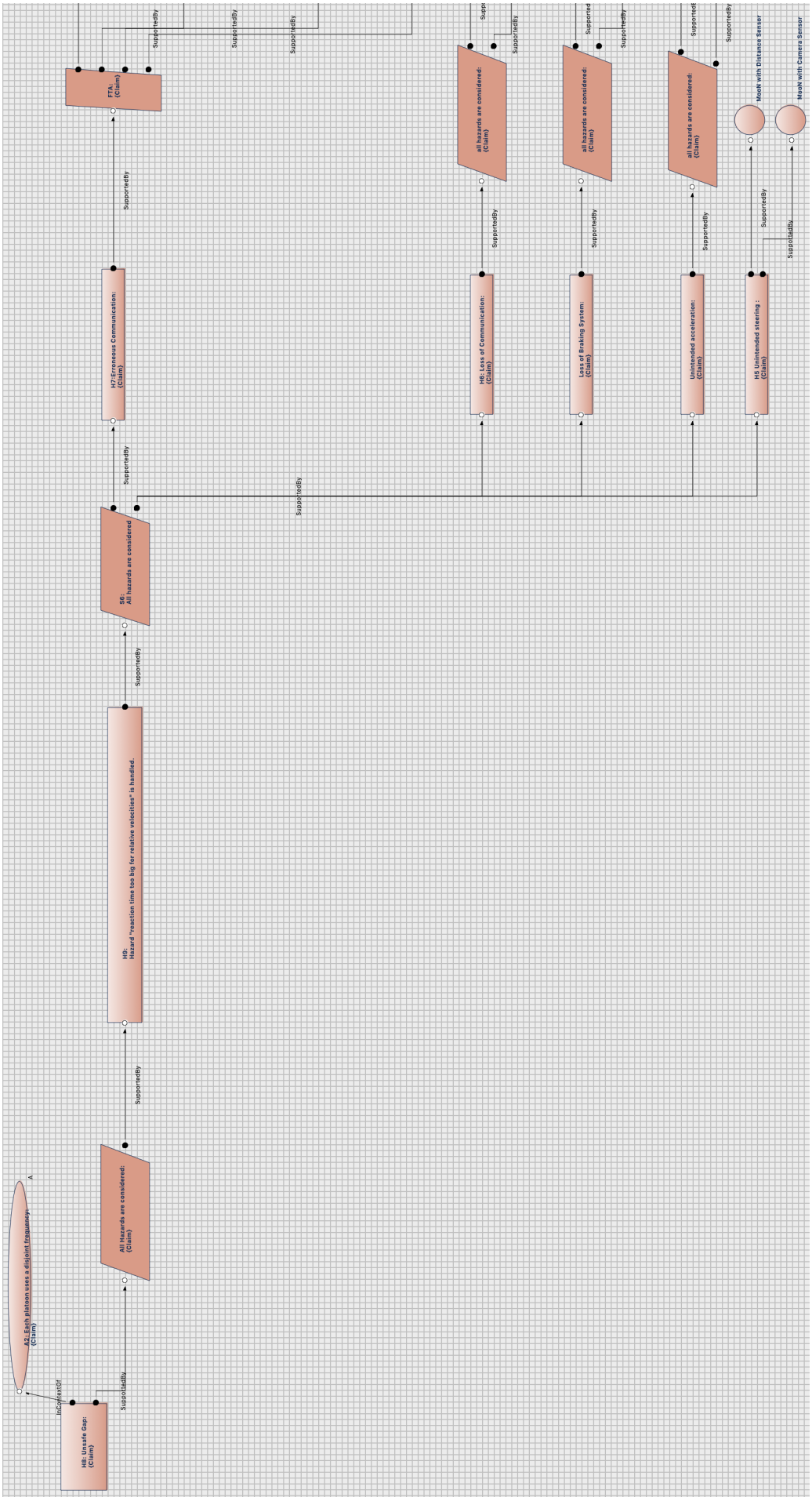


Figure 12: HARA represented in a model using Goal Structured Notation (1/2).

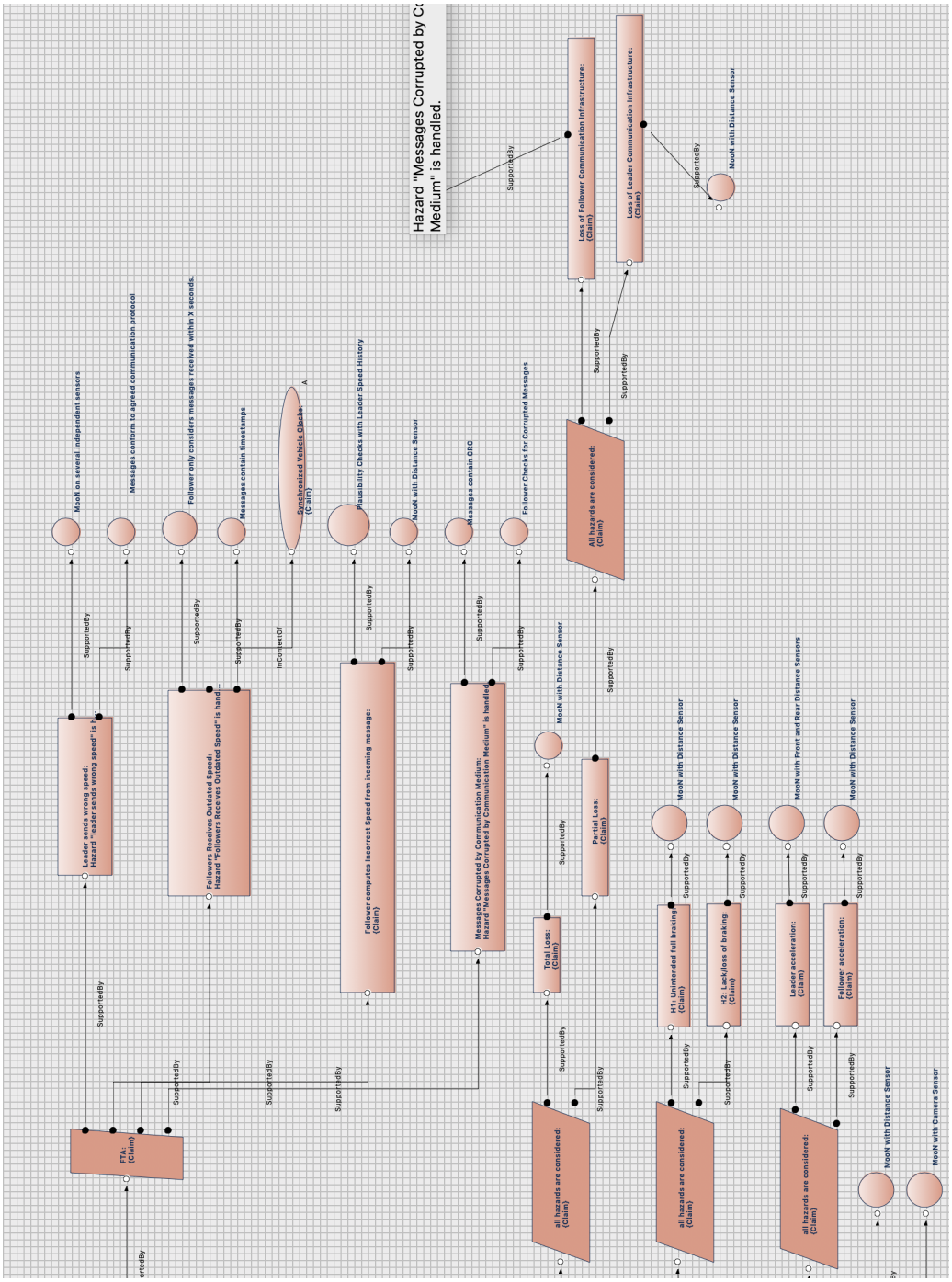
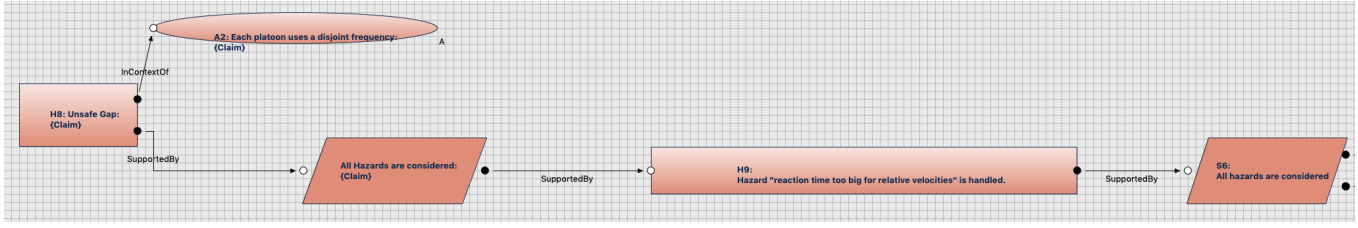


Figure 13: HARA represented in a model using Goal Structured Notation (2/2).

We detail some branches of the GSN-model below. It starts by using a strategy where are all relevant hazards are considered as shown below:

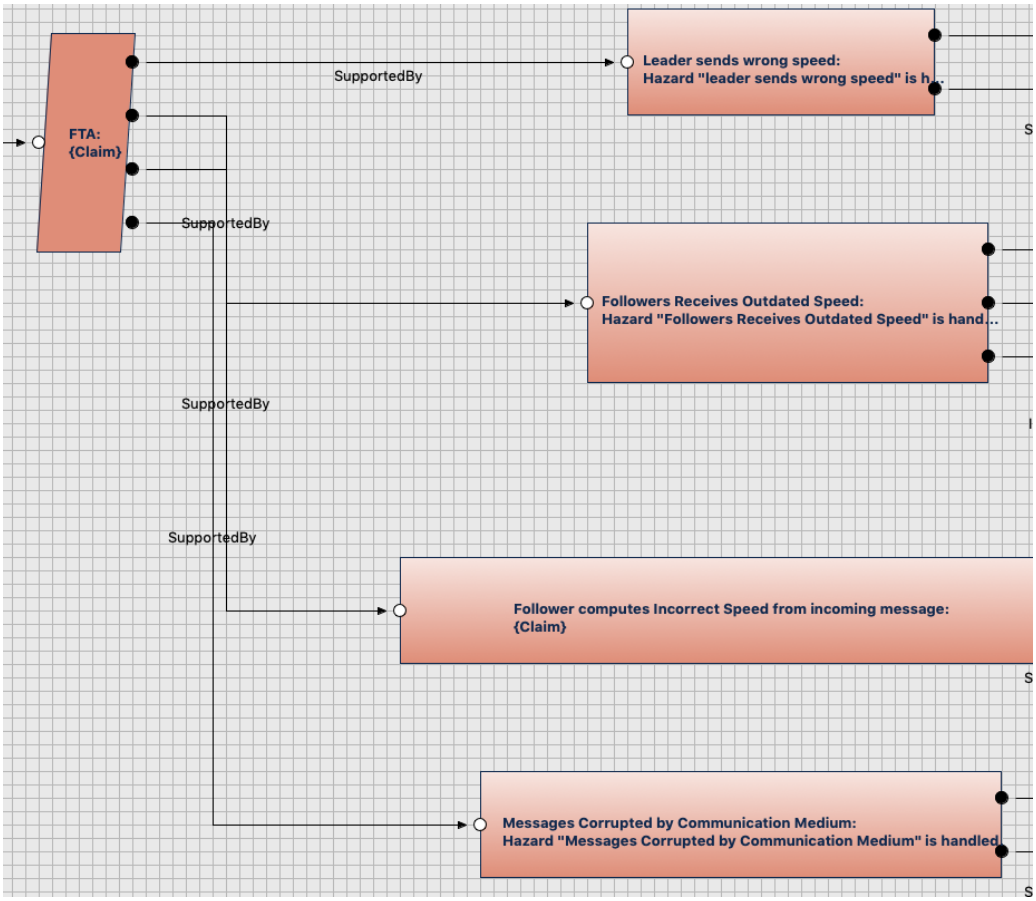


We considered until now five main hazards:

- Erroneous Communication

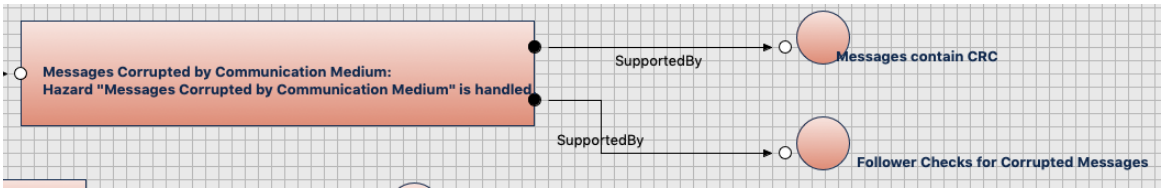
- Loss of Communication
- Loss of Braking System
- Unintended Acceleration
- Unintended Steering

The hazard on Erroneous Communication is further refined by using Fault Tree Analysis (FTA) as shown by the figure below:



The FTA analysis further refines how the main hazard can occur. For example, the leader can send a wrong speed, or the follower can compute an incorrect speed from the incomming messsage.

For each hazard, a solution, normally using a safety pattern, is deployed. For example, to control the hazard related to ”Messages Corrupted by Communication Medium” is shown below:



This means that a safety requirement to the platooning system shall include CRC checks.

2.1.3.5 Threat Identification

To detect all the vulnerabilities and potential threats of the system, it is appropriate to use vulnerability and threat analyses. Thus, some initial questions should be formulated.

- Will there be any sensitive data and/or personal identifiable information stored on or transmitted by?

- Will the system have any safety critical function?
- Which connection and/or communication will the system have?

In the CCCC vertical a Threat Analysis and Risk Assessment (TARA) analysis has been elaborated to identify threats and assess the risks. Identifying them, allows valuable resources to be applied to the highest risk potential threats.

The risk assessment component of a TARA considers the severity of the possible outcomes on the system and the likelihood that a potential attack can successfully be carried out, referred to as the attack potential. Focusing on the highest risk potential threats, the TARA helps to assign the most valuable cybersecurity controls during the application of more analysis techniques.

Using the tool AutoFOCUS3 [2] we have constructed the attack defense tree starting from the GSN model above with the HARA (cf. Figure 12). The whole tree is depicted in Figure 14. ThIS attack defense tree includes a number of attacks that can affect the safety-critical communication functions that could lead to the hazard *Unsafe Distance*, that is, causing vehicles in a platoon to be too close to each other.

The construction of attack defense trees using the safety analysis is a means to understand in a more structured fashion how attacks can cause safety problems and how they can be avoided. Further information on attack defence trees can be found in section 3.2.1.1.2.

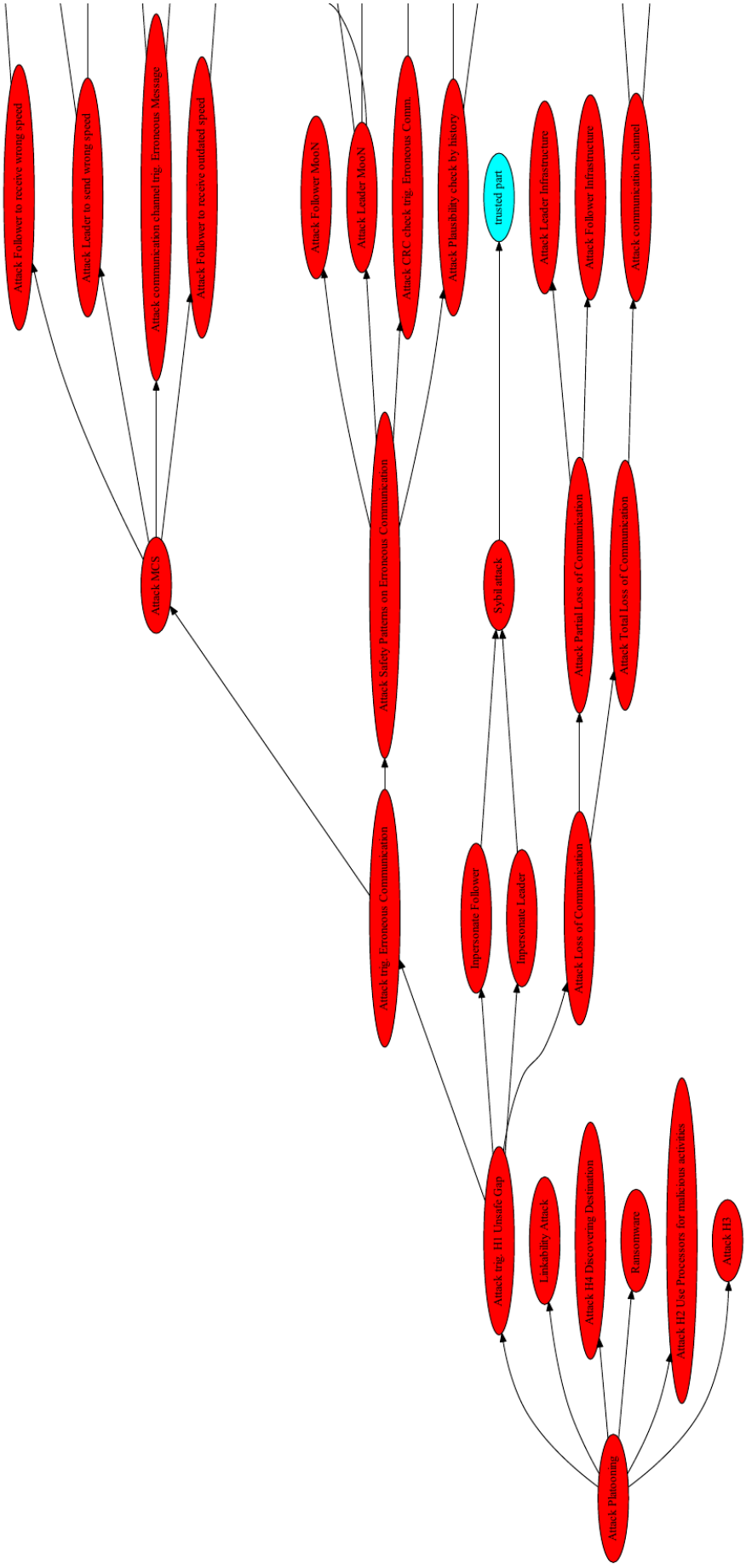


Figure 14: TARA represented as an Attack Defense Tree (1/2).

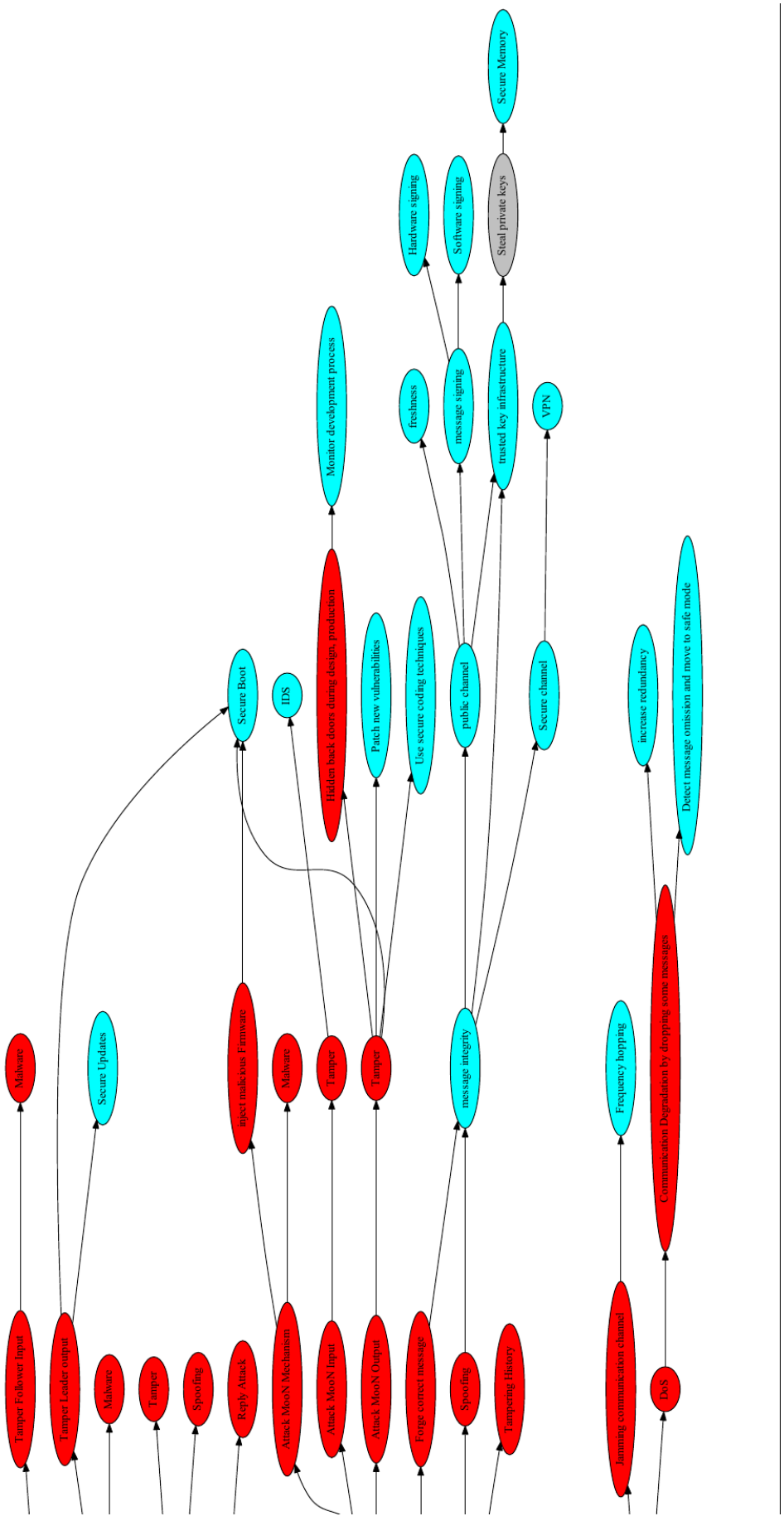
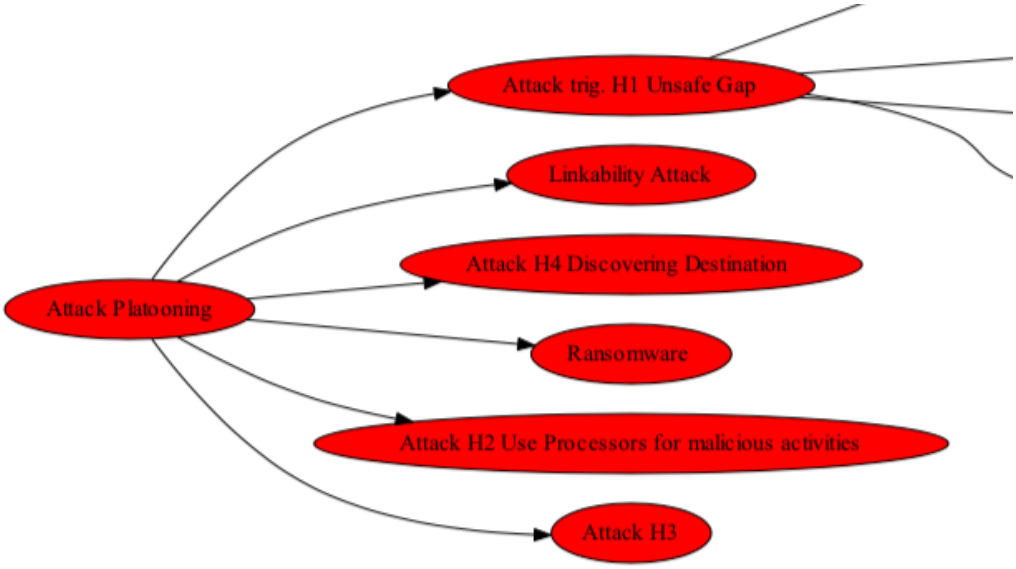


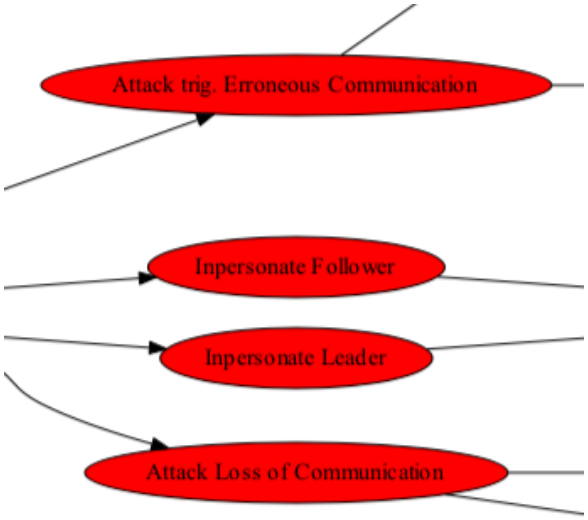
Figure 15: TARA represented as an Attack Defense Tree (2/2).

We zoom into some of its branches below, specially those that are related to the safety analysis described above. The root of the tree is shown below:

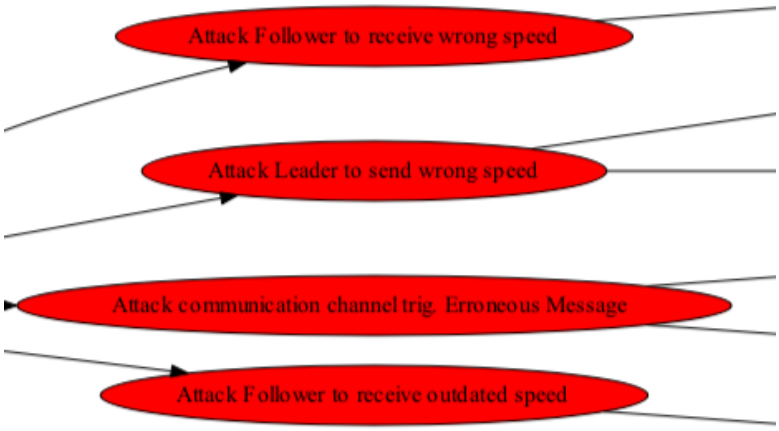


It considers a number of possible attacks to to the platooning scenario. However, we focus more on the threat where the attacker triggers an unsafe gap, which is safety-related.

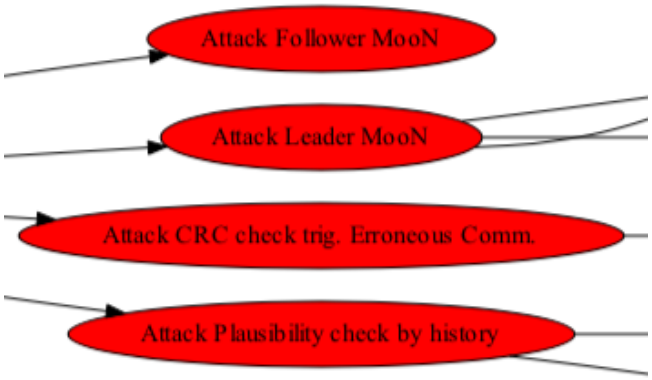
The branch on attack causing unsafe gap is further refined as shown by the following image:



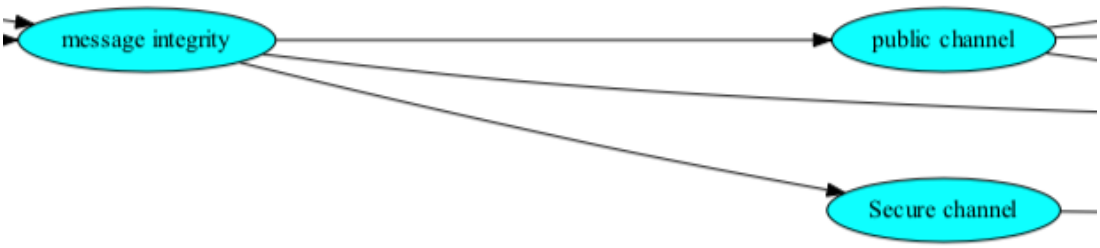
Notice that the attack triggering an erroneous communication of loss of communication are extracted from the safety analysis. The node “Attacker trig. Erroneous Communications” is refined by two nodes. One where the attack triggers this error by triggering an event that causes this event, as specified by the FTA analysis in the safety analysis. These are depicted in the figure below:



Alternatively, the attacker can attack directly safety control-mechanisms, such as voters (called MooN), or CRC checks:



Notice that these patterns are directly extracted from the safety analysis. To depict the specification of counter-measures, we expand the node “Attack CRC check trig. Erroneous Comm.”. A possible attack is to “forge correct message”. A way to counter-measure this is to ensure the integrity of message transmitted between the vehicles. As shown below, there are two cases for integrity depending on the type of communication channels available. Either the channel is public or private (such as in a VPN connection):



The refinement for the case when the channel is public is shown below:

One can guarantee message integrity in a public channel by using message signing, having a trusted key infrastructure, and adequate mechanisms to establish message freshness.

Moreover, message signing can be done by software or by hardware, as shown below:

Moreover, the trusted key infrastructed can be attacked and a possible counter-measure is to use Secure Memory. As shown below:

HARA and TARA are established methods enabling co-analysis. Furthermore, combined co-analysis techniques such as Failure Modes, Vulnerabilities and Effects Analysis (FMVEA) or extended Fault Trees (combination be-tween an attack and fault tree) are necessary in order to identify which attacks lead to safety failure modes or hazards.

2.1.3.6 Requirements Identification

The security requirements of the Connected & Cooperative Car Cybersecurity (CCCC) vertical are needed to satisfy the following high-level security objectives:

- maintenance of the intended operational performance of all vehicles in the platooning functions;
- ensurance of the functional safety of the vehicles;
- protection of the drivers and system assets.

The use cases defined in the section 2.1.3.2, the HARA defined in section 2.1.3.4 and the TARA defined in section 2.1.3.5 have been used as inputs to identify high-level requirements. These requirements have been grouped based on the four basic properties of any system's security: confidentiality, integrity, availability and authenticity.

The current list of requirements will be further extended once the safety and security analyses have been completed by February 2020[3]. Later on, safety and security co-analyses techniques, such as trade-off and security/safety by design techniques will be applied to refine the requirements, being the final set of requirements be completed by April 2020 using the Protection Profile document used as part of the certification according to the Common Criteria standard [26].

Safety Requirements (RSF)

[3]see section 3.2.2 for the roadmap of T5.2

Safety requirements are specifications for the risk reduction of the physical integrity of humans. The following requirements are relevant to guarantee the safety in the platooning scenario.

Identifier RSF_1

Description A collision shall be avoided due to a full braking or lack of braking.

Use case references UC1, UC2, UC3, UC4

Identifier RSF_2

Description A collision shall be avoided due to an acceleration of one or more of the followers.

Use case references UC1, UC2, UC3, UC4

Identifier RSF_3

Description A collision shall be avoided due to an unwanted steering of any vehicle.

Use case references UC1, UC2, UC3, UC4

Identifier RSF_4

Description A collision shall be avoided due to a loss of V2V information.

Use case references UC1, UC2, UC3, UC4

Identifier RSF_5

Description A collision shall be avoided due to tampering of V2V information.

Use case references UC1, UC2, UC3, UC4

Confidentiality requirements (RSC-CFD)

Confidentiality is satisfied if data or objects are not read by an unauthorized party. The following requirements are relevant to guarantee the confidentiality in the platooning scenario.

Identifier RSC-CFD_1

Description Provide end-to-end protection in the communication between the leader and the followers.

Use case references UC1, UC2, UC3, UC4

Integrity requirements (RSC-ITG)

Integrity is satisfied if data or objects are not changed (written) or generated by an unauthorized party. The following requirements are relevant to guarantee the integrity in the platooning scenario.

Identifier RSC-ITG_1

Description Unauthorized modification of data shall be prevented or at least detected. Whenever a message is received from the leader to the followers, the integrity of all information along the functional path must be ensured to prevent Man-In-The-Middle attacks using a secure channel.

Use case references UC2, UC3

Identifier RSC-ITG_2

Description Fressness of messages coming from the leader should be ensured, in particular to prevent replaying these data may trigger some undesirable behaviour.

Use case references UC2, UC3

Identifier RSC-ITG_3

Description The value of the speed coming from the leader should be compared with the previous one, in particular to detect abnormal values reception.

Use case references UC2, UC3

Identifier RSC-ITG_4

Description To guarantee the integrity of the message in a public channel Hardware or Software signing should be used.

Use case references UC2, UC3

Identifier RSC-ITG_5

Description To ensure that the key infrastructure is secure from attackers some counter-measures, such as employing a secure memory, shall be implemented.

Use case references UC2, UC3

Availability requirements (RSC-AVL)

Availability is satisfied if data, objects or services are available. The following requirements are relevant to guarantee the availability in the platooning scenario.

Identifier RSC-AVL_1

Description The availability of the WiFi communication between the leader and the followers should be ensured.

Use case references UC4

Authenticity requirements (RSC-AUT)

Authenticity is satisfied if an author of data or an object is who it claims to be. The following requirements are relevant to guarantee the authenticity in the platooning scenario.

Identifier RSC-AUT_1

Description When a reduction or increment of speed is performed by the leader, the message sent from the leader to the followers shall be authentic in terms of origin, content and time.

Use case references UC2, UC3

Identifier RSC-AUT_2

Description The system should control the access to the leader's WiFi access point, so that only the followers can connect to it.

Use case references UC4

Other Non Functional Requirements (RNF)

Identifier RNF_1

Description Long distances between the leader and followers shall be avoided.

Use case references UC1, UC2, UC3, UC4

Identifier RNF_2

Description Maneuvers of merging and splitting a platoon shall be done in adequate time not affecting the platoon performance, such as its speed.

Use case references UC1, UC2, UC3, UC4

Identifier RNF_3

Description Vehicles shall obey traffic laws, such as not travel in an illegal direction.

Use case references UC1, UC2, UC3, UC4

2.1.3.6.1 Protection Profile

In the development of the CCCC vertical and in particular for the “Vehicle Platooning”, in order to show the evaluability of the proposed solution, WP5 partners are working on the definition of a protection profile for a particular category of security requirements related to ”Secure communication between vehicles”.

This Protection Profile must include the minimum set of requirements, to be used in a secure automotive system, that the various technical solutions proposed by a generic product supplier, will have to implement in order to guarantee security in communication between vehicles.

From an analysis of the existing literature, the attention of the working group fell on a Protection Profile in particular [9] which was taken as a reference point and example for the definition of the document to be produced to be adopted as input for the phase of validation of the evaluability of automotive vertical, topic of Task 5.4 into WP5.

Naturally many of the elements dealt with by referenced PP are outside the contents of the vertical in question, and therefore the contents will be partially adopted in the “PP in definition” starting from the results of the risk analysis carried out by the tools identified in this deliverable and the security features that the prototypes adopted in the vertical will be able to achieve.

2.1.4 Standards and Certifications

Nowadays different standardisation approaches w.r.t. safety and security concerns exist. Those standards address the system development life-cycle not only from the perspective of safety concerns but also from security. Especially, those aspects of security which impact on safety are tackled. Besides, these recent standards promote safety and security co-engineering.

The following standards will be considered in the implementation of the Connected & Cooperative Car Cybersecurity (CCCC) vertical:

- ISO 26262 ”Functional Safety Road Vehicles” [60] for functional safety
- SAE J3061 “Cybersecurity Guidebook for Cyber-Physical Vehicle System” [103] for cybersecurity
- Common Criteria guidelines for Information Technology Security Evaluation [26]

Society of Automotive Engineering (SAE) J3061 and International Organization for Standardization (ISO) 26262 are linked and require integrated communication to maintain consistency and completeness, in fact the SAE J3061 guide follows the process framework described in ISO 26262.

More details about the safety and security standards for the automotive domain are described in Section 3.2.1.6.

The OpenCert tool (cf. section 3.1.11) will be applied helping the user in the hole assurance process of the CCCC Vertical. OpenCert is a management tool to support the compliance assessment and certification of Cyber-Physical Systems (CPS) spanning the safety and security. It will support knowledge management about the above standards (e.g. ISO 26262 and SAE J3061), regulations and interpretations, in a form that can be stored, retrieved, categorized, associated, searched and browsed. Opencert will assist on the development of assurance cases, evidence management, assurance process management, and global monitoring of the compliance with standards and regulations.

2.1.5 Assessment Methods and Tools

To achieve the goals and requirements of the CCCC vertical different tools and methods will be used. At design phases, Formal verification will be used to ensure the safety and security on the system design. On the left side of the V-model, for safety, some V&V methodologies such as Fault injection will be implemented, besides, Security penetration testing will be applied.

2.1.5.1 Simulation-based Fault Injection

After the system design architecture has been modelled, traditional safety analysis techniques and Fault Injection (FI) can be put together in order to perform a combined analysis of the system. Fault Injection consists in the accomplishment of controlled experiments where the observation of the system's behaviour in presence of faults is induced by the injection of faults in the system. FI has emerged as a way to perform an analysis at the same time that helps verifying and validating the safety of a certain design.

In the CCCC vertical, the Simulation-based Fault Injection technique will be used to simulate how a cyber attack can affect the vehicle motor, for example by changing the velocity to an abnormal value. It will be applied using the Sabotage tool (cf. section 3.1.12).

Virtual prototypes will be created (mathematical description of both the design state and of the vehicle properties). Faults will be injected into behavioural models, enabling an early Dependability assessment of the system: Safety Evaluation (fault effects not known) and Early Safety Validation. Thus, the observability and controllability on the target system will be enhanced.

2.1.5.2 Formal Verification (Security/Safety by design)

Formal methods have been successfully used for a wide range of applications to ensure that systems are safe and secure by design. For example, in a recent paper [79], we demonstrate how to use formal methods to determine which events that are trasmitted in Industry 4.0 applications have to be encrypted to ensure that intruders cannot inject or tamper messages and cause catastrophic events.

As formal methods rely on precise mathematical models, one can define notions of soundness and completeness based on precise assumptions, such as intruder models specifying intruders capabilities. However, due to the complexity of systems, formal methods do not scale, suffering from *state space explosion problem* to be used for the full verification of the final system, but works well on early phase design by systematically discovering vulnerabilities that could be exploited by intruders.

We enumerate some uses of formal methods for the vertical on connected cars that we will deploy:

1. **Safety and Security co-design:** Vehicles in the platoon shall keep the minimum safe gap to the vehicle in the front. Intruders can cause messages between vehicles to be delayed or even dropped, thus affecting the functions resorting on CACC that enable a safe gap. Formal methods can be used to understand which gap shall be maintained in the presence of such intruders. For example, in [52], we have demonstrated how UAV flight strategies can be impacted by uncertainties. The same machinery can be applied for the vertical.
2. **Protocol Security Verification:** Formal methods have been suceessfully used for the verification of security protocols, being used for the verification of large protocols [28], cyber-physical protocols [80], and against denial of service attack [117]. Vehicles in the platoon scenarios will establish properties, such as authentication or the exchange of confidential information, by means of protocols. We will use formal methods to verify these protocols for logical attacks.

2.1.5.3 Software Verification Methods

Software verification and validation (V&V) aim at increasing the level of assurance of software, particularly that of safety-critical and mission-critical software, whereby validation commonly refers to en-

suring that the final software product meets initial user requirements, while verification is performed throughout the different phases of the software development life cycle in order to identify flaws and bugs as early as possible [25, 46, 122]. Thus, verification activities target artifacts of the respective life cycle phase, e.g., requirements, software designs or actual code. Correspondingly, the activities comprise technical reviews, testing or formal verification, all of them representing dedicated research fields with considerable bodies of work. Moreover, one should distinguish verification techniques targeting the actual software product, and others targeting the development process, e.g, the capability maturity model (CMM) developed by the US Department of Defense Software Engineering Institute.

The verification techniques covered by the CAPE program focus on the actual software product. This holds true for fault injection, formal verification and penetration tests, as well as all the techniques and tools developed in task 5.3, which take modern software development trends into account, especially the ever increasing use of open source software as well as agile development methodologies that result in short release cycles. In more detail, the contributions of task 5.3 aim at avoiding the presence of common security vulnerabilities in own code, the presence of known vulnerabilities in 3rd party code and so-called supply chain attacks, the latter of which which leads to the injection of malicious code into ones software product (cf. Section 3.3).

2.1.5.4 Penetration Testing

*A*s part of the assessment and in order to validate the security goals and the security requirements of the system (which can involve safety goals and safety requirements) a grey box penetration test will be performed.

A holistic approach will be followed which involves ECU level, Internal communication level, Architecture level and External communication level, as it can be seen in the Figure 16.

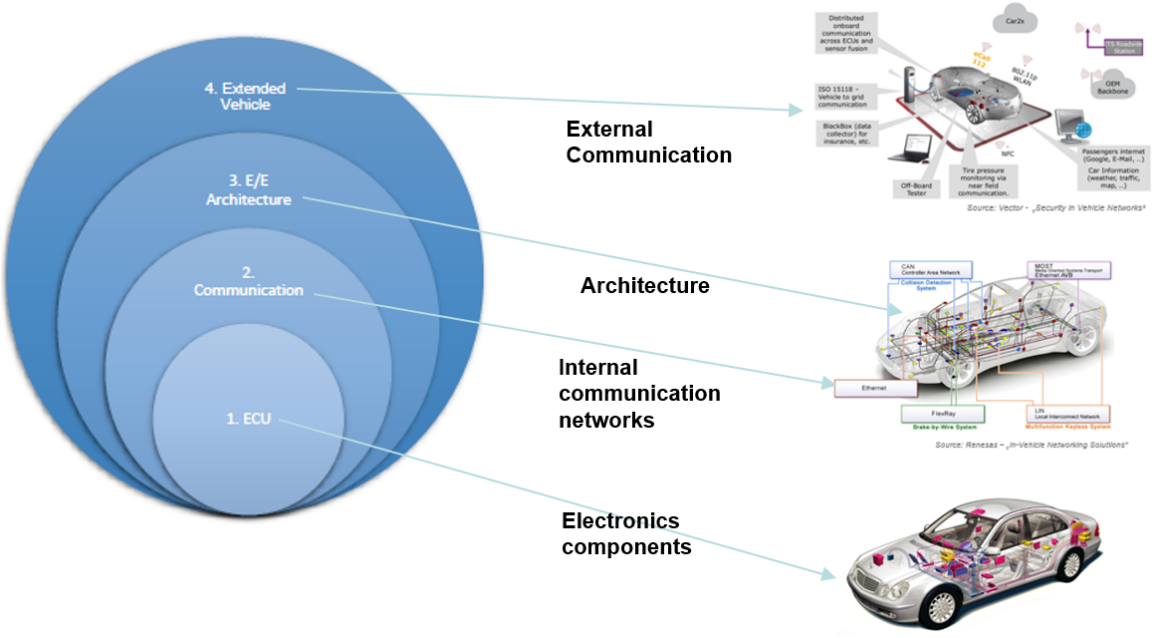


Figure 16: Illustration of the Penetration Testing Approach

This security penetration test will check the vulnerabilities of the system in terms of HW, SW, Communication interfaces and System's Architecture, taking into account all possible attack vectors and not only focusing on safety but also privacy and other concerns.

Previously to perform this analysis, the security requirements are going to be checked in order to get inputs and targets. On the other hand, some of the tools being developed in the project are going to be used, such as the visualization tool developed by UKON which offers an overview of the known SW vulnerabilities of each of the ECUs of the system. This kind of information helps the penetration

tester to perform more complex architecture attacks. Complementing the SW information of this tool with the HW vulnerabilities of the system is another task which is going to be done in order to detect complex attack vectors which can be tested.

The output of this assessment will feed again the system following the development cycle process proposed in this project as part of continuous integration (cf. Figure 16). This fact means, that from this analysis, some requirements may be added/changed as well as the Security concept (and also the Safety Concept) may be re-designed.

2.2 Vertical 2: Demonstration of a Complex System Assessment Including Large Software and Open Source Environments, Targeting e-Government Services (CINI)

The vertical 2 leverages the collaboration in the context of a Joint Lab between Fondazione Bruno Kessler (one of the institutions part of the SPARTA partner CINI) and Istituto Poligrafico e Zecca dello Stato (IPZS). IPZS is the Italian State Mint and Polygraphic Institute. Among other relevant activities, IPZS handles the production of the identity cards in Italy and the shipment of the CIE to the Municipalities. The goal of the Lab is the design of innovative authentication solutions based on the CIE, shown in Figure 17. CIE (version 3.0) is a personal identification document that is replacing the paper-based identity card in Italy and is used for both online and offline identification. The microprocessor of CIE is contactless and can be read using smartphone with a Near Field Communication (NFC) interface.

Figure 17: The Italian Electronic Identity Card (CIE).

To give an idea of the dimensions of the considered software environment, and the impact on the citizens, let us provide some numbers. The current number of CIE released to citizens is more than 13 millions. The number of downloads of the "companion app"—that performs the authentication mechanism through the CIE—is around 10,000. Given that, in this preliminary phase, only a few preparatory services are available, these numbers are going to rapidly increase, together with the number of the provided services.

The innovative authentication solutions based on CIE—which are topics of vertical 2—include many software components, most of them open source. Indeed, these components include:

- Security Assertion Markup Language (SAML) [86]: an open standard for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider;
- Shibboleth[4]: among the world's most widely deployed federated identity solutions, connecting

[4] https://www.shibboleth.net/

users to applications both within and between organizations. Every software component of the Shibboleth system is free and open source.

Some of the innovative authentication solutions based on CIE had been included in the notification process to the European Commission, according to the electronic IDentification Authentication and Signature (eIDAS) regulation [90]. This notification process included some preliminary steps, a presentation in Brussels (given by IPZS in January 2019), and a peer review phase. The process had been recently concluded and is currently in the *notified* status[5] This means that these solutions are recognized as authentication solutions by all the member states of EU. In order to avoid severe security issues, it is thus extremely important to ensure that the solutions guarantee the proper level of security.

For this purpose, this vertical has as goal to advance the cyber-security of the innovative authentication solutions based on the usage of the Italian national electronic identity card. In the next sections, we will discuss the specific case study we will investigate, namely identity management solutions based on CIE, the available infrastructure for demonstrating the effectiveness of our goal, and related assessment and certification requirements.

2.2.1 Case Study Objectives, Description and Relevance

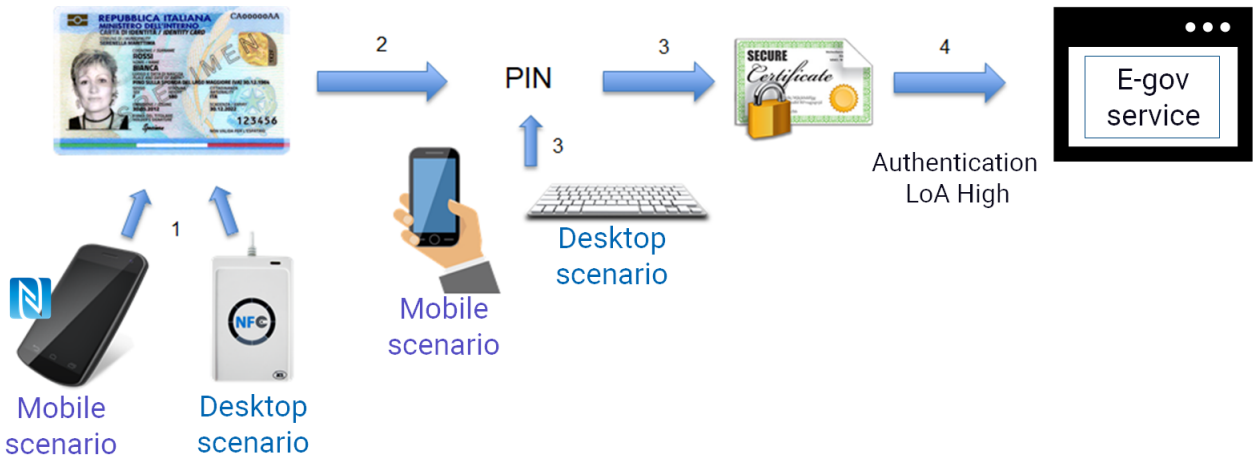


Figure 18: Overview of the case study of Vertical 2.

An overview of the authentication solutions based on the usage of CIE is reported in Figure 18. Authentication is initiated by the holder who is invited to enter the authentication PIN. Without this PIN no information can be read from the Card and it is therefore not possible to read data or initiate authentication transactions on the network without the knowledge of the holder. The CIE-based Italian Identification Scheme for accessing services envisages two mutual authentication scenarios: a so-called “desktop” scenario in which the user uses his/her CIE with a workstation equipped with a RF Smart card reader and the so-called “Middleware CIE” and a “mobile” scenario in which the user uses his/her CIE with an Android smartphone equipped with NFC interface alongside an authentication App. Given that these scenarios leverage the (cryptographic capabilities of the) Card, the level of Assurance (LoA) of the resulting authentication is rated high. The high assurance level refers to an electronic identification means, in the context of an electronic identification system, which provides, with regard to the claimed or declared identity of a person, a degree of protection from misuse or alteration of the identity, implemented in accordance with relevant technical specifications, regulations and procedures.

[5] https://ec.europa.eu/cefdigital/wiki/display/EIDCOMMUNITY/Italy+-+eID

From a security perspective, in [40], a security study of authentication schemes used in eID services is presented. The security analysis shows that 7 out of the 15 European eID services were vulnerable to XML-based attacks, enabling efficient DoS and Server Side Request Forgery attacks. On 5 out of the 15 eID services, the authors were even able to exfiltrate locally stored files and send them to an arbitrary domain. As pointed out in [40], this survey "reveals the complexity of current authentication systems, which is a natural consequence of the complex technology stack in use. Peculiarities of TLS, XML, SAML, and HTML/JavaScript/AJAX must be considered, and each of these technologies must be strengthened against potential attacks. Additionally, interactions of the various layers and potential security relevant consequences must be taken into account." Of course, the insecurity of a single component can bypass the security of the entire system.

Recently, a major vulnerability has been found in the eIDAS-Node Integration Package provided by the European Commission [27]. This vulnerability allowed an attacker to bypass the signature verification, allowing an attacker to send a manipulated SAML response to an eIDAS-Connector to authenticate as anybody.

These are only a couple of examples showing the demand for tools which facilitate the (automatic) security analyses of authentication solutions. The scenarios we considered in vertical 2, namely innovative authentication solutions based on the usage of CIE, is even more complex. Indeed, besides the proper usage of the SAML protocol, it involves the CIE, mobile technologies, communication channels based on NFC, customized versions of Shibboleth, just to mention a few.

One needs to answer the following question: How can one be reasonably sure that cyber-attacks cannot exploit vulnerabilities in the solution (for example, the communication channels, mobile application, middleware) and cause serious security issues?

2.2.2 Architecture and Technology of the Case Study

The authentication scenarios achievable by means of the CIE envisage, respectively, the use of a service from a Desktop workstation equipped with an RF reader and through a smartphone with NFC interface. The application protocol adopted is SAML v2.0 in the same mode implemented for the SPID [87] authentication system:

- for authentication requests (based on the construct `<AuthnRequest>`) the binding HTTP Redirect is used;
- for SAML responses (based on the construct `<Response>`) the binding HTTP POST is used.

The attributes are also set according to the mode envisaged for SPID.

2.2.2.1 Desktop Scenario

The desktop scenario envisages secure access to a service of a Public Administration using the computer browser and through the CIE. For such an authentication scenario, on his/her workstation the user configures a RF smart card reader and the "Middleware CIE", a software library available for Windows and Mac OSx operating systems, which allows the integration of the CIE within the guest operating system as an external cryptographic token. The Middleware CIE, in particular, interacts with the browser to carry out, completely securely and transparently for the user, the communication between the smart card reader and the microprocessor of the CIE for the purposes of establishing a secure and authenticated connection toward a server component for authentication provided by the Ministry of the Interior. This component verifies the validity status of the digital certificate, withdraws attributes linked to the holder of the CIE and propagates them, alongside the confirmation of authentication, to the service that the user intends to use.

The user is only required to insert the PIN to unlock the use of the private authentication key and complete the process.

The steps illustrated in the diagram in Figure 19 are described below:

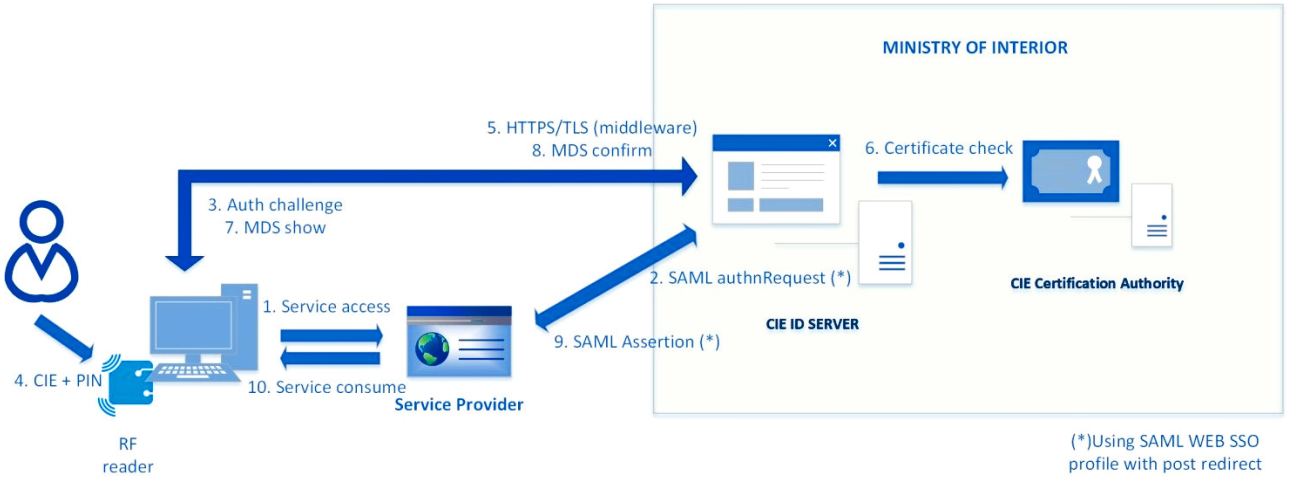


Figure 19: Desktop Scenario.

1. Through a web browser the user requests access to a service provider by specifying the CIE as an authentication mechanism;
2. The service provider sends an SAML authentication request (through the construct `<AuthnRequest>`) for access to the service to the component CIE ID SERVER;
3. The CIE ID SERVER component requests the user to use his/her CIE to authenticate him/herself;
4. By following the onscreen instructions, the user presents the CIE to the RF reader and enters the PIN;
5. Having verified the correctness of the PIN a secure HTTPS/TLS channel is created with the component CIE SERVER ID;
6. From the secure session the latter verifies the validity of the digital certificate associated with the user by contacting the Authentication CA of the Ministry of the Interior and retrieves the minimal attributes relative to the user from the certificate;
7. The user views the attributes that will be sent to the service provider on the browser;
8. The user authorises the transmission of the attributes displayed;
9. The component CIE ID SERVER redirects the user to the service provider by sending an assertion of successful authentication, including attributes, to the latter;
10. The service provider grants access to the service.

2.2.2.2 Mobile Scenario

The “mobile” scenario, shown in the diagram in Figure 20, provides for the use of the CIE as an authentication tool to gain access to a service provider. A necessary requirement for this scenario is the availability to the user of a mobile terminal with NFC interface which allows the interfacing of the CIE.

In detail, this identification scheme envisages that the user accesses a service provided by a service provider through the browser of his/her smartphone and selects the mode of access via CIE. When authenticating using the CIE, he/she is then redirected to a “companion app” that performs the authentication mechanism through the CIE with the CIE SERVER ID component described above.

The detailed steps of the procedure shown in the diagram in Figure 20 are described below:

1. The service provider sends an SAML authentication request (through the construct `<AuthnRequest>`) for access to the service to the component CIE ID SERVER;

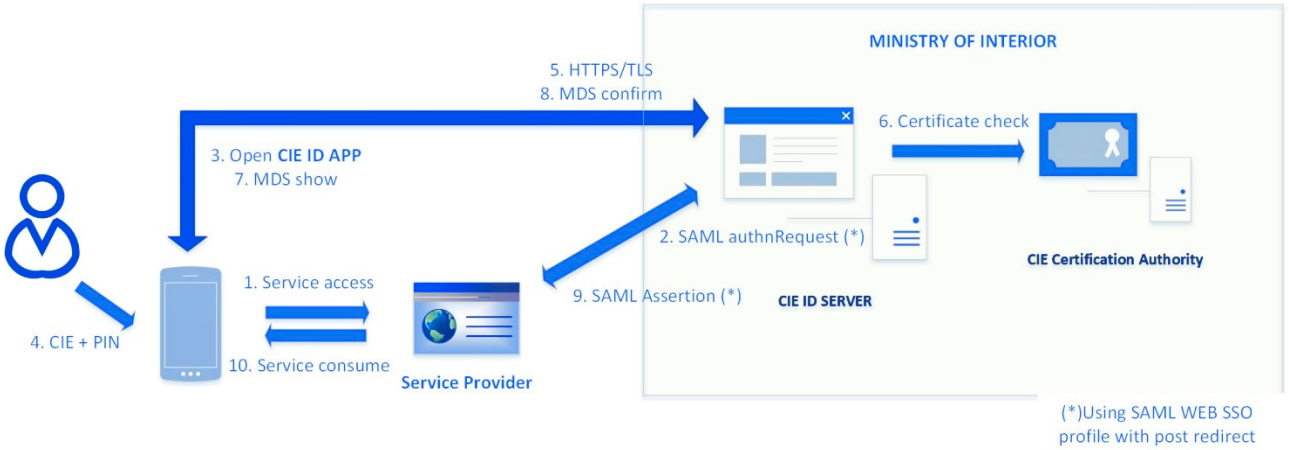


Figure 20: Mobile Scenario.

2. The CIE ID SERVER component requests the user to use his/her CIE to authenticate him/herself and sends a notification to the mobile terminal which triggers the launch of the app “CIE ID APP”;
3. By following the instructions shown on the app, the user presents the CIE to the smartphone’s NFC reader and enters his/her PIN;
4. Having verified the correctness of the PIN, a secure HTTPS/TLS channel is created between the app CIE ID APP and the component CIE ID SERVER;
5. From the secure session the latter verifies the validity of the digital certificate associated with the user by contacting the Authentication CA of the Ministry of the Interior and retrieves the minimal attributes relative to the user from the certificate;
6. On the CIE ID APP the user views the attributes that will be sent to the service provider;
7. The user authorises the transmission of the attributes displayed;
8. The component CIE ID SERVER redirects the user to the service provider by sending an assertion of successful authentication, including attributes, to the latter;
9. The service provider grants access to the service which takes place through the browser of the mobile terminal used in step 1.

2.2.2.3 Integration of the CIE-based Identification Scheme with eIDAS

Integration of the CIE-based identification scheme with the eIDAS infrastructure is carried out through the FICEP (First Italian Crossborder eIDAS Proxy) project. FICEP is the first “Italian cross-border server”: its implementation allows Italian nationals in possession of the CIE to access the network services of the Member States of the European Union using the CIE itself as a means of authentication. At the same time European citizens in possession of national electronic identification means recognised under eIDAS will have access to the services of the Italian Public Administrations.

The application protocol adopted is SAML WEB Single Sign-On (SSO) with Post Redirect and setting of the attributes in the same mode implemented for the SPID authentication system.

2.2.2.4 Scope of the Demonstrations

The vertical 2 includes several components, as depicted in Figure 21. Each of them must be carefully implemented in order to avoid security issues. The components currently identified for the demonstrations of vertical 2 are:

1. the CIE ID APP, and

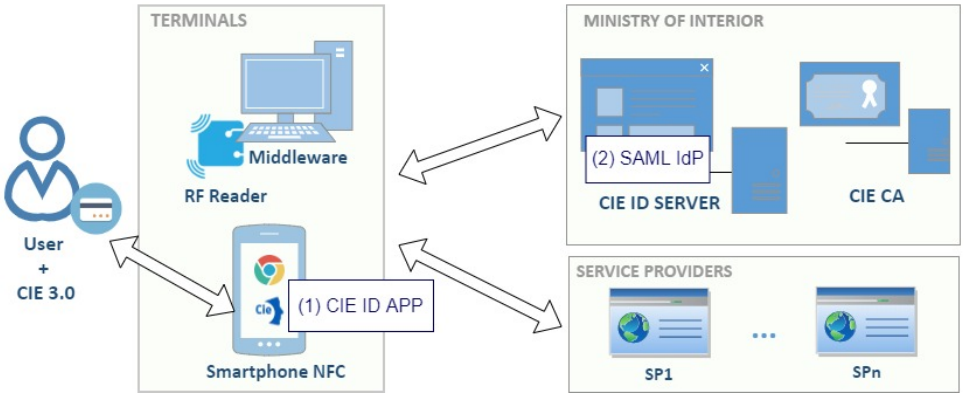


Figure 21: Components in the scope of the demonstrations.

2. the SAML IdP on the CIE ID SERVER.

The current software development process involves four main environments:

- a development environment and a testing environment hosted by FBK (part of the SPARTA partner CINI), and equipped with a version control system (Git-repository);
- a pre-production and production environments, hosted by the Italian Ministry of the Interior.

For each environment, it is available a copy of the CIE ID APP, capable of interacting with the IdP of the considered environment. The official CIE ID APP points to the production environment, and it is publicly available on the Google play store.

The preliminary versions of the components are developed in the development environment and tested on the testing environment, where a security analysis is performed. Then, the components are migrated on the Italian Ministry of the Interior servers, where the previous security analysis and tests are repeated. Finally, the components are moved on the production environment.

Given that the described scenarios for vertical 2 are in production, a security assessment methodology is of course already in place (based on the usage of automatic tools and manual inspection). Nevertheless, the current process will certainly benefit from the assessment framework developed in the context of the SPARTA project. Indeed, it will offer cutting-edge security analysis tools and will leverage novel paradigms (e.g., continuous integration), which will contribute to increase the overall security of the system.

2.2.3 Assessment and Certification Requirements

2.2.3.1 Security Requirements

The overall goal of these solutions is to guarantee the authorization property, namely the service provider must authenticate properly the user. To achieve that, the user directly authenticates to the identity provider (i.e. the CIE ID SERVER) and generates a signed authentication assertion. The service provider leverages this assertion to authenticate the user. Besides the authentication property, other properties must be taken into account. The confidentiality and integrity of the user data (and privacy related aspects) must be satisfied. In addition, the availability of the services must be guaranteed.

The list of requirements relevant for vertical 2 could be further extended, once the requirement identification phase has been completed in the coming months. We have currently identified the following two main requirements, which are relevant to guarantee the security in the proposed e-government scenario:

Identifier RS_1

Description The CIE ID APP must ensure the proper level of security. Indeed, a security issue in this mobile application could lead to severe security issues. For instance, a malicious user could steal the user's PIN or could authenticate on behalf of the victim.

Use case references Mobile Scenario

Identifier RS_2

Description The software implementing the service offered by the CIE ID SERVER must reach a proper level of assurance, so to avoid security vulnerabilities, which could lead to severe security issues.

Use case references Desktop Scenario, Mobile Scenario

2.2.3.2 Standards and Certifications

The presented scenarios must satisfy: (i) European (e.g., eIDAS) and (ii) national (e.g., SPID for Italy) laws, regulations and guideline principles that are particularly relevant to digital identity and privacy.

2.2.3.2.1 eIDAS and SPID

A central theme in the European Union is the definition of a common regulation on digital identity (e.g., eIDAS [90]) that guides both public and private sectors to define the security requirements and mitigate the known attacks and vulnerabilities. Many EU member states have also defined new national regulations (e.g., SPID [87] in Italy).

For instance, regarding privacy, in Article 5, paragraph 1, eIDAS states that the processing of personal data shall be carried out in accordance with Directive 95/46/EC of the European Parliament and the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Thus, all the principles related to the Data Protection laws, such as minimal disclosure, purpose specification and consent, must be directly applied to the eID scheme solutions.

The regulations pose interoperability requirements as well: the Italian eIDAS-Node is implemented according to the eIDAS technical specifications. It is integrated into the eIDAS Interoperability Framework [eIDAS IF] in accordance with the eIDAS Technical Specifications of the eIDAS Technical Subgroup [eIDAS Arch], [eIDAS SAML], [eIDAS Attributes], [eIDAS Crypto].

2.2.3.2.2 SAML SSO

Security Assertion Markup Language 2.0 (2005 - hereafter SAML) [86] is among the most widespread standards used to exchange authentication and authorization assertions (in XML formats). SAML requires that User (called Principal) is registered with an Identity Provider (IdP), which after receiving a request message from a Service Provider (SP) will respond with an authentication result (called assertion). An assertion contains data used by SP to decide whether to grant or deny for that Principal the access to a particular resource. A SAML assertion can contain three types of statement: authentication statement that indicates if a user is authenticated; attribute statement that describes some of the user attributes; and authorization decision statement that specifies whether a user is authorized to do a specific action on a specific resource. SAML defines different protocols, bindings and profiles. A protocol specifies which requests and responses are exchanged. A binding describes how SAML requests and responses are mapped into the communication and transmission protocols (SOAP or HTTP). A profile specifies a use-case scenario choosing a combination of assertions, protocols and bindings. Among the profiles defined in the standard, the most used is the web browser SSO profile [85]. A SSO protocol enables Users to login to an IdP only once and gain access to several SPs without requiring to authenticate for each one of them.

2.2.3.3 Assessment Methods

The complex scenarios considered in this vertical involve many components and communication channels. Each of them requires specific assessment methods, including formal verification, software verification methods, vulnerability assessment, and security analysis of mobile applications. Concerning the components in the scope of the demonstrations (cf. Section 2.2.2.4), and in order to validate the security requirements reported in Section 2.2.3.1, we plan to perform the following assessment methods. Notice that the list of the tools relevant for vertical 2 could be further extended, together with the details of the assessment methods, once the requirement identification phase has been completed in the coming months[6].

2.2.3.3.1 Software Verification Methods and Vulnerability Assessment.

We plan to adopt software verification and validation techniques aiming at increasing the level of assurance of the software. For this purpose, we will use the Steady tool (cf. Section 3.1.9), targeting the SAML IdP, deployed using Shibboleth on the CIE ID SERVER (cf. Figure 21). Indeed, in the context of vertical 2, the basic version provided by Shibboleth has been customized in order to support the interaction between the IdP and the mobile application which communicates with the CIE.

On the one hand, it is of utmost importance to assess the presence of common security vulnerabilities in the 3rd party code used. For instance, the known vulnerabilities affecting the specific version of Shibboleth integrated in the scenario must be taken into account. On the other hand, some vulnerabilities can be due to the customization phase, which could have injected novel issues.

The Steady tool will be used to detect whether both the Shibboleth version integrated in the scenario and the code implemented to customize the solution depend on open-source components with known vulnerabilities, and to collect evidence regarding the execution of vulnerable code.

2.2.3.3.2 Vulnerability and Risk Assessment of Mobile Applications.

In the “mobile” scenario described in Section 2.2.2.2, the user uses his/her CIE with an Android smartphone equipped with NFC interface alongside an authentication App (CIE ID APP). It is thus extremely important to leverage methodologies and techniques for the automatic security analysis and risk evaluation of Android mobile application CIE ID APP (cf. Figure 21). A security flaw in the authentication App could lead to severe security issues, allowing a malicious user to authenticate on behalf of the victim.

To this aim, we plan to use the tool Approver (cf. Section 3.1.4), for an in-depth, fully automatic security analysis of the authentication App, and a precise report of the security concerns. Besides that, we will possibly use other tools of the SPARTA assessment framework, like TSOpen (cf. Section 3.1.14) to be sure that the libraries used by the App under-development do not contain any malicious code.

In addition, we plan to apply to this vertical (some of) the continuous integration techniques which will be developed in the context of task 5.3 (cf. Section 3.3). In particular, we envisage the use of the plugin for the integration of Approver for the automated submission of the application package during the development phase (cf. Table 63), so to be able to identify vulnerabilities in the early stages of the software lifecycle.

[6] The T5.3 roadmap (section 3.3.3) details how the assessment methods identified and detailed in T5.2 will be applied in this vertical

2.3 Summary of Certification Requirements

The previous sections presented an overview of the assessment and certification requirements for CAPE verticals: CCCC and e-government services.

The CCCC vertical provides a platooning use case where several vehicle travel in a sequence formation led by a platoon leader. The objectives of the research are to demonstrate the safety and security co-engineering and how respective standards can be integrated and impact the assessment activities. This vertical will be illustrated through three demonstrations using small-scale vehicle infrastructure to simulate real life environments where safety and security strategies (and their connection) will be studied.

The e-government vertical is based on authentication solutions in the context of the CIE. The use case will focus on improving cyber-security aspects of those solutions through two scenarios (mobile and desktop) where a user authenticates with a central CIE identification server to get access to a service provider.

	Vertical 1 - CCCC	Vertical 2 - e-Government
Domains	Cyber-Physical Systems, automotive, safety and security, network communication, model-based engineering	e-government, authentication, authorization
Description of application	Vehicle platooning	Electronic identity card
Assessment requirements	ISO 26262, SAE J3061, Common Criteria	eIDAS, SPID
Assessment tools required	Simulation-based fault injection, formal verification, software verification, penetration testing	security analysis, software verification, penetration testing
Certification requirements	confidentiality, integrity, availability, authenticity	authentication, confidentiality, integrity, availability
Architecture	Cloud, Edge, IoT	Cloud, mobile, desktop
Technologies used	C/C++, Java, Python, Ruby, Cuckoo	SAML, SOAP, HTTP, NFC

Table 1: Summary of Certification Requirements

Chapter 3 SPARTA Assessment Specifications

This chapter describes assessment specifications for the SPARTA project as laid out in the CAPE program tasks, and provides high level realization roadmaps for each task:

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 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 the tools and techniques described in T5.1, T5.2 and T5.3 in the CAPE verticals

The outcome of the WP5 tasks takes the form of 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 integration of 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, ... 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.

3.1 T5.1 - Assessment Procedures and Tools

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. It studies assessment for IoT devices, hardware, kernel, software, communication infrastructures. It delivers practical tools for developers. It supports T5.2 and T5.3. This task will rely on an analysis of existing national cybersecurity certification initiatives such as the CyberEssential label, the ANSSI certification label, the BSI recommendation, the VDS certification, or the Italian national framework performed in WP11. The task will also consider systems where continuous monitoring and assessment is necessary such as the case of adaptive security. Task performers will engage in constructive competition based on a Program-Lead-approved benchmark defined at the start of the task.

3.1.1 SPARTA Cybersecurity Assessment Tool Framework

3.1.1.1 Overview of the Assessment Tool Framework

This section provides an overview of the SPARTA cybersecurity assessment framework. The framework is composed of:

- a list of assessment tools to that can be used during different phases of the security engineering lifecycle.
- a security engineering process description, that indicates in which phases each assessment tool can be used
- a safety engineering process description
- a common criteria process description

Figure 22 shows the SPARTA cybersecurity assessment framework. The process descriptions are shown in parallel, in order to highlight the time dependencies between the steps of a process, and between steps of different processes. The figure suggests that security and safety co-engineering can be performed, in parallel with cybersecurity Common Criteria certification. For example ”security requirements analysis” can be done in parallel with ”safety goals definition”, and finish with incremental certification of the ”security target evaluation (ASE)” (more specifically ASE REQ security requirements). The safety certification process is not considered in the SPARTA assessment framework because it is beyond the scope of the SPARTA project. The security engineering process covers both software and hardware development, however the focus is on software development. Each of these processes will be described in a separate section below. The framework covers the following phases of the software lifecycle:

- the design phase covering requirements, architecture, design, development, unit testing, integration testing, acceptance testing and deployment,
- the operation phase when a system is running in its target environment
- the end of life phase when the system is taken out of operation

The design life cycle is assumed to be iterative.

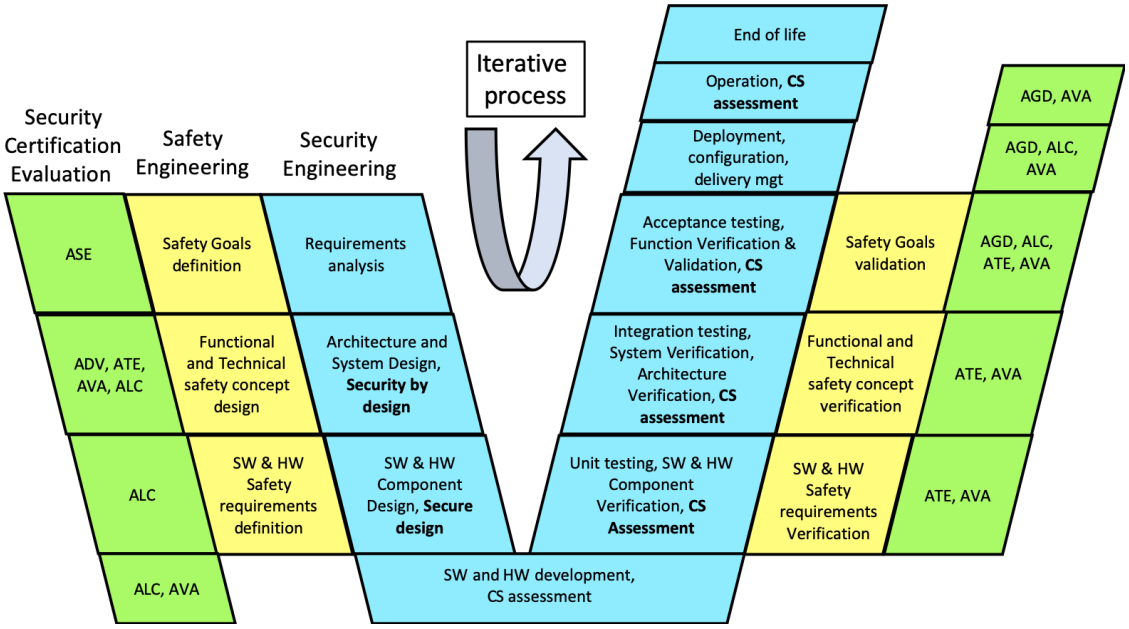


Figure 22: V-Model - Certification for safety and security

In this section we use the well known iterative V-Model development lifecycle to compare the different steps of the security engineering, safety engineering and cybersecurity certification processes. One of the main motivations in using the V-Model was to be able to compare security and safety development processes. While software development models have evolved towards agile processes that describe the iterative nature of software development, the hardware nature of safety engineering processes is less adapted to an agile process model. Using the V-Model is a good compromise to compare security engineering and safety engineering processes as was done in the AMASS ECSEL H2020 project [10] led by TECNALIA. For the cybersecurity certification process the Common Criteria is used. It is defined in ISO/IEC 15408 that defines the general model and evaluation criteria and ISO/IEC 18045 that defines the methodology for IT security evaluation. The use of the V-Model for describing the SPARTA cybersecurity assessment framework will be revised. A continuous integration approach is described later on in this chapter for integrating the different assessment tools. In light of that experience the choice of the V-Model will be revised and other development lifecycle models will be analysed such as a DevOps model that would be more in-line with continuous integration.

The assessment tools of the SPARTA assessment framework can be used during different phases of the software lifecycle. The table below introduces the different tools of the SPARTA assessment framework. Each tool will be described in more detail in a separate section below. The table shows the tool acronym, its name and the partner SPARTA partner responsible for its development.

Tool acronym	Description	Partner
RA	Risk assessment (NeSSoS)	CNR
SB	Sabotage	TEC
VA	Vulnerability assessment	SAP
FC	Frama-C	CEA
PT	Penetration testing	EUT
OC	OpenCert	TEC
VI	Visual investigation of security information	UKON
AF	Autofocus	FORTISS
MRA	model risk assessment for cyberphysical interconnected infrastructures	NCSRD
FS	Foreshadow-VMM Assessment Tool	CNIT
VCS	VaCSInE	CETIC
VA2	Vulnerability assessment	UniLu
LBD	Logic Bomb Detection	UniLu
RAAs	Risk Assessment of Android app	CINI
IMT	IDS and SIEM assessment tool (IDS)	IMT
BW	“Buildwatch” - A sandbox to monitor development processes	UBO

Table 2: CAPE Framework tools summary

Figure 23 and table 3 show in which phase of the security engineering process each of the assessment tools may be used. Table 4 show tools that may be used in which phase of the safety engineering process. The security and safety engineering processes and their phases will be described in separate sections below. The figure shows the majority of the tools on the security process phases. Four tools however can be used in the more general contexts:

- development process: AF and OC can be used during most of the phases of the lifecycle. AF is an integrated development environment that covers many phases, i.e. requirements until development including testing, simulation, and deployment of embedded system software. OC is an open product and process assurance/certification management tool to support the compliance assessment and certification of Cyber-Physical Systems (CPS). It can be used during many of the lifecycle phases both by the product developers as well as by an auditor.
- VA and VI can be used at organisation level and for systems of systems.

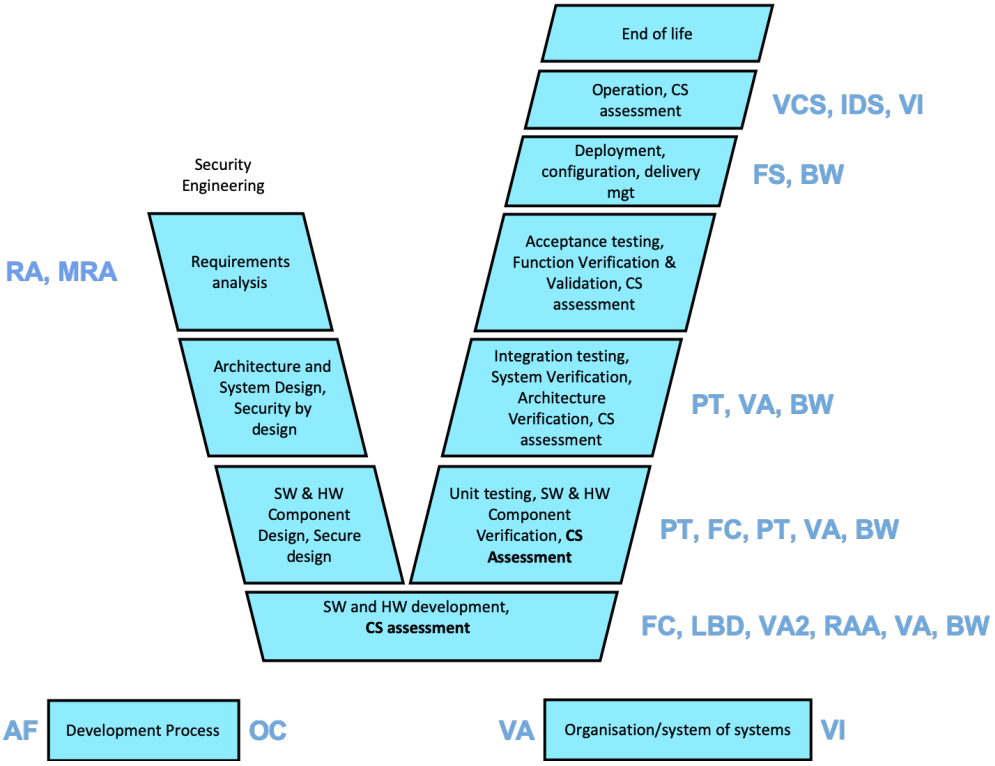


Figure 23: V-Model vs CAPE tooling

Tool / Lifecycle phase	Requirements analysis	Architecture Design	Component Design	Developmer process	Unit testing	Integration testing	Acceptance testing	Deployment	Operatior	End of life
RA	X									
VA				X	X	X				
FC				X	X					
PT					X	X				
OC				X						
VI									X	
AF				X						
MRA	X									
FS								X		
VCS									X	
VA2				X						
LBD				X						
RAA				X						
IDS									X	
BW				X	X	X		X		

Table 3: Summary of the SPARTA framework tools relation with security V-Model phases

Tool / Lifecycle phase	Goals definition	Fct/tech design	Reqs definition	Reqs validation	Goals validation	Fct/tech validation
OC	X				X	
SB		X				X

Table 4: Summary of the SPARTA framework tools relation with safety V-Model phases

3.1.1.2 Security Engineering Process

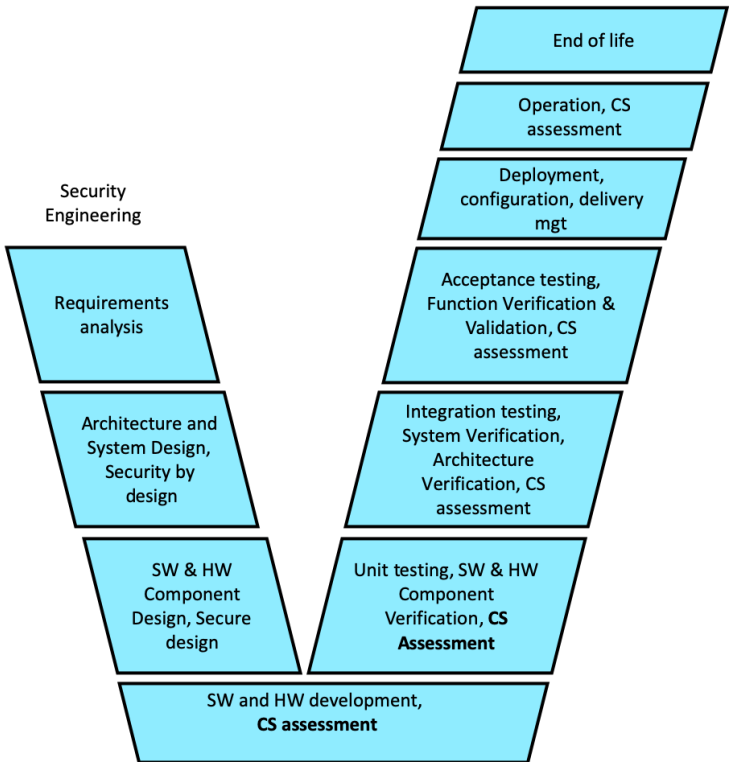


Figure 24: V Model - Security Engineering Process

Figure 24 shows the different steps of the iterative security engineering process. In the context of SPARTA security engineering is focused on software development. The hardware dimension is also considered in the process because the automotive vertical does involve both software and hardware components in the security requirements. The overall process distinguishes three main phases: (1) design and deployment, (2) operation and (3) end of life of the system.

The different steps of the security engineering design process are the following:

- Security requirements analysis: during the requirements analysis of the system the security requirements analysis step consists in defining the security requirements of the system and the assumptions made on the future operating environment.
- Security by design: during the architecture and system design step security by design consists in defining the security architecture such that the security requirements are satisfied.
- Secure design: during software and hardware component design secure design consists in designing the components of the security architecture such that architectural security requirements are met.
- Code security assessment: during software and hardware development cybersecurity assessment can be done on the software and hardware.
- Unit testing security assessment: during unit testing an assessment on the results of the security unit tests can be made with respect to security requirements on the security components.
- Integration testing security assessment: during integration testing an assessment on the results of the security integration tests can be made with respect to architectural security requirements.
- Acceptance testing security assessment: during acceptance testing an assessment on the results of the security acceptance tests can be made with respect to user security requirements.
- Secure deployment: during the deployment phase security monitoring of the operation phase can be deployed.

In addition to the (1) design and deployment phase cybersecurity assessment can also be made during the (2) operation and (3) end of life phases of a system.

3.1.1.3 Safety Engineering Process

In this case, to explain a safety engineering process, the automotive domain will be considered since the Connected & Cooperative Car Cybersecurity (CCCC) vertical, where safety is in place, is related to vehicle systems. The ISO 26262 standard process has some specific phases and subphases of the safety lifecycle. The planning of the safety activities regarding development is depicted in Figure 25.

Once the system concept is defined, a **Hazard analysis** and a **Risk assessment** is done where the probability of exposure, the controlability and the severity of a hazardous event with regard to the system is determined. With these parameters the Automotive Safety Integrity Level (ASIL) is defined and subsequently the **safety goals**, i.e. the top-level safety requirements are obtained.

During the subsequent phases and subphases, some detailed safety requirements are derived from the safety goals. The **functional safety concept** is developed by taking into account the preliminary architectural assumptions. This safety concept is developed by deriving the functional safety requirements from the safety goals and allocating them to elements within the architecture and the necessary interactions to achieve to safety goal.

Then, the **product development requirements at system level** are specified, including the design of the system architecture, the development of the technical safety concept, and the allocation of the technical safety requirements to the elements of the system. To finish with the definition of the requirements, the **lowest Hardware (HW) and Software (SW) requirements** are established. At this point the hardware and software are developed.

Henceforth, **safety verification and validation activities** are executed, which include the validation of the functional safety concept aspects, the implementation of the safety goals and the validation of the assumptions concerning the effectiveness and the performance of external measures.

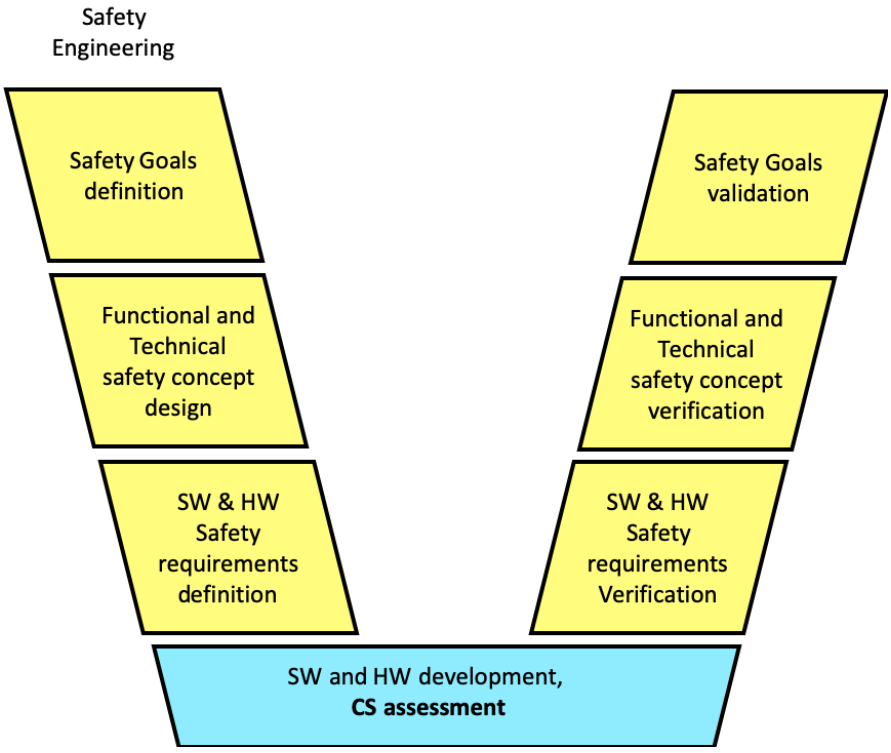


Figure 25: V Model - Safety Engineering Process

3.1.1.4 Cybersecurity Certification Process

This section describes the phases that will make up the process of evaluability for the purpose of certification of a generic target (product, system, process, etc.), it will be indicated which are the expected elements (requirements) that the various procedures and the various tools must have in the various phases of the activities that make up the process of evaluability.

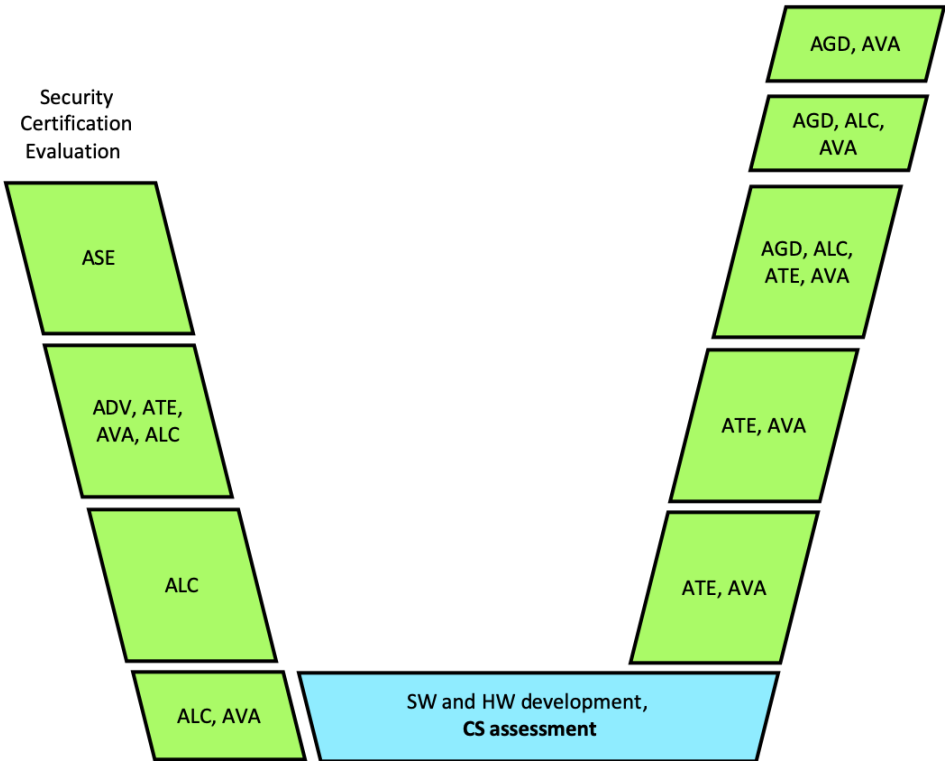


Figure 26: V Model - certification process

Among the main objectives of WP5 there is the simplification, but at the same time the completeness of the certification processes while trying to consider at the same time the needs that may arise in dealing with the analysis of very different elements (products, systems, processes and services).

In this direction the starting idea is to generalize what are the phases of a hypothetical certification process, which is by its nature adaptable to different elements (that we can define in general Target of Evaluation - TOE), but which also covers their entire life cycle.

Therefore in the initial hypothesis we focused on the idea of Cyber security seen as a process that accompanies a certain TOE throughout its life cycle as is perfectly outlined in Figure 3.19, naturally regardless of the need for a security certification of the TOE.

If we then want to go within the scope of the certifications, we can immediately verify that the phases of a generic process of evaluation/certification of the security of a TOE are going to map perfectly with those of Cyber security process just mentioned.

Taking as an example Common Criteria (ISO / IEC 15408), a standard that has become established and consolidated in recent decades, in the next scheme we can note how the various assurance classes, that characterize the activities carried out during a process of evaluation/certification of a TOE, go perfectly to map the needs defined in the various phases of the introduced Cyber security process.

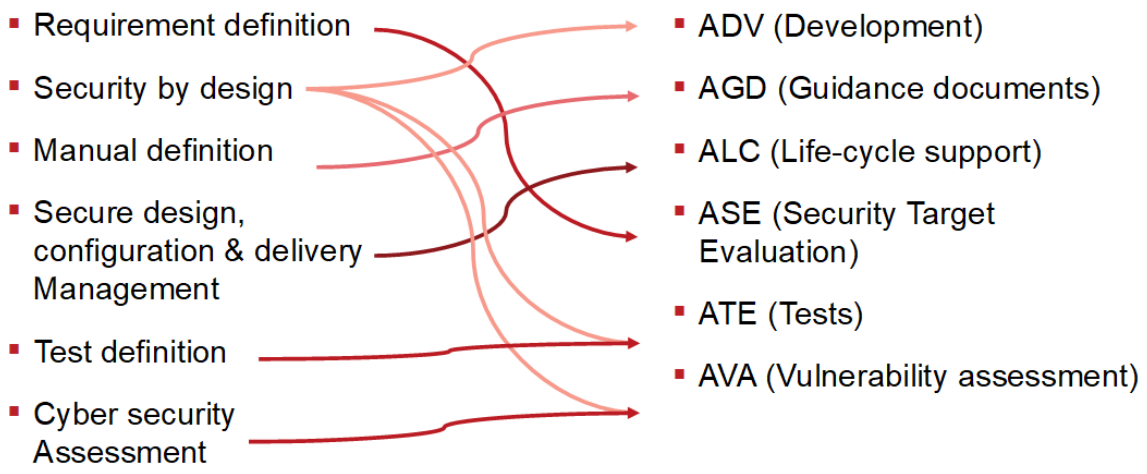


Figure 27: Common Criteria Assurance Classes mapping

For completeness, a rough description of these assurance classes and their correlation with the Cybersecurity process phases is given:

- 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 tho 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 requirements 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 analyse 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.

3.1.1.5 Cyber-security Certification Initiatives

This section provides an overview of national, international and European cybersecurity certifications initiatives, based on the work achieved in work package 11 SPARTA D11.1 - Mapping of International

and national cybersecurity certification initiatives [107] and the ECSO State of the Art Syllabus - Overview of existing Cybersecurity standards and certification schemes v2 [37]. Those initiatives can be separated in several main categories:

standards that provide rules, guidelines or characteristics for activities or for their results, aimed at achieving the optimum degree of order in a given context,

frameworks that provide voluntary guidance, based on existing standards, guidelines, and practices for organizations to better manage and reduce cybersecurity risk,

certifications that provide assurances that a product, process or service is in conformity with certain standards.

3.1.1.5.1 International cyber-security certification initiatives

Name	Type	Body	Country / Region	Industry	Verticals
ISO27K	Standard	ISO/IEC	International	General	Service providers and organisations
Common Criteria	Standard	CCRA/SOG-IS	International	General	Products and services
CIS Critical Security Controls	Set of controls	SANS Institute	International	General	service providers and organisations
Industrial Internet of Things Security Framework	Framework	Industrial Internet Consortium	International	Industrial IoT systems	IoT
Cloud security Alliance Cloud Controls Matrix	Set of controls	SANS Institute	International	Cloud service providers	Cloud
ISO-SAE 21434 & SAE J3061	Standard	ISO/IEC	International	Vehicles	Road vehicles
ISO 27034	Standard	ISO/IEC	International	General	Application security
IEC 62443	Standard	ISA/IEC	International	Industry	Automation controls

Table 5: International cyber-security standards and frameworks overview

ISO/IEC 27001 [62] provides requirements for an Information Security Management System (ISMS) . ISO describes an ISMS as 'a systematic approach to managing sensitive company information so that it remains secure. It includes people, processes and IT systems by applying a risk management process.' The standard describes how an organisation must set its security objectives and determine the risks that threaten these objectives to ensure its assets remain secure.

Common Criteria [26] for Information Technology Security Evaluation is an international standard (ISO/IEC 15408) for computer security certification. The standard is composed of catalogs of functional and assurance requirements, instructions on how to build specifications ("Security

Targets" based on "Protection Profiles") and conduct independent security evaluations based on these requirements that will provide evaluation ratings ("Evaluation Assurance Level (EAL)").

CIS Critical Security Controls [23] is a prioritised list of 20 security controls that an organisation could implement to thwart the most pervasive cybersecurity attacks. Every control consists of a number of 'sub-controls', which are concrete actions an organisation can take.

Industrial Internet of Things Security Framework [55] identifies, explains and positions security-related architectures, designs and technologies, as well as identify procedures relevant to trustworthy Industrial Internet of Things (IIoT) systems. It describes their security characteristics, technologies and techniques that should be applied, methods for addressing security, and how to gain assurance that the appropriate mix of issues have been addressed to meet stakeholders' expectations.

Cloud security Alliance Cloud Controls Matrix [29] is a catalog of requirements for security assurance in the cloud designed to provide fundamental security principles developed by the Cloud Security Alliance (CSA) to guide cloud vendors and to assist prospective cloud customers in assessing the overall security stance of a cloud provider. It covers fundamental security principles across 16 domains (e.g. Datacentre Security Asset Management, Mobile Security and Anti Malware, and Security Incident Management, E-discovery Cloud forensics, and Incident Reporting) to help cloud customers assess the overall security risk of a Cloud Service Providers.

ISO-SAE 21434 [58] aims at standardising the cybersecurity engineering process by specifying requirements for cybersecurity risk management for road vehicles and systems across the whole engineering lifecycle (design, production, operation, maintenance and decommissioning).

SAE J3061 [103] provides guidance on vehicle Cybersecurity by establishing a set of high-level guiding principles for cyber-physical systems: tools and methods for design, verification and validation of vehicle CPS, basic cybersecurity principles for vehicle systems, ...

ISO 27034 [61] offers guidance on information security to ensure that computer applications deliver the necessary level of security in support of the organisation's ISMS. It consists of 6 parts, some are still drafts: Overview and concepts (2011), Organisation normative framework (2015), Application security management process (expected publication May 2017), Application security validation (expected publication 2019), Protocols and application security control data structure (expected publication May 2017) and Case studies (2016). ISO/IEC 27034 targets various roles of the software development cycle: architects, analysts, programmers, testers, IT team, DBA's, Admins, but provides also guidance to auditors on e.g. how to evaluate the scope and process of verification measurements for the corresponding Application Security Controls.

IEC 62443 [53] applies to industrial automation and control systems in the operational technology domain. It provides guidance on policies, procedures, systems requirements, components requirements. The ISA/IEC 62443 take into account Industrial Automation and Control Systems (IACS) specificity, notably with Health, Safety or Environment (HSE) implications where the response should be integrated with other existing risk management practices addressing these risks.

3.1.1.5.2 National cyber-security certification initiatives

Name	Type	Body	Country / Region	Industry	Verticals
NIST CSF	Framework	NIST	USA	General	Critical infrastructures
IT Grundschutz	Standard	BSI	Germany	General	Service providers
ISKE	Standard	RIHA	Estonia	State and government	E-Government
Security Visa	Certification	ANSSI	France	General	General
Cyber Essentials	Label	CREST/GCHQ	United Kingdom	General	General
National Cyber Security Framework	Framework	CIS/CINI	Italy	General	General
VDS 3473	Certification	VDS	Germany	General	SME
FINCSC	Certification	JYVSECTEC	Finland	General	General
Cyber Fundamentals	Certification	State/Federal	Belgium	General	SME

Table 6: National cyber-security standards and frameworks overview

US - NIST Cybersecurity Framework for Improving Critical Infrastructure Cybersecurity [82] organizes basic cybersecurity activities at their highest level (Identify, Protect, Detect, Respond, Recovery) to manage cybersecurity risk by organizing information, enabling risk management decisions, addressing threats, and learning from previous activities.

Germany - IT Grundschutz [63] covers technical, organisational, infrastructural and personnel aspects of information security, providing a systematic approach to information security that is compatible with ISO/IEC 27001 through modules catalogs that contains a short description of the applicable components, approaches, and IT systems, as well as an overview of the threat scenario and the recommended safeguards.

Germany - VDS 3473 [119] [94] - "Vertrauen durch Siecherhiet" (VdS) is major independent testing institute with a focus on corporate security and safety. The VdS scheme is structured into domains (e.g. organisation, technology, mobile devices, prevention, ...) and four certified levels built on top of an online self-assessment: VdS quick audit, VdS 3473 Certificate, VdS ISO 27001 certificate and KRITIS for critical infrastructures.

Estonia - ISKE [57] is modelled on the IT-Grundschutz standard developed by the Federal Office for Information Security of Germany (BSI). ISKE forms a complete ecosystem from regulation to support tools and auditing practices. It spans all security domains from organisational security and risk assessment to technical methods and measures. Compulsory for the public sector since 2008, the ISKE standard employs a three-level assessment for an entity's security requirements (high, medium, low). The standard seeks to balance confidentiality, integrity and availability of data.

France - ANSSI Security Visa [14] helps companies and government authorities to select security solutions by qualifying and certifying those solutions in order to make sure they are compliant with corresponding standards.

United Kingdom - Cyber Essentials [30] provides guidelines, technical security controls, (self-)assessment and certification for protection against common cyber attacks, with a focus on keeping UK businesses safe.

Italy - National Cyber Security Framework [56] is based on the NIST Framework for Improving Critical Infrastructure Cybersecurity and is the result of a Public-Private-Partnership. It provides cyber security guidelines and a framework for SMEs, large enterprises, critical infrastructures and sector regulators.

Finland - FINCSC [45] - the Finnish Cyber Security Certificate is a certification mechanism that aims as ensuring business continuity and data protection for companies of all sizes. It focuses on improving understanding and providing a common criteria for cyber security and provides various levels of certification.

Belgium - Cyber Fundamentals Initiative will support SMEs to ensure a minimum level of security through a set of security requirements, basic technical controls and information on organizational and technical implementation of information security. The Cyber Fundamentals will help to address different compliance requirements, such as the GDPR, but it will also help to reduce the risk of becoming a victim of the most common cyber attacks.

3.1.1.5.3 European cyber-security certification initiatives

EU CyberSecurity Act [42] aims at addressing the fragmentation of European cybersecurity frameworks and initiatives. It is still in the early stages of development and will create a body of regulations by reviewing current frameworks and candidate schemes and establish a European cybersecurity certification framework for ICT products, services and processes. The European Union Cybersecurity Agency and the establishment of an EU cybersecurity certification framework (ENISA EU cybersecurity agency [41]) are tasked with improving the EU's reaction to cyber-attacks, cyber resilience and trust in the Digital single market.

eIDAS [90](Electronic Identification, Authentication and Trust Services) is an EU regulation on electronic identification and trust services for electronic transactions in the European Single Market. eIDAS provides standards for electronic signatures and electronic transactions used in online business or public services transactions.

3.1.2 Tools Descriptions and Development Plans

The following sections describe the use cases that will guide the development of the tools composing the CAPE Continuous Assessment Framework and the main user requirements derived from these use cases. From these requirements, we extract a prioritized list of software requirements to be implemented, and define the roadmap and the verification methods to address these software requirements. The tools are described in a short standardised way: identifier, name, owner, description, corresponding main assessment phases, technologies required, url, documentation, example tool usage and EU projects the tool has been built upon. When justified/applicable, the tools description will also include the following sections:

- User, software and certification requirements
  - Use cases: description of tool use cases in the relevant case studies (UC identifier, name, description, actors, basic flow), e.g. "Apply the security policy to the system"
  - User requirements: description of the certifications requirements, and when possible related to a compliance standard e.g. "Track and monitor all access to network resources (PCI-DSS)"
  - Software requirements: list of the software requirements of the tool (SR identifier, name, description, actors, basic flow)
- Specifications
  - Description of components: description of the components that the tool consists of

- Architecture: description of the tool architecture where components are presented, in order to define a detailed tool roadmap
- Development roadmap: relates the use cases and the architecture, it also describes how the proposed architecture will be realized
- Software verification and validation plan: list of methods for verifying the software requirements. These requirements are intended to be implemented in the different design and development cycles and will be verified and demonstrated upon their completion as part of WP5

For brevity purposes, some of the tools are only briefly described in the next sections, and further details (requirements, specifications) are explained in the appendix chapter 8.

3.1.3 Frama-C (CEA)

Identifier	FC
Name	Frama-C
Owner	CEA
Main functions	Static analysis, code comprehension and audit, verification of safety and security properties
Description	A platform for sound analyses of C code, based on formal methods.
Assessment phases	Development, Unit testing
Technologies required	C source code must be available; Unix-like environment with OCaml
URL	https://www.frama-c.com
Documentation	• User manual: https://frama-c.com/download/frama-c-user-manual.pdf • Tool paper: http://julien.signoles.free.fr/publis/2015_fac.pdf
Example usage	• Exhaustive identification of runtime errors (e.g. buffer overflows), mostly automatic • Proof of functional properties via code annotations • Code exploration, navigation and audit, augmented with a display of all possible values at runtime
Continuing work from projects	• U3CAT - Critical C code analysis (French Research Agency) [116] • STANCE - Source code analysis toolbox (FP7) [108] • VESSEDIA - Software analysis tools for IoT (H2020) [120], • DECODER - Unified knowledge base for software projects (H2020) [32]

3.1.3.1 User requirements description

UC1	Runtime errors and vulnerability identification via static analysis
Description	Apply an abstract interpretation-based static analysis to a test case to obtain an exhaustive list of possible runtime errors and security vulnerabilities related to deviations from the standard, including buffer overflows, null pointer dereferencing, uninitialized variable reads, division by zero. This use case is mainly related to T5.3, as one of the analysis tools in the CI/CD pipeline. This use case can eventually contribute to T5.2, for code-level properties related to both security and safety (e.g. memory safety).
Actors	Software developers & testers
Basic flow	Frama-C is configured to parse a whole program or a specific function to be tested; the analysis is triggered manually or via an automated mechanism (e.g. once per day). Results are presented in a standardized format (e.g. Static Analysis Results Interchange Format (SARIF)), or via the Frama-C graphical interface, for manual inspection.
UC2	**Code audit accelerated by a value analysis**
Description	Augment the inspection of a program (possibly performed by an external user, who did not participate in its development) by providing an exhaustive list of all possible variable values and code execution paths at each program point, for each variable and expression in the program; equivalent to a ”static debugger” which does not require running the code and which provides values for all possible program executions. This use case complements the previous one and can be used for the same tasks (mainly T5.3, eventually T5.2), but not at the same scale: while UC1 is oriented towards large-scale, automated verification, UC2 is related to manual assessment, e.g. after a potential vulnerability has been found.
Actors	Code auditor
Basic flow	Either Frama-C is configured to parse the code to be audited, or the developers provide a saved state of the analysis. The auditor uses a graphical interface to explore the code and understand all possible behaviors, e.g. to explore whether a potential vulnerability can happen, and under which conditions.

Table 7: Frama-C - Use Cases

UR1.1	Quasi-automatic analysis configuration
Description	Initial configuration of the analysis on a code base with minimal effort and required expertise
Actors	Software developers & testers
UR1.2	**Exchangeable analysis results**
Description	Analysis results must be exchangeable with other tools, for reuse and integration
Actors	Software developers & testers
UR2	**Audit-centered analysis exploration and report**
Description	The tool must produce a specific report for auditors, based on information provided by the developers and tailored for the purposes of reviewing an analysis, including its parametrization, and environment details. This analysis is complemented by a graphical interface focused on assessment needs.
Actors	• Software developer (environment description provider) • Code auditor (environment description consumer)

Table 8: Frama-C - User Requirements

SR1.1	CI-based set of parametrization options + example use cases
Description	Definition of a CI-oriented set of parametrization options to minimize the expertise necessary for an initial setup. A set of diverse code bases is included to ensure the designed parametrization performs as expected.
Actors	Software developers & testers
Basic flow	Identify and collect a set of representative, varied code bases for the analysis; define an automatic parametrization that ensures the best performance/precision trade-off on such set, adding new metrics and options as needed.
SR1.2	**Standardized output format**
Description	Support of a standardized format for analysis results, such as the SARIF standard.
Actors	Software developers & testers
Basic flow	Adapt the existing analysis result messages to conform to SARIF.
SR2	**”Audit” mode**
Description	Two sets of specific configurations and analysis features which allow, on one side, a developer to provide a complete description of the environment (options, external files, annotations, etc.); On the other side, a different set of configurations and features aimed at a code auditor, which allows verification of the previous artifacts and conformity of the environment. A graphical interface and visualization features help with code navigation and comprehension.
Actors	• Software developers • Code auditors
Basic flow	Produce a report of the analysis including environment information. This report is sent to an auditor, along with the code, who can verify its conformity, using a graphical interface as support.

Table 9: Frama-C - Software requirements

3.1.3.2 Technical specifications

Frama-C is a platform for C code analysis based on formal methods. It is comprised of several modular parts, which include code transformations, safety and security analyses, and a graphical interface to explore results and perform semi-interactive proofs.

In CAPE, CEA's focus is to improve one of the main analyzers of the Frama-C platform, called Eva, a value analysis based on abstract interpretation. It performs an automatic, whole-program static analysis which outputs an extensive list of possible runtime errors. Eva also provides information about each program variable at each statement, for all possible executions, easily accessible via a graphical interface. The current architecture of Frama-C/Eva is presented in Figure 28.

Given the two use cases related to Frama-C/Eva, there are two main modes of usage of the analyzer:

CI mode (automatic) : Eva is used as a static analysis tool, similarly to a code sanitizer, during a build process. A fast, automatic analysis is required, outputting data for a continuous integration process.

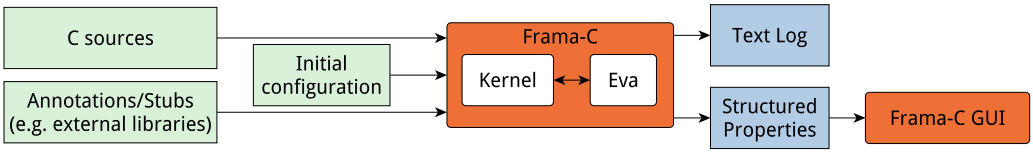


Figure 28: Frama-C/Eva's current architecture

Audit mode (interactive) : Eva is used to augment the auditor's understanding of the code, complementing but not replacing human expertise during an assessment. Frama-C's graphical interface provides the set of all possible variable values, plus code navigation possibilities, providing points-to and aliasing information, and evaluation of arbitrary expressions.

Concerning the automatic use mode, Frama-C/Eva has been historically developed for in-depth analyses of safety-critical code bases developed using a traditional process, with few revisions and a long assessment period. In CAPE, the transition to a CI-based analysis with rapid assessments imposes changes to its architecture, as illustrated in Figure 29.

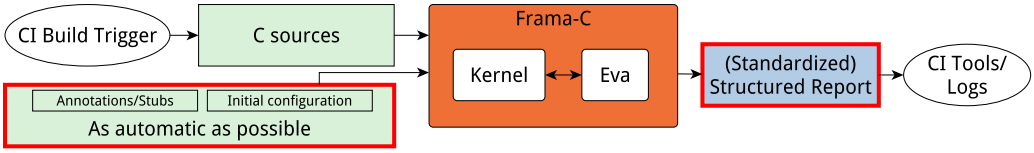


Figure 29: Frama-C/Eva's architecture for CI builds

For the audit mode, the goal is to complement automatic analysis and to support external assessments taking into account the environment, subject to changes. Figure 30 highlights the differences with respect to the existing architecture.

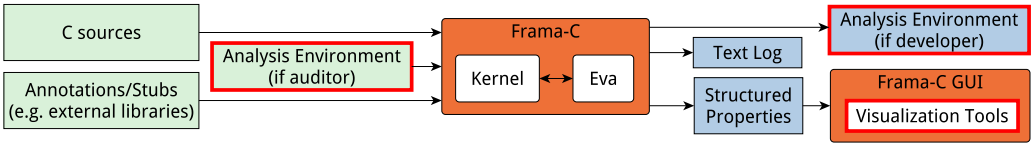


Figure 30: Frama-C/Eva's architecture for audits

3.1.3.3 Development roadmap

Use Case	Architecture components	Realisation	Involved partners
UC1	Frama-C kernel	Simplify/automate parsing and initial setup	CEA
UC1	Markdown-Report plug-in	Produce outputs in standardized format (SARIF)	CEA
UC2	Frama-C kernel and GUI	Produce environment summaries and check their conformance	CEA

Table 10: Frama-C - Use cases, realisations and architecture

Both UC1 and UC2 will be implemented in D5.3, for the demonstrator prototypes. UC2 requires more user interaction and feedback, so it is expected that it should evolve more significantly before the final demonstrator.

3.1.3.4 Software verification and validation plan

SR id	Description	Verification method	Demonstration scenario
SR1.1	CI-based configuration and use cases	Check applicability and usability on a set of existing code bases	Set of open-source code bases
SR1.2	Standardized output format (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 11: Frama-C - Demo scenarios and verification methods

3.1.4 Approver (CINI)

Identifier	RAA
Name	Approver
Owner	CINI
Main functions	SAST and DAST of Android apps; Risk Assessment of Android apps
Description	Approver is an automatic toolkit for the in-depth, fully automatic security analysis of mobile applications. Approver automatically detects, evaluates and provide comprehensive reports explaining the security risks hidden in the mobile applications. The key features include, but are not limited to: • Advanced Application Analysis based on state-of-the art static analysis techniques • Automated Security Policy Verification, ensuring that mobile apps comply with security requirements and regulations • Risk Score and Reports. Detailed, per-app risk reports that summarize the security concerns of the analyzed applications.
Assessment phases	Development process
Technologies required	Approver is available as a SaaS tool; in case of on-premise solutions it requires Docker (docker compose and docker swarm), VirtualBox, MySQL, and MongoDB.
URL	https://approver.talos-sec.com
Documentation	Not publicly available
Example usage	Security evaluation and risk assessment of Android apps during development; VA/PT of Android Apps; Risk assessment of Android apps in corporate environment

3.1.5 Foreshadow-VMM Assessment Tool (CNIT/University of Rome Tor Vergata)

Identifier	FS
Name	Foreshadow-VMM Assessment Tool
Owner	CNIT / University of Rome Tor Vergata
Main functions	Assess the presence of the Foreshadow-VMM Vulnerability and measure the troughput of the covert-channel
Description	The tool spawns two virtual machines (VMs), which will act as victim and attacker VMs. It then repeatedly applies the Foreshadow-VMM attack to determine if the CPU has the flaw and, if so, gets statistic of the covert channel as throughput and false read rate.
Assessment phases	Deployment
Technologies required	Intel VT-x and Intel HyperThreading
URL	https://gitlab.com/marcux_95/l1_tf/
Documentation	https://www.researchgate.net/publication/335340376_Exploiting_Foreshadow-VMM
Example usage	In a cloud environment, detect if a malicious VM can leak information from other VMs running concurrently.

3.1.5.1 User requirements description

UC1	**Assesment of L1-TF Vulnerability**
Description	The Cloud Infrastructure owner can tell if two virtual machines, managed by two different verticals, can exfiltrate information from each other.
Actors	• Cloud Owner • Cloud Infrastructure • Victim VM • Attacker VM
Basic flow	The Cloud Owner can instantiate on the cloud infrastructure, on top of the Hypervisor, the two VMs. A requirement is that the two VMs must share the same core, thus sharing the same L1-Data cache. The Victim VM is shipped with a dummy program which instantiates a “secret” string in memory and continuously access it, thus ensuring that the secret key is store in the L1-D cache. The attacker VM is shipped with the assessment tool, which tries to retrieve the secret key by exploiting the Foreshadow-VMM vulnerability.

Table 12: Foreshadow-VMM - Use Cases

CR1	Assess the presence of the vulnerability
Description	The Cloud owner assesses the presence of the vulnerability on its infrastructure.
Actors	Cloud Owner

Table 13: Foreshadow-VMM - Certification requirements

SR1	Linux OS support
Description	In the current state of the tool, it has been implemented only to support Linux environments.

Table 14: Foreshadow-VMM - Software requirements

3.1.5.2 Technical specifications

The Foreshadow-VMM Assesment tool consists of the following components:

- Victim VM. Materializes in memory the secret key to be stolen by the attacker.
- Attacker VM. Exploits the Foreshadow-VMM vulnerability to retrieve the secret key.
- Host kernel patch. Responsible for providing the Extend Page Table Pointer of the victim VM.
- Host Kernel module. Responsible for the translation of the guest physical address into the host physical address.

In order to be sure that the two machines are running on the same physical core, two VMs have been instantiated using Kernel-Based Virtual Machine (KVM), setting CPU affinities to force KVM to run the two VMs on the same core. We then provided an attacker VM with the following two tools:

- an **Offending Kernel Module** (OKM), triggering the terminal fault on the Probe.
- a **Probe**, a user space program devised to be offended by the OKM and to probe the cache for active lines.

At first, Probe is started. The kernel associates with such process with a Process ID (PID) and creates the relevant page table tree inside the kernel structure, namely the *mm* data structure. The Probe instantiates a temporary variable called *tmp*, which has a certain virtual address, thus creating a Page Table Entry for that variable.

The Probe will pass it's PID to the OKM using a character device, passing also the virtual address of the *tmp* variable in the *u64* format, needed by the Offending Kernel Mo1dule (OKM) to actually tamper the page table entry associated with the *tmp* variable.

At this point, the OKM can be loaded inside the attackers' VM and recives the PID of the Probe and the *vaddr* of the *tmp* variable. Starting from the PID, the OKM extracts the *mm* struct from the *task* struct, which is process-specific. With such information, the OKM can start a page walk to arrive to the leaf node of the tree, which contains the Page Table Entry (PTE) for the *tmp* variable of the Probe, which contains the translation between the virtual address and the physical address of the variable, aside with other control bits, such as the present bit of the PTE.

At this point, we are able to manipulate the page table entry, setting/clearing control bits and altering the translation between virtual and physical address. In particular, clearing the present bit on the *tmp* variable PTE will cause a terminal fault when the Probe will access again the *tmp* variable.

It is also possible now to tamper the translation, since the physical address is encoded inside the PTE from bit 47 down to 12. In fact, substituting this address with a malicious one, allows us to read any data residing in the L1D cache on the subsequent step.

When the Probe will access again the *tmp* variable, the kernel will perform a page walk, arriving on the tampered PTE. It will find that the PTE is not valid, since the present bit has been cleared, rising

an exception which will abruptly terminates the Probe after a certain time window. In the meanwhile, the access to the *tmp* variable has been passed to out-of-order execution with the tampered address, making a side effect on the cache based on data contained in such a location, which may belong to the Host OS or to the Victim VM. At this point, the Probe intercepts the exception and starts to read speculatively the content of the L1-data cache. At this point the attack can be automatized to retrieve any arbitrary string from the L1-D cache.

3.1.5.3 Development roadmap

Use Case	Architecture components	Realisation	Involved partners
UC1	Victim VM, Attacker VM, Host Kernel Patch, Host Kernel Module	Use the tool to asses the presence of the L1-TF Vulnerability	CNIT

Table 15: Foreshadow-VMM - Use cases, realisations and architecture

3.1.5.4 Software verification and validation plan

SR id	Description	Verification method	Demonstration scenario
SR1	The tool compiles in the current Kernel	Check if the Attacker VM is able to retrieve the secret key	Use the tool on the current infrastructure

Table 16: Foreshadow-VMM - Demo scenarios and verification methods

3.1.6 NeSSoS Risk Assessment Tool

Identifier	RA
Name	NeSSoS risk assessment tool
Owner	CNR
Main functions	The tool supports risk assessment of organisation's IT system. The tools is questionnaire-based and aims at providing quantitative results.
Description	This is an on-line tool that helps organisations to estimate their expected losses due to the possible cyber attacks of various kinds. The tool asks for the core assets and potential impact, once the assets are compromised. Next, the tool requests for the information about the security countermeasures installed and practices applied in the organisation. Then, it computes the expected losses for the organisation. The tool is available in three modes : "short" (with very small set of questions) for a quick estimation, "medium" for a more in depth analysis, and "complete" for in-depth assessment.
Assessment phases	The tool provides high level assessment, so it is important for Risk Management process at the global level.
Technologies required	The analysis targets IT networks (with possible use of cloud, mobile and teleworking)
URL	https://www.cybersecurityosservatorio.it/en/Services/survey.jsp
Documentation	N/A
Example usage	A responsible person for security of a system initiates the risk assessment process in order to ensure that all relevant threats are covered, estimate possible risks due to cyber threats, and plan further actions for treatment of residual risks. The process can be executed as by the security team of the owner, as well as by external auditors (as well as for certification purposes, e.g., to identify the main risks, identify required protection, and analyse the effectiveness of existing countermeasures and practices).

The tool will be used in order to evaluate existing risks for the e-government system.

See section 8 of the appendix for further details (user requirements, technical specifications and development roadmap) for this tool.

3.1.7 IDS and SIEM Assessment Tool (IMT)

Identifier	IDS
Name	IDS and SIEM assessment tool
Owner	Télécom SudParis (IMT)
Main functions	Network traffic stress testing
Description	The tool generates different kinds of traffic to fulfill different testing purposes: • synthetic legitimate traffic to provide a realistic testing environment without incurring privacy issues • synthetic legitimate/malicious traffic bootstrapping from existing traces with the ability to apply various transformations to the resulting traffic in order to generate a wealth of different traffics and increase the size of the dataset • synthetic adversarial traffic which has the ability to evade a detector Adversarial learning can also feed back to the security tool to improve its detection (IDS) and/or correlation ability (SIEM).
Assessment phases	Operations
Technologies required	Adversarial learning, Generative learning, Neural networks
URL	N/A
Documentation	N/A
Example usage	• Automatic noise generation for penetration testing • Results analysis to characterize IDS/SIEM robustness • Network traffic trace generation/amplification • Attack traffic mutation

In CAPE, IMT aims at improving the parser to extract new features that will enable a more faithful modelling of traffic traces. By reliably modelling real traffic traces, we believe that we will be able to generate more realistic network traffic. The models learned from a single traffic trace allows the generator to reproduce traffic for this particular trace. One particular challenge is the feasibility of producing real traffic features from certain model features.

The traffic generator takes as input the model features learned from the parser and generates a synthetic network traffic. A legacy and more heavy approach was to set up a number of agents supporting the protocols identified in the input and to have them generate a traffic envelope that fulfills the distribution of protocols and their respective amplitudes over time. While this work is interesting to pursue particularly with respect to agent orchestration, in CAPE, we aim at developing novel approaches based on generative networks. In particular, a first attempt using autoencoders – a type of neural networks that efficiently learn and reproduce inputs – has been studied. IMT will focus on improving the realism of generated traffic, with respect to packet contents and flow behaviour. Indeed, the autoencoder actually output a feature vector and not network traffic per se. IMT needs to develop

a function (translator) to generate life-like traffic from the outputted feature vector.

Secondly, in order to generate malicious vector able to challenge the systems under evaluation (IDS, SIEM), IMT will develop a generative adversarial network-based (GAN) approach. Using a GAN, we aim at improving concurrently two aspects of the generated traffic: its realism and its malice, so that it becomes difficult for the system under evaluation to discriminate real, legitimate traffic from the malicious, synthetic one.

Finally, a human interface module should summarize the results of the test and assist the tester in identifying the weaknesses of the system under evaluation to make recommendations on how to improve it.

See section 8 of the appendix for further details (user requirements, technical specifications and development roadmap) for this tool.

3.1.8 Risk Assessment for Cyberphysical Interconnected Infrastructures (MRA)

Identifier	MRA
Name	Risk assessment for cyberphysical interconnected infrastructures (MRA)
Owner	NCSRD
Main functions	The main function of the MRA is the identification and modelling of the cyber-physical interconnections of infrastructure assets. Firstly, a user inserts a threat scenario, which is the initiating point for an impact assessment accounting for cascading effects within and between interconnected infrastructures. MRA follows the ISO 27005 approach, and its novelty lies on the decomposition of both the likelihood and consequences elements of risk. The tool is customizable with respect to the impacts and their importance to the network operation.
Assessment phases	Requirement analysis
Technologies required	Java or Python development
URL	To be provided upon final assessment
Documentation	To be provided upon final assessment
Example usage	The security/IT department of a facility should engage the MRA process covering as many risks as possible to the infrastructure (or specific assets therein), and identify based on selected risk levels the most appropriate mitigation plan. The MRA tool has been applied in the risk assessment of a simple infrastructure comprising of interconnected assets (smart lights of the NCSRD facility). The smart lighting infrastructure has been subject to different types of attacks and possible impacts (both in the cyber and physical domains).
Continuing work from projects	The framework behind the tool has been used within the ISF funded project for CI Protection of national critical infrastructures, serving as a common basis for comparing different types of threats under a common approach.

3.1.8.1 User requirements description

This tool will quantify existing risks for vertical 1:

UC1	Cyber attack with cascade impacts
Description	Evaluation of cascading effects from cyber attack
Actors	• Safety / security manager (Risk experts) • IT (security) team
Basic flow	Once the characteristics, interconnections and processes of the infrastructure are defined, the user will identify cascading effects from a cyber attack that will disrupt the operation of an asset.
UC2	**Continuous risk quantification**
Description	Identification of attractive assets of infrastructure
Actors	• Safety / security manager (Risk experts) • IT (security) team

Table 17: MRA - Use Cases

SR1	MRA stand alone tool
Description	The tool is to be available as a stand alone service
Actors	• Developers (NCSRD) • Users (as above)
Basic flow	Make the tool available to the community

Table 18: MRA - Software requirements

3.1.8.2 Technical specifications

The MRA tool is build on the following components:

- A user interface that allows user to input information about the infrastructure, its assets and interconnections, their properties and potential vulnerabilities and other needed ancillary input.
- A modeling component that performs a cascade analysis of the assets and estimates risk
- A display element that transfer outputs to users
- A database storing all required / processed / produced information.

In brief, the tool work as described in the following lines. Users input enters the required inputs (infrastructure assets, properties, interconnections, safeguards, potential impacts), which are stored in the database. The software passes the data to the modeling component and identifies: a) Potential threats, b) Attractive assets, c) Interconnections and potential cascade effects, d) Impacts (in the cyber and physical domains), e) risk.

This information is fed back to the user through the display element. If the tool can be extended for continuous risk assessment, the interfaces need to be customized to allow inputs from a machine-readable format

3.1.8.3 Development roadmap

Use Case	Architecture components	Realisation	Involved partners
UC1	Interface, database, modeling component, display	Use the tool to define risks in cyberphysical systems	NCSRD
UC2	Risk modeling component	Use the tool and define attractive assets as targets	NCSRD

Table 19: MRA - development roadmap

3.1.8.4 Software verification and validation plan

SR id	Description	Verification method	Demonstration scenario
SR1	Stand-alone tool	Check if the tool is available online and provides the risk levels. Use the tool through the web interface	

Table 20: MRA - verification and validation plan

3.1.9 Steady (SAP)

Identifier	VA
Name	Eclipse Steady (incubator project)
Owner	SAP
Main functions	Discover, assess and mitigate open source components with known vulnerabilities in Java and Python projects.
Description	The open-source vulnerability assessment tool supports software development organizations in regards to the secure use of open-source components during the development of Java and Python applications. As such, it addresses the OWASP Top 10 security risk [89] "Using Components with Known Vulnerabilities", which is often the root cause of data breaches.
Assessment phases	Application development
Technologies required	Java or Python development project with source code and open source dependencies (either declared according to common dependency managers of the respective prg. language, or available in file system)
URL	• https://github.com/SAP/vulnerability-assessment-tool • https://projects.eclipse.org/projects/technology.steady
Documentation	• User manual: https://sap.github.io/vulnerability-assessment-tool/ • Tool paper: https://arxiv.org/abs/1806.05893
Example usage	• Detect whether a Java or Python application depends on open-source components with known vulnerabilities • Collect evidence regarding the execution of vulnerable code in a given application context (through the combination of static and dynamic analysis techniques) • Understand and choose the best non-vulnerable version of a dependency affected by a vulnerability

3.1.9.1 User requirements description

The main high-level functionality of Steady is to support the detection of open source dependencies with known vulnerabilities. The two use cases below differ in regards to the actor performing the analysis and the targeted component. Those use cases are largely independent of a given industry or vertical and their specific security and certification requirements.

UC1	**Detect, assess and mitigate dependencies with known vulnerabilities *in application projects***
Description	The developer of a given application uses Steady to scan all application dependencies, either manually or as part of automated build processes. Typically, this is done using one of the available plugins for different build tools such as Maven [1]. Then, he uses the Web frontend to understand and assess findings and find the best mitigation option.
Actors	Developer (in the context of a given application)
Basic flow	The analysis of application dependencies is triggered either manually, or in some automated fashion, e.g., through a periodic or commit-trigger build job. Analysis results are made available through Steady's Web front-end or by producing a report accessible to developers on the respective build system. Additionally, it is possible to break build jobs in order to signal developers that vulnerable dependencies have been found.
UC2	**Detect dependencies with known vulnerabilities *in open source projects* and suggest mitigations**
Description	An organization uses Steady to scan open source projects that are of particular importance to the entire organization, e.g., because many of the organization's applications depend on them. Findings (and fix suggestions) are communicated to the respective open source project maintainers.
Actors	Developer (outside the context of a given application)
Basic flow	See UC1

Table 21: Steady - Use Cases

SR1	Comparison of Java source code and bytecode with intermediate representation
Description	Steady follows a code-based approach to identify whether given packages contain programming constructs that are subject to known vulnerabilities. In case of Java, for instance, it checks whether the signature of a vulnerable Java method is contained in the dependencies of an application. In a next step, it has to determine whether the method body equals (or is closer to) the vulnerable version or the fixed version of the respective method, according to versioning information gathered from the open source project's VCS. So far, Steady uses different techniques and heuristics, each one having certain limitations. The requirement SR1 is to develop a new approach based on abstract code representations that can be built from Java source and bytecode, e.g., Jimple [2].
SR2	**Implementation of a light-weight scan client**
Description	The current architecture of Steady requires users to operate a server-side backend (cf. Figure 31). While this is acceptable for large software development organizations, it hinders adoption by individuals. This requirement SR2 is to develop a light-weight scan client that consumes information from a shared vulnerability database.
SR3	**Shared vulnerability database**
Description	The current architecture of Steady requires users to download commit information for open source components from a Git repository in order to populate a server-side vulnerability database. Today, this process is partly manual, which represents an obstacle to continuous and automated exchange and synchronization of vulnerability information. This requirement SR3 is to develop a new structure and tooling to improve the maintenance and exchange of vulnerability information in a shared Git repository, in conjunction with the light-weight scan client mentioned in SR2.

Table 22: Steady - Software requirements

3.1.9.2 Technical specifications

At high-level, cf. Figure 31, Steady comprises a number of client-side scan tools that analyze a given application, either manually or as part of automated build processes (`plugin-maven`, `plugin-gradle`, `cli-scanner`). Analysis results are uploaded to (and persisted by) a RESTful component called `rest-backend`, which is one out of several components that run server-side, e.g., in private or public clouds. The components `frontend-apps` and `frontend-bugs` are Html5 applications rendered by a browser, and used by end-users to consume the analysis results. The remaining components, `patch-analyzer` and `rest-lib-utils` are related to the analysis and processing of commit information (of open source projects) and packages available on public or private package repositories.

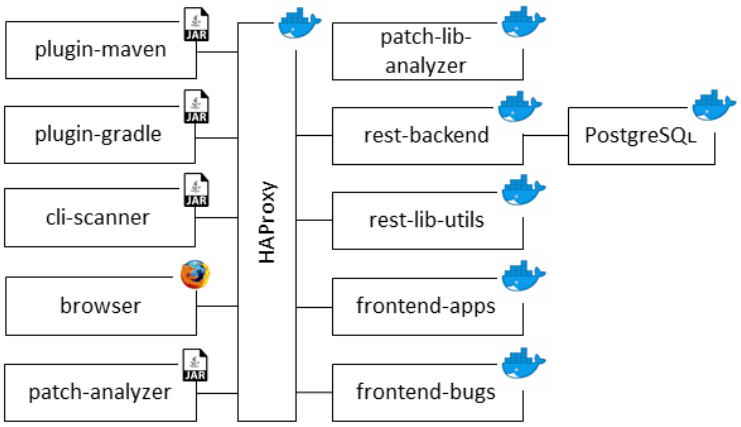


Figure 31: High-level architecture of Steady

3.1.9.3 Development roadmap

The high-level development roadmap is to implement the software requirements SR1-SR3 within 2020-21 such that it can be demonstrated at project end as explained in Table 23.

3.1.9.4 Software verification and validation plan

SR id	Description	Verification method	Demonstration scenario
UC1	Scan app(s) related to e-government vertical	Check if scans succeed and findings are correct	e-government
UC2	Scan open source projects	Check if scans succeed and findings are correct	Not applicable

Table 23: VA - Demo scenarios and verification methods

3.1.10 Package Scanner (SAP)

Identifier	PS
Name	Package Scanner
Owner	SAP
Main functions	Detect whether a package, as downloaded from package repositories such as PyPI, contains malicious code
Description	This tool aims to detect malicious code in software packages that are distributed by package repositories such as PyPI for Python. It mitigates a number of attack vectors that aim at injecting malicious code during the build process or in the package repository itself (cf. attack tree in Section 3.3.2.2).
Assessment phases	Application development
Technologies required	Python
URL	Not yet available
Documentation	Not yet available
Example usage	• Detect whether any of the dependencies of a given Python application contains malicious code. • Scan entire package repositories.

3.1.10.1 User requirements description

UC1	**Scan a given open source package**
Description	The tool takes as input a Python package of given open source project and checks whether it contains malicious code that is not present in the respective VCS
Actors	Developer
Basic flow	The developer uses the command line interface of the tool in order to trigger the scan of a package, e.g., one residing in the file system or downloadable from a given URL such as the PyPI package repository. Analysis results are written into a report.

Table 24: Package Scanner - Use Cases

3.1.10.2 Technical specifications

Figure 32 visualizes the communication of the Package Scanner with the user and other system components, esp. the package repository from which the distributed package is obtained.

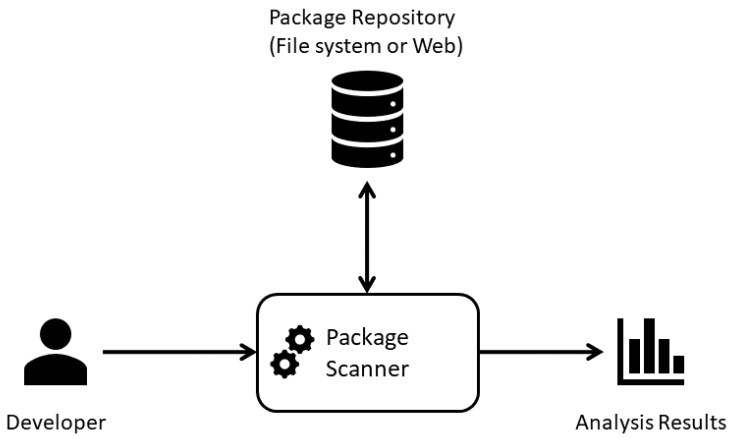


Figure 32: High-level architecture of Package Scanner

3.1.10.3 Development roadmap

The high-level development roadmap is to implement the Package Scanner within 2020 such that it can be demonstrated at project end as explained in Table 25.

3.1.10.4 Software verification and validation plan

SR id	Description	Verification method	Demonstration scenario
UC1	Scan Python packages (known malicious and unknown)	Check if known malicious are found, manual review of findings for unknown packages	Not applicable

Table 25: PS - Demo scenarios and verification methods

3.1.11 OpenCert (TEC)

Identifier	OC
Name	OpenCert
Owner	TEC
Main functions	OpenCert is an open product and process assurance/certification management tool to support the compliance assessment and certification of Cyber-Physical Systems (CPS) spanning the largest safety and security-critical industrial markets, such as aerospace, space, railway, manufacturing, energy and health.
Description	OpenCert supports a number of features, including the cross/intra-domain reuse of assurance assets and the semi-automated construction of assurance cases. • Standards & Regulations Information Management: This activity group supports knowledge management about standards (e.g. DO178C, ISO26262, EN 50128/50126/50129, etc.), regulations and interpretations, in a form that can be stored, retrieved, categorized, associated, searched and browsed. • Assurance “Project” Management: This is the core set of functionalities concerned with the development of assurance cases, evidence management, assurance process management, and global monitoring of the compliance with standards and regulations. The most relevant services of the OpenCert tool are to provide functionality that supports guidance and re-use of assurance artefacts. In addition, these services offer an evolutionary and transparent product and process assurance and certification with the ability to automate the most labour-intensive activities (e.g., traceability, compliance checking, assurance process planning, and metrics management, among others), as well as providing facilities to integrate the engineering activities with the certification activities from early stages. • Compliance Management: The OpenCert tool helps “engineers” to assess where they are with respect to their duties to conform to safety practices and standards, and still to motivate them to see the effective progress of the work and level of compliance. • Modular and Incremental Certification: OpenCert supports a modular safety assurance and certification approach to enable cost-effective reuse of pre-qualified building blocks in different contexts (e.g., systems, configurations, upgrades).
Assessment phases	Within the V-Model for the Safety Engineering process (Figure 25), the OpenCert tool supports the phases of ”Safety Goals definition” and ”Safety Goals validation”.
Technologies required	JDK 1.8, PostgreSQL9.3, Windows 64 bits

URL	Polarsys Repository: https://www.polarsys.org/projects/polarsys.opencert
Documentation	The OpenCert tool is one of the outputs of the AMASS ECSEL project [13] • Website: https://www.polarsys.org/opencert/ • AMASS Platform User Manual [12] • AMASS Platform Developers' Guide [11]
Example usage	• Automotive domain: The OpenCert tool has been used in the framework the OPENCOSS [88], AMASS [13] and AQUAS [16] projects, to manage assurance and certification management of systems with standards such as J3061 and ISO 26262. • Industrial automation domain: Managing compliance with IEC 61508, IEC 62443 and IEC 62351 standards.

3.1.11.1 User requirements description

UC1	Support the Safety and Security compliance assessment and certification of the platooning scenario
Description	Standards, regulations, certification advisory circulars and so on are managed by OpenCert as Reference Frameworks. A Reference Framework includes project-independent information that can be (re)used by various projects, e.g. models of standards or generic processes. Threfore, the first activity to be executed in UC1 is to digitalize each standard that the platooning has to be compliant with (ISO 26262 and SAE J3061). The Process Engineer role in charge of this activity should be an expert in the standards, compliance and certification processes. This responsibility could also be assigned to a Safety Manager who creates the Reference Framework and models the knowledge coming from a the functional safety standard ISO 26262 and a Security Manager models the SAE J3061. The Reference Framework(s) contains: • Reference Standards' requirements to fulfil • Reference Activities to execute, and • Reference Artefacts to manage. The next step to achieve is the Assurance Project creation. The Assurance Manager or Process Engineer can maintain the lifecycle of projects by creating Assurance Projects, they should have knowledge of the standards as well as a safety and/or security background. For the platonning scenario, at least one Assurance project will be created. During the Assurance Project creation fase, the Compliance Baseline is also defined. The Baselines are subsets of Standards to be applied in a specific Assurance Project. Baseline Models are created by tailoring of Reference Frameworks, i.e. by importing a Reference Framework model and selecting elements for the current project. Thus, a Baseline defines the elements of a Reference Framework model that have to be applied to the current Assurance Project. For the platonning scenario two Baselines should be necessary, one based in the ISO 26262 and other in the SAE J3061. The following is the Evidence Management, that includes the collection and handling of the assurance evidences of an Assurance Project, done by the Assurance Manager, Assurance Engineer and Process Engineer actors. When managing Assurance Evidences, the first step is usually to determine what evidences must be provided. Afterwards, the Evidence Artefacts must be collected and might also have to be evaluated and traced to other Artefacts. Finally, the Assurance Manager, Process Engineer, Assurance Engineer and Systems Engineer actors are responsible to argue the safety/security of the platooning in an Assurance Case to resolve the safety/security trade-off.

Table 26: OpenCert - Use Cases (1/2)

Actors	• Assurance Manager(Safety Manager, Security Manager) • Process Engineer • Assurance Engineer (Safety Engineer, Security Engineer, V&V Engineer) • Systems Engineer
Basic flow	• Digitalize the applicable Safety standards in the OpenCert module • Digitalize the applicable Security standards in the OpenCert module • Create the Assurance Project that will allow to manage the Safety/Security certification • Evidence Management • Create Argumentation (Safety Case) ⇒ Co-assessment

Table 27: OpenCert - Use Cases (2/2)

UC1 will be specified in the D5.2 *Demonstrators specifications* deliverable and implemented in the D5.3 *Demonstrator prototypes* deliverable.

3.1.11.2 Development roadmap

OpenCert will be used in its current version and with its current functionalities in Vertical 1, i.e. it is not necessary to tackle new developments in order to comply with the use case UC1 described in table 26.

3.1.12 Sabotage (TEC)

Identifier	SB
Name	Sabotage
Owner	TEC
Main functions	Sabotage is model-driven and simulation-based fault injection tool to accomplish an early evaluation dependability evaluation of safety-critical systems. It can be used in different areas such as automotive or robotics and it has been built upon the FARM [17] model.
Description	The Sabotage tool can be used in an early assessment of safety-critical systems. It is a tool based on the Simulation fault injection technique which involves the construction of a simulation model (Simulink) of the system under analysis. Thanks to this simulated system the verification and validation is achieved during its early development phases. Model-driven and simulation-based fault injection tool allow to accomplish an early evaluation dependability evaluation of safety-critical systems. The framework sets up, configures, executes and analyses the simulation results. It includes a fault model library and it is possible to connect to virtual environments such as a virtual vehicle or a robot. The Sabotage tool is based on Eclipse combined with Matlab/Simulink. It includes a Fault Model library constituted as C templates and which are integrated in Matlab/Simulink as S-function blocks. The safety engineer starts configuring the fault injection experiments by creating the fault injection policy or fault list (*Where should the faults be injected? What is the most appropriate fault model representing the functional failure modes? How should the faults be triggered within the system? Where should the fault effect be observed?*). Then the faulty model is created, and the fault free simulation compared to the faulty ones. This means running, storing, visualizing and computing the obtained simulation traces, while comparing the results versus a pre-established safety requirement or pass/fail criterion.
Assessment phases	Within the V-Model for the Safety Engineering process (Figure 25), the Sabotage tool supports the phases of ”Functional and Technical safety concept design” and ”Functional and Technical safety concept verification”.
Technologies required	• Java SE 1.8 • MATLAB/Simulink from 2017b version.

URL	N/A
Documentation	• Sabotage: A Simulation-Based Fault Injection Tool Framework [102] • AMASS D3.3 deliverable, pages 57-62 [10]
Example usage	• Automotive domain: The feasibility of the fault injection approach has been demonstrated in the AMASS project [13] by applying it to a Lateral Control system, an ACC (Adapted Cruise Control) and a DC (Direct Current) Motor controller. • Robotics domain: The feasibility of the fault injection approach and model-based techniques for safety analysis have been applied in terms of a robotic arm manipulator in the eITUS project [38]. Fault injection has been combined with the Gazebo simulator in order to perform an early safety validation or robustness simulation.
Continuing work from projects	The tool has been developed as part of the AMASS (Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems) (https://www.amass-ecsel.eu/) and eITUS (https://robmosys.eu/e-itus/) (RobMoSys ITP) projects"

3.1.12.1 User requirements description

UC1	**Fault-injection and analysis of faulty scenarios with simulation**
Description	Sabotage will be used to simulate how a cyber-attack can affect the vehicle motor by changing the velocity to an abnormal value, so faults will be injected in the behavioural model (Simulink model) of the longitudinal control function.
Actors	Safety Engineer
Basic flow	The following steps would be followed: 1. Model the system architecture 2. Perform Simulation-based Fault Injection 3. Perform Model-based Safety Analysis 4. Define the safety concept/safety mechanisms 5. Perform Simulation-based Fault Injection 6. Check if the safety mechanisms are correctly implemented and a sufficient level of safety has been achieved.If not go to step 4.

Table 28: Sabotage - Use Cases

UC1 will be specified in the D5.2 *Demonstrators specifications* deliverable and implemented in the D5.3 *Demonstrator prototypes* deliverable.

Sabotage will be used in its current version and with its current functionalities in Vertical 1, i.e. it is not necessary to tackle new developments in order to comply with the use case UC1.

3.1.13 Visual Investigation of Security Information for Larger Software Development Organizations (UKON)

Identifier	VI
Name	Visual investigation of security information for larger software development organizations (WP5 (2-3))
Owner	University of Konstanz (UKON)
Main functions	The main goal is to enable the visual investigation of the security status of individual software components and the assessment of the associated risk posed by own and third-party components through the following functionalities: (1) Getting an overview of the current system and status using visually salable visualizations (hierarchies). (2) Security information has multiple sources and levels (file, component, organization). (3) Data collection and presentation of the security status of modules and properties. (4) Enable to explore vulnerability results. (5) Assessment of risk on a software component level, esp. for third party dependencies. (6) Show structural dependencies of projects and divisions on own and third-party software components.
Description	Visually investigate the security status of software projects. Explore the security status of own packages, whereby status reflects the use of tools, the number, and severity of findings as well as assessments. Understand organization-wide dependency on components, used as input to harmonize component and version used, and to decide whether important components deserve support.
Assessment phases	Optimize resources to reduce exposure and risk management
Technologies required	The tool will be developed as a web tool and will work with technologies such as Java, Python, and JavaScript.

3.1.13.1 User requirements description

UC1	**Visual Investigation of Large Software Organizations**
Description	The automatically detected known vulnerabilities in large software organizations such as the Eclipse Foundation are presented and explored.
Actors	• Software Developers • Project Managers
Basic flow	A software developer or project owner provides a project; the tool then depicts automatically show detected known vulnerabilities in the component, in the dependencies to internally developed packages, as well as external third-party libraries.

Table 29: VI - Use Cases

CR1	Increase confidence in analyzed systems
Description	The system should increase the confidence in the security of software systems by (a) enabling the presentation and exploration of vulnerability results (b) showing exposure to other projects including internal and external (up-stream and down-stream) dependencies.
Actors	• Software Developer • Project owners • Managers
CR2	**Multi-source levels of analysis**
Description	The tool should be able to support multiple source levels of analysis, i.e., component, project organization level to show the impact of vulnerabilities on different levels in the software organization.
Actors	• Software Developer • Project owners • Managers
CR3	**Information representation**
Description	The shown information should be encoded in simple visual metaphors (i.e., histograms or treemaps).
Actors	• Software Developer • Project owners • Managers
CR4	**Vulnerability prioritization**
Description	The system should support the prioritization of vulnerabilities, to ensure that most urgent issues are addressed first.
Actors	• Software Developer • Project owners • Managers
CR5	**Interdependence analysis**
Description	The tool should allow for the analysis of interdependence in horizontally (i.e., collaborating organizations) and vertically (i.e., projects using components).
Actors	• Software Developer • Project owners • Managers

Table 30: VI - Certification requirements

SR1	Web Application Prototype
Description	The developed prototype should be a web prototype and enable the visual investigation and exploration of vulnerabilities.
Actors	• Software developers • Project owners
Basic flow	Investigate vulnerabilities in developed internal and external components (packages)

Table 31: VI - Software requirements

3.1.13.2 Technical specifications

The developed prototype will build upon the Eclipse Steady project and use other web technologies (e.g., JavaScript). The tool will have a backend developed in Python.

3.1.13.3 Development roadmap

Use Case	Architecture components	Realisation	Involved partners
UC1	Interactive visualization prototype and Eclipse Steady	Build the prototype to get the vulnerabilities information from Eclipse Steady	SAP, Fortiss

Table 32: VI - Use cases, realisations and architecture

3.1.13.4 Software verification and validation plan

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 33: VI - Demo scenarios and verification methods

3.1.14 Logic Bomb Detection in Android Apps (UniLu)

Identifier	LBD
Name	TSOpen (LBD)
Owner	UniLu
Main functions	Detection of logic bombs in Android apps by leveraging static analysis methods.
Description	Logic bombs are mechanisms used by malicious apps to evade detection techniques. Typically, an attacker uses 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 goal of TSOpen is to detect such logic bombs. The approach used to perform the detection is fully static and combine multiple techniques such as symbolic execution, path predicate reconstruction, path predicate minimization, and inter-procedural control-dependency analysis. In a first version, TSOpen will focus on detecting triggers related to time, location and SMS.
Assessment phases	Development Process (to check that a library used by the app under-development does not contain logic bomb). Operation (to check that a given app does not contain any logic bomb that could hide the presence of malicious behavior).
Technologies required	Android app Analysis (the source code is not required). The analysis is directly performed on the apk file of the application.
URL	TBD
Documentation	TBD
Example usage	Malware detection

3.1.14.1 User requirements description

UC1	**Detecting hidden malicious code.**
Description	Malware tends to use logic bombs in order to bypass dynamic analyses.
Actors	Antivirus company
Basic flow	In order to detect malicious applications, antivirus companies use multiple techniques and filters. TSOpen could be used as a filter layer to rule out applications hiding malicious behavior.

Table 34: TSOpen - Use Cases

SR1	A standalone command line tool.
Description	The tool is available as a command line tool.
Actors	Software analyst
Basic flow	Execution of the tool. Analysis of the results.
SR2	**Trigger database**
Description	This requirement is to develop a database of applications known to contain logic bombs as well as benign applications containing similar behavior.
Actors	Software analyst
Basic flow	Connect to the database. Query the database.

Table 35: TSOpen - Software requirements

3.1.14.2 Technical specifications

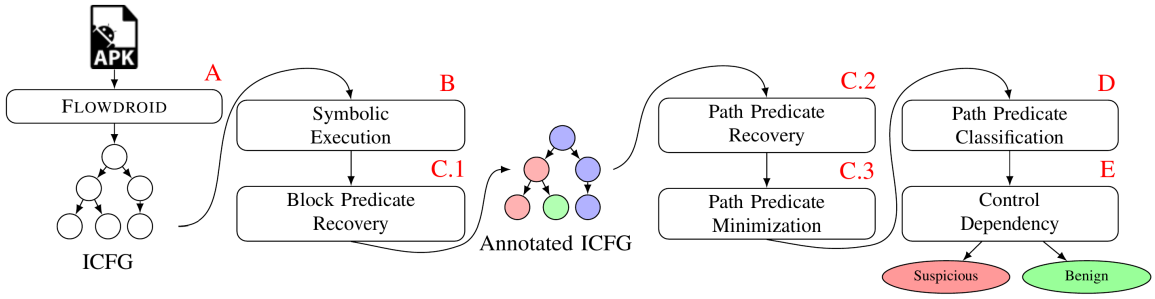


Figure 33: Overview of TSOpen

TSOpen is developed over Flowdroid which provides a useful model of the Android Framework on which one can easily apply algorithms. Figure 33 provides an overview of the tool. First, an inter-procedural control flow graph from Flowdroid is retrieved on which TSOpen applies a symbolic execution in order to retrieve the semantic of objects of interest. Then simple predicates are retrieved during the block predicate recovery to annotate the ICFG. The annotated ICFG is then used to retrieve the full path predicate of every instructions. A predicate minimization algorithm is then applied in order to rule out false dependencies. Afterwards, a first decision is taken during the predicate classification step to get suspicious predicates. Finally, a control dependency step is applied in order to take the decision regarding the suspiciousness of the potential logic bomb under study. The TSOpen tool consists of a standalone executable Java archive file (jar). It has to be executed with the command line or in scripts.

3.1.14.3 Development roadmap

Use Case	Architecture components	Realisation	Involved partners
UC1	Detection of hidden malicious code	Build the prototype to detect logic bombs	N/A

Table 36: TSOpen - Use cases, realisations and architecture

3.1.14.4 Software verification and validation plan

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 command line
SR2	Trigger database	Check if the database contains correct triggers	Connect to the database

Table 37: TSOpen - Demo scenarios and verification methods

3.1.15 Vulnerability Detection Tool For DevOps Communities (UniLu)

Identifier	VA2
Name	SafeCommit (not definitive yet)
Owner	UniLu
Main functions	Detect security relevant commits (also referred as patches for sake of simplification) in Continuous Integration Ecosystem. in particular: • Detection of commits introducing vulnerabilities • Detection of commits fixing vulnerabilities
Description	The goal of this tool is twofold: • First, detection of patches which fix software vulnerabilities. To that end, both code and textual features will be engineered and assessed. These features will be then used by machine learning algorithms designed and selected to cope with unbalanced datasets. • Second, detection of patches which introduce software vulnerabilities. Like mentioned previously, code and textual features will be investigated, but it is highly probable that the features are different. 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.
Assessment phases	Software Development
Technologies required	Repository mining, Machine Learning. This tool will work on C and Java code.
URL	TBD
Documentation	TBD
Example usage	Just before committing their code in a version control repository, developers can check their code to ensure they are not introducing a vulnerability.

3.1.15.1 User requirements description

UC1	Vulnerability Introducing Commit/Patch
Description	In this use case, SafeCommit is used to detect commits/patches that introduce vulnerabilities into a code base.
Actors	Software Developers
Basic flow	Just before committing their modifications (i.e. a commit) into a code base (i.e., a version control repository such as GIT), developers can check if their modifications introduce a vulnerability. In this way, SafeCommit allows to avoid the introduction of vulnerabilities at the very early stage of software development.
UC2	**Vulnerability Fixing Commit/Patch**
Description	In this use case, SafeCommit is used to detect commits/patches that fix vulnerabilities into a code base.
Actors	Software Developers/Maintainers
Basic flow	In a typical scenario of vulnerability correction, a developer proposes changes bundled as a software *patch* by pushing a *commit* (i.e., patch + description of changes) which is analyzed by the project maintainer, or a chain of maintainers, who eventually reject or apply the changes to the master branch. When the patch is accepted and released, all users of the relevant code must apply it to limit their exposure to attacks. The reality, however, is that, for most organizations, there is a lag between a patch release and its application. While in the cases of critical systems, maintainers are hesitant to deploy updates that will hinder operations with downtime, in many other cases, the lag is due to the fact that the proposed change has not been properly advertised as *security-relevant*, and is not thus viewed as critical. With SafeCommit, maintainers will be immediately inform that a security relevant commit has been performed. As a result, SafeCommit will incite maintainers to quickly propagate or reject the relevant changes; and also to notify end users and third-party developers so that they can update their outdated releases.

Table 38: SafeCommit - Use Cases

3.1.15.2 Technical specifications

Note that SafeCommit will be developed in the course of the Sparta project.

SafeCommit will use a machine-learning based approach. In particular, SafeCommit will address a **binary classification problem** of distinguishing security patches[3] from other patches. As any classification problem, well-labeled datasets are more than welcome. To develop SafeCommit, the first main step will consist in building such datasets (module 1 in Figure 34). 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. Then, we will build prediction models using machine learning classification techniques. One model to identify vulnerability inducing patches (Module 2), and one model to detect vulnerability fixing patches (Module

[3]Either patches fixing vulnerabilities or patches introducing vulnerabilities.

3). In particular, 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 learnt, we will assess their efficiency by performing extensive empirical studies in real development environment.

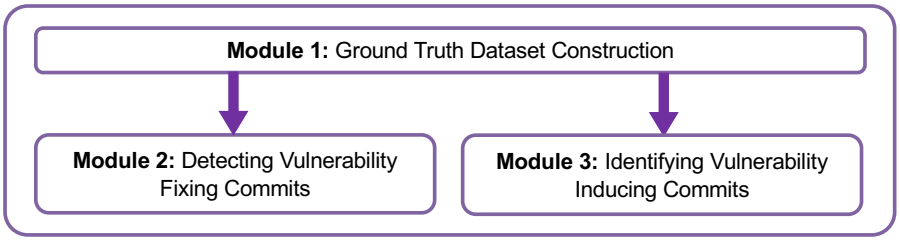


Figure 34: Various Modules of SafeCommits

3.1.15.3 Development roadmap

Use Case	Architecture components	Realisation	Involved partners
UC1	Modules 1 & 2	Build a ground truth dataset, propose features from code and text, build the prediction models	UniLu
UC2	Modules 1 & 3	Build a ground truth dataset, propose features from code and text, build the prediction models	UniLu

Table 39: SafeCommit - Use cases, realisations and architecture

3.1.15.4 Software verification and validation plan

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 40: SafeCommit - Demo scenarios and verification methods

3.1.16 AutoFOCUS3 (FTS)

Identifier	AF3
Name	AutoFOCUS3
Owner	FTS
Main functions	Requirement Engineering, Safety Analysis, Security Analysis, Modelling Tools, Code Generation, Verification
Description	A Model-Based Engineering Tool.
Assessment phases	Development Process
Technologies required	AutoFOCUS can be downloaded as a stand-alone application. It is part of Eclipse Foundation.
URL	https://af3.fortiss.org/
Documentation	https://af3.fortiss.org/docs/
Example usage	• Glossary; • Safety Analysis using Goal Structure Notation Models; • Security Analysis using Attack Defense Tree Models; • Textual and Structured Requirements Engineering; • Architecture modeling using hierarchical component structure; • Design Exploration Methods; • Requirements traceability; • Executable semantics; • Automatic Code Generation; • Hardware Deployment Mapping; • Code deployment.

3.1.16.1 User requirements description

UC1	Support the Safety and Security compliance assessment and certification of the platooning scenario
Description	Safety and Security analysis can be specified in AutoFOCUS in the form of GSN and Attack Defense Models. These analyses can be quantified in AutoFOCUS and help understand which counter-measures shall be deployed.
Actors	Safety and Security Engineers
UC2	**Architecture Modeling for Vertical 1**
Description	The architecture of rovers used by the FTS lab considered for Vertical 1 has already been described in AutoFOCUS. However, no security counter-measures have been modeled. The AutoFOCUS model can be used for further security analyses of Vertical 1.
Actors	• Architect Engineer • Security Engineer

Table 41: AutoFOCUS3 - Use Cases

UR1	Certifications, such as those used by the automotive industry, e.g., ISO 26262, have been taken into account in several projects involving AutoFOCUS.
Description	The V-model described above can be accommodated in AutoFOCUS.

Table 42: AutoFOCUS3 - User Requirements

AutoFOCUS is a stand-alone tool. Most of the features work without the need to install additional tools. For more details of the machine requirements we refer the reader to the AutoFOCUS site https://af3.fortiss.org/docs/.

3.1.16.2 Technical specifications

AutoFOCUS is organized into several Java Plug-ins (more than 20). Each plug-in is responsible for some particular AutoFOCUS feature. For the project, we are developing the AutoFOCUS security plug-in. It contains features, such as threat analyses using Attack Defense Trees, and algorithms for extracting security relevant information from safety analysis.

3.1.16.3 Development roadmap

We will develop the following functionalities:

- Algorithms for the automated construction of Attack Defense Trees from GSN Models;
- Algorithms for safety and security trade-off analyses;
- Algorithms for the quantitative evaluation of combined safety and security assessments.

3.1.16.4 Software verification and validation plan

We are going to validate AutoFOCUS using the use-cases developed in Vertical 1.

- Modeling HARA and TARA of the platooning scenarios using, respectively, Attack Trees and GSN models;
- Complement existing Attack Trees by using the developed algorithms for extracting security relevant information from GSN Models;
- Carry out trade-off analyses involving the proposed counter-measures and control-measures;
- Infer the confidence on the combined safety and security assessments based on the trade-off analyses.

3.1.17 Buildwatch (UBO)

Identifier	BW
Name	Buildwatch
Owner	UBO
Main functions	Monitor the build, test, or install process on an operation system call level.
Description	During the build, test, and install processes within application development, changes are made to the state of the host. While malicious code is often heavily obfuscated, the changes to the host system are of more limited variability and therefore, are easier to comprehend. *Buildwatch* provides a sandbox to monitor these changes.
Assessment phases	Application development
Technologies required	A process to monitor (e.g. build script, test suite, installation) that is built on top of GNU/Linux.
URL	N/A
Documentation	N/A
Example usage	• Infer host state changes during development processes (i.e. build, test, install). • Infer differences in form of host state changes for specific code changes contained by a commit or a version of the project, e.g. when updating a dependency.

In order to use the Buildwatch Sandbox in a continuous integration pipeline supported development process,additional required interfaces have to be added.

See section 8 of the appendix for further details (user requirements, technical specifications and development roadmap) for this tool.

3.1.18 VaCSInE (CETIC)

Identifier	VCS
Name	VaCSInE
Owner	CETIC
Main functions	Adaptive continuous security orchestration in polymorphous environments
Description	Ensure security of systems based on policies, continuous monitoring, and assessing security (certification) requirements in Cloud-Edge-IoT network environments
Assessment phases	Operations
Technologies required	streaming log analysis, Kubernetes, K3S, KubeEdge, Network Function Virtualisation (NFV)/Service Function Chaining (SFC) tooling, TOSCA, ARMv8
URL	https://github.com/cetic/vacsine
Documentation	https://github.com/cetic/vacsine
Example usage	Intrusion detection triggers remediation (e.g. firewall re-configuration), remediation is checked against the system's security policy (derived from certification criteria) and applied to the edge infrastructure (industrial systems, 5g, connected cars, ...).
Continuing work from projects	BEACON - Enabling Federated Cloud Networking - Horizon 2020 [20]

In CAPE, VaCSInE will demonstrate how to ensure continuous assessment of edge systems by developing adaptative security mechanisms based on security policies derived from certification requirements.

See section 8 of the appendix for further details (user requirements, technical specifications and development roadmap) for this tool.

3.1.19 Continuous Integration of Assessment Tools

In the recent years, the need to improve software delivery in terms of speed and quality has given rise to a set of practices that combine continuous build, testing, integration, delivery, ... The *DevOps* approach, closely related to Agile software development method, combines software development ("Dev") and operations ("Ops") processes to ensure that new features are added to a software solution in the shortest time possible, and with a high level of quality. Various services will be needed to support the DevOps aspects of the CAPE framework:

- a *version control system (VCS)* for the framework tools if they do not have one already or as a mirror of the original tool source code repository
- *artifact repositories* to store the various tools artifacts (binaries)
- *Continuous Integration and Continuous Delivery (CI/CD) orchestrator* to coordinate the various deployment and integration operations of the demonstrations scenarios
- *environments* for integration and testing of the framework: this can take the form of virtualised infrastructure or actual physical testbeds such as model connected cars or electronic ID card reader
- *reporting tools* to ensure observability of the framework: dashboards, alerts, logging, ...

Figure 35 provides example tooling alternatives for a typical self-hosted DevOps pipeline. Gitlab [4] for instance provide other services than only SCM: CI/CD with Gitlab-CI. Other tools like Jenkins [5] is focused on CI/CD, Gatling[6] on load testing, Jest[7] on UI testing, Jfrog Artifactory[8] provides artifact repository service, private infrastructure as a service solutions such as Kubernetes[9] have built-in integration with DevOps tooling, and finally observability can be implemented with the help of monitoring platforms such as Zabbix[10].

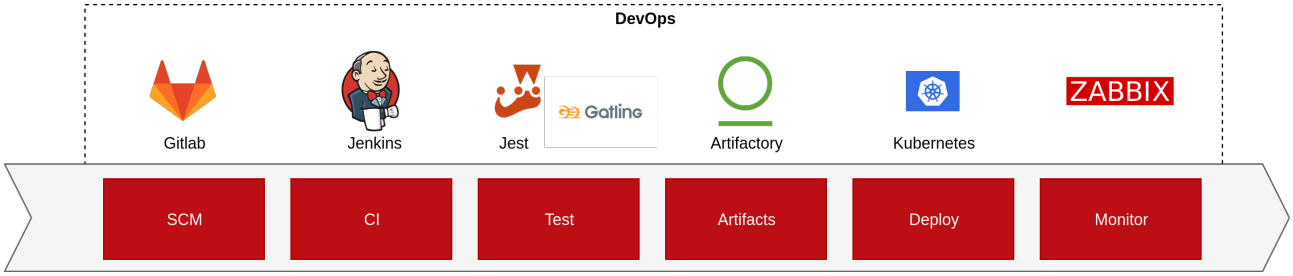


Figure 35: DevOps pipeline sample tooling

The DevOps approach itself can be complemented with security procedures to ensure continuous security assessment, this is sometimes called *DevSecOps*. Figure 36 provide example tooling alternatives for a typical DevSecOps pipeline [11]. In this example, Sonarqube provides static code analysis [12], Splunk[13] and OpenSCAP[14] cover the orchestration of continuous security assessment activities, OpenVAS[15] can scan for vulnerabilities, Kube-Bench[16] checks if Kubernetes deployments meet the CIS (Center for Internet Security) security benchmarks. The Elastic SIEM[17] offering for security analytics by Elasticsearch

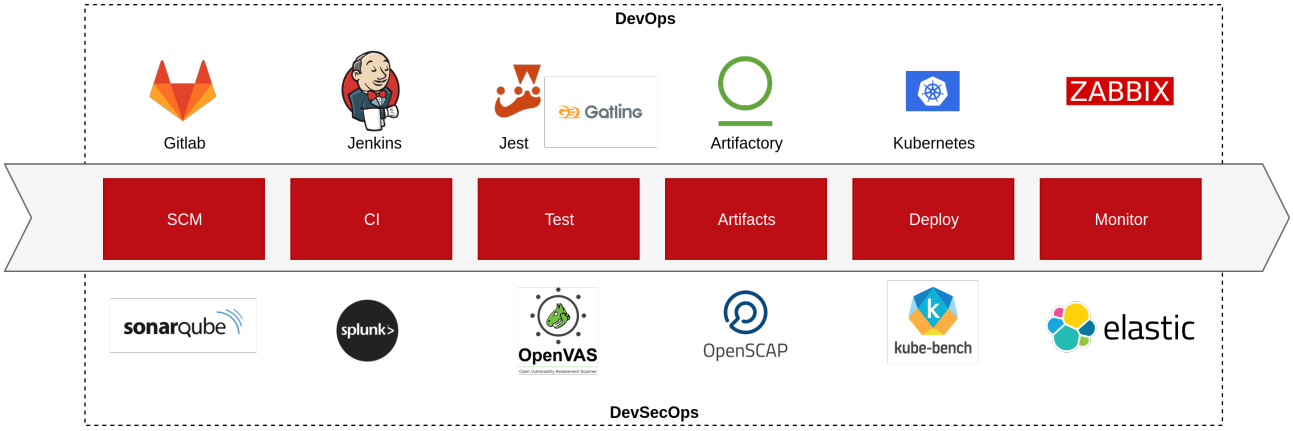


Figure 36: DevSecOps pipeline sample tooling

[4] https://gitlab.com
[5] https://jenkins.io
[6] https://gatling.io/
[7] https://jestjs.io/
[8] https://jfrog.com/artifactory/
[9] https://kubernetes.io
[10] https://www.zabbix.com/
[11] The ECSO Cybersecurity Market Radar http://www.ecs-org.eu/documents/uploads/the-ecso-cybersecurity-market-radar-high-resolution.pdf provides positioning of cybersecurity tools aligned with the categories identified in the NIST framework.
[12] https://www.sonarqube.org/
[13] https://splunk.com
[14] https://www.open-scap.org/
[15] http://www.openvas.org/
[16] https://github.com/aquasecurity/kube-bench
[17] https://www.elastic.co/products/siem

To ensure continuous integration of the assessment tools, the CAPE framework will rely on both DevOps and DevSecOps methodologies and tooling to facilitate its development processes on one hand and the continuous assessment process on the other.

3.1.20 Task Roadmap

The **goal** of Task 5.1 is to improve the automation of the (self-)assessment process by providing tools and procedures to the assessment activities developed in T5.2 and T5.3. in the context of both WP5 verticals: connected cars (Section 2.1) and e-government (Section 2.2).

The **objectives** of Task 5.1 are to propose 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. The various tool use cases identified in this chapter will be implemented and integrated to form a coherent continuous self-assessment framework. The framework will then be applied on both WP5 verticals through prototypes in a first iteration (D5.2, D5.3) and then demonstrated in D5.4. The tools that can not be demonstrated in the verticals context will be demonstrated independently of the verticals. When possible, tools that are covering opposite levels in the branches of the V-Model will be benchmarked against each others.

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 can be included in D5.2 (cf. Table 43).
- January 2022 (M35): *Final prototypes* have been evaluated in the respective demonstration scenarios, its summary is ready to be included in D5.4.

Development of the ”early prototypes” will typically follow the following timeline:

- M12-M15: **detailed design** of the changes and additions to the various tools based on the uses cases identified in this chapter
- M14-M18: **implementation** of a first prototype version of the use cases identified in this chapter
- M14-M23: **verification and validation** that the framework tools software requirements are satisfied by the implementation. This step can include unit, integration, system and/or acceptance testing steps. Note that this step will start with the implementation when possible.
- M16-M23: **integration** of the various tools to obtain a first prototype version.

The timeline for the final prototypes of the tools will be guided by the demonstration scenarios specified in D5.2 on the verticals and 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

Table 43 provides and overview of use cases supported by tools identified in task 5.1 that will be specified and prototyped in D5.2 and D5.3 and then evaluated in D5.4 :

Tool	Ref	Use-case	Partner
Frama-C (FC)	3.1.3	UC 1 - Runtime errors and vulnerability identification via static analysis UC 2 - Code audit accelerated by a value analysis	CEA
VaCSInE (VCS)	3.1.18	UC 1 - Enforce security policy on an edge infrastructure based on certification criteria UC 2 - Continuous self-assessment for adaptative security with service function chaining	CETIC
Approver (RAA)	3.1.4	Integration in automated pipelines	CINI
Foreshadow-VMM (FS)	3.1.5	UC 1 - Assesment of L1-TF Vulnerability	CNIT
NeSSoS (RA)	3.1.6	UC 1 - Evaluation of e-government risks	CNR
AutoFOCUS3 (AF3)	3.1.16	UC 1 - Support the Safety and Security compliance assessment and certification (platooning) UC 2 - Architecture Modeling for Vertical 1	FTS
IDS / SIEM a.t. (IDS)	3.1.7	UC 1 - Synthetic traffic generation from existing traces UC 2 - Attack traffic mutation	IMT
Package Scanner (PS)	3.1.10	UC 1 - Scan a given open source package	SAP
Steady (VA)	3.1.9	UC 1 - Detect, assess & mitigate dependencies with known vulnerabilities in application projects UC 2 - Detect dependencies with known vulnerabilities in open source projects & suggest mitigations	SAP
OpenCert (OC)	3.1.11	UC 1 - Support the Safety and Security compliance assessment and certification (platooning)	TEC
Sabotage (SB)	3.1.12	UC 1 - Fault-injection and analysis of faulty scenarios with simulation	TEC
Buildwatch (BW)	3.1.17	UC 1 - Build Host State Introspection	UBO
Visual investigation (VI)	3.1.13	UC 1 - Visual Investigation of Large Software Organizations	UKON
Logic Bomb Detection (LBD)	3.1.14	UC 1 - Detecting hidden malicious code	UNILU
SafeCommit (VA2)	3.1.15	UC 1 - Vulnerability Introducing Commit/Patch UC 2 - Vulnerability Fixing Commit/Patch	UNILU

Table 43: Overview of tools extended/developed in the context of T5.1

3.2 T5.2 - Convergence of Security and Safety

In this section, first we discuss the techniques and specifications that we are developing/considering in the Task T5.2, and then discuss the Roadmap of the Task, elaborating on how these techniques and specifications are going to be put together.

3.2.1 Specifications

We classify the types of techniques and specifications that we are working in the following categories:

- **Safety-security co-analysis techniques** is concerned with techniques supporting analyses that take both safety and security into account. Examples of these techniques include techniques that can extract security relevant information from safety analysis and trade-off analyses.
- **Requirements Engineering** discusses methods for representing safety and security requirements. We discuss, in particular, protection profiles as per the Common Criteria as a means for requirements engineering.
- **Modelling and Implementation** discusses Model-Based System/Software Engineering methodology, in particular, the tooling we use to support this modelling and implementation methodology.
- **Safety and Security Co-Verification and Validation Techniques** discusses techniques, such as Formal Methods, Penetration Testing, and Visualization techniques, that can be deployed for verifiying and validating the safety and security of systems/software.
- **Updates** discusses techniques that enable safe and secure update of systems and softwares.
- **Assessments** discusses some of the assessments/standards that take into account both safety and security.

Remark: The task description in the proposal had a narrower scope than the one described above. The proposal focused only on the co-analyses techniques. In order to better support the verticals, in particular, the Connected Car vertical, we included as well Modelling and Implementation, Co-Verification and Validation and Updates. This follows closely the V development model.

3.2.1.1 Safety-security Co-analysis Techniques

Besides improving the safety and the security of systems, the integration of safety and security can lead to a number of benefits. We highlight some possible benefits:

- **Early-On Integration of Safety and Security:** Safety and security assessments can be carried out while the requirements of system features are established. Safety assessments provide concrete hazards which should be treated by security assessments, thus **helping security engineers to set priorities**. For example, a safety hazard shall be given a higher priority compared to other security attacks which do not cause catastrophic events.
- **Verification and Validation:** While safety has many well-established methods for verification, security verification relies mostly on penetration testing techniques, which are system dependent and therefore, resource intensive. The integration of Safety and Security can facilitate security verification. **Much of knowledge gathering can be retrieved from safety assessment, thus saving resources**. For example, FTAs describe the events leading to some hazardous event, while FMEAs describe single-points of failures. This information can be used by security engineers to plan penetration tests, e.g., exploit single-point of failures described in FMEAs, thus leading to increased synergies and less development efforts.
- **Safety and Security Mechanisms Trade-Off Analysis:** By integrating safety and security analysis, it is possible to analyze trade-offs between control and counter-measures proposed to support safety and security arguments. On the one hand, **safety and security measures may support each other, making one of them superfluous.** For example [49, 84], there is not

need to use Cyclic Redundancy Check (CRC) mechanisms for safety, if messages are being signed with MAC (Message Authentication Codes) as the latter already achieves the goal of checking for message corruption. On the other hand, **safety and security mechanisms may conflict with each other.** For example, emergency doors increase safety by allowing one to exit a building in case of fire, but it may decrease security by allowing unauthorized persons to enter the building. Such trade-off analysis can help solve conflicts as well as identify and remove redundancies reducing product and development costs.

Below, we describe some of the main techniques for establishing safety, security and review some techniques proposed for Safety and Security co-analyses.

3.2.1.1.1 Safety techniques

We review some safety techniques used by engineers to evaluate and increase the safety of a system, namely, Hazard Analysis and Risk Assessment (HARA), Fault Tree Analysis (FTA), Failure Modes and Effect Analysis (FMEA), Goal Structured Notation, and Safety mechanisms.

Hazard Analysis and Risk Assessment (HARA)

After the system has been defined, the first actually safety-specific process step is to identify the hazards of the item in terms of accident risks that the system can cause to humans. This activity, in the automotive ISO 26262 standard, is named Hazard Analysis and Risk Assessment (HARA).

HARA is a method to identify and categorize hazardous events of items and to specify safety goals and ASILs related to the prevention or mitigation of the associated hazards in order to avoid unreasonable risk. The hazard identification determines as outputs:

- The **hazards**, which are the ultimate failure effects and can therefore serve as top-level events in a Fault Tree Analysis or other type of safety analysis.
- The **safety goals**, which are the top-level requirements, with the meaning that a given hazard shall be prevented even in presence of failures, and therefore are the starting point for refinement into safety requirements that add to the normal functional requirements for the system.
- The **hazard risk levels** or **safety integrity levels** (ASIL in the ISO 26262), which determine the subsequent effort to be spent on (semi)formal specification, implementation according to certain rules and guidelines, and verification, such as depth of testing or application of formal verification techniques for higher hazard risk levels. They are also an important input parameter for setting up the safety plan (selection of methods to be applied, in particular) and the safety case.

The process of an ISO 26262 HARA is illustrated in Figure 37 (the explanations below refer to the number labels in the diagram). An Item, by definition, is a system or an array of systems that are required to implement a function at the vehicle level (platooning).

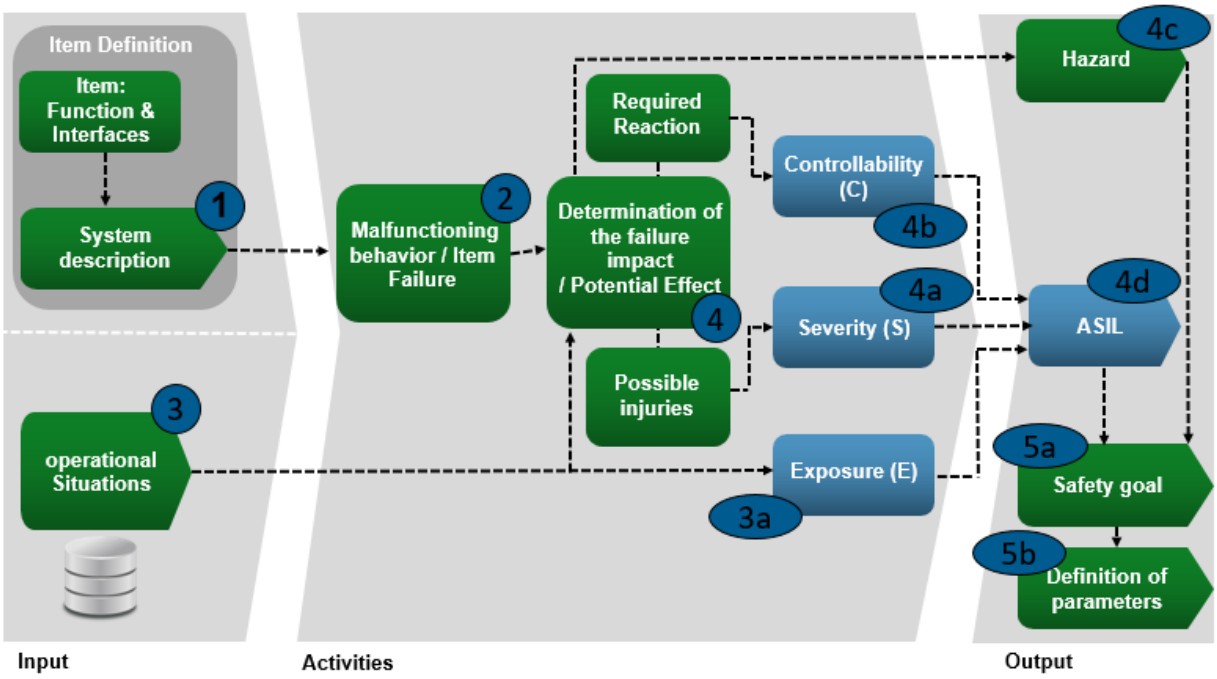


Figure 37: ISO 26262 HARA Process Flow.

Input (1) involves the System Description from Item Definition Phase, showing the outer interfaces and main functions of the system. From these, in step (2) the malfunctions are derived. To judge the potentially resulting accidents, it is necessary to discuss the malfunctions in different environmental contexts or usage scenarios, e.g. driving in city traffic with vulnerable road users nearby, manually driving on a highway at high speed, automatically driving on a highway as part of a platoon at moderate speed, waiting at a red signal with pedestrians crossing in front of the vehicle, etc. This requires the presence of a catalogue of situations (3) as an auxiliary input.

In the next step (4), the effects of the malfunctions are determined for certain environmental and usage scenarios, which finally leads to the definition of the hazard (4c). Associated with step (4) is the rating of the hazard in terms of ASIL. The ASIL is calculated out of three factors:

- E for Exposure (3a), which depends only on the usage scenario and can therefore be provided along with the situation catalogue (cf. Figure 39).
- S for Severity (4a), which depends on the type of injuries that can be expected to any humans – inside or outside of the car – when the described type of accidents happen. These must be manually ranked by experts from S0 – no injuries – to S3 – severe injuries, survival not probable (cf. Figure 38).
- C for Controllability (4b), which rates the probability that the driver or other affected people (e.g. pedestrians, that could in some cases jump aside) can prevent the accident, ranking from C0 – normal control operations that every driver does all the time, like slightly adjusting the speed to C3 – almost uncontrollable. Note that for automated driving functions that allow the driver to perform side tasks, or that run vehicles in close distance to each other in a platoon, C3 must be assumed in most cases (cf. Figure 40).

	Class			
	S0	S1	S2	S3
Description	No injuries	Light and moderate injuries	Severe and life-threatening injuries (survival probable)	Life-threatening injuries (survival uncertain), fatal injuries

Figure 38: ISO 26262-3:2018 Table1: Classes of severity.

	Class				
	E0	E1	E2	E3	E4
Description	Incredible	Very low probability	Low probability	Medium probability	High probability

Figure 39: ISO 26262-3:2018 Table2: Classes of probability of exposure.

	Class			
	C0	C1	C2	C3
Description	Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable

Figure 40: ISO 26262-3:2018 Table3: Classes of controllability.

From these factors, the ASIL can be automatically calculated (4d) according to the Figure 41.

Severity class	Exposure class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A[a]
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 41: ISO 26262-3:2018 Table4: ASIL determination.

The remaining steps are (5a) the naming of a corresponding safety goal (i.e. top-level safety requirement), normally by just adding “avoid” to the hazard name and (5b) the specification of additional parameters, e.g.Safe state, Fault Tolerant Time Interval, Defining parameters. The output of the HARA are the rated hazards that will serve as root events for safety analysis and the rated safety goals that will be refined into more detailed safety requirements.

Fault Tree Analysis (FTA)

Fault Tree Analysis (FTA) is a top-down approach used in order to understand which events may lead to undesired events. It is one of the most important techniques used in safety engineering. An FTA is a tree with the root labeled with the top undesired event. The tree contains “and” and “or” nodes specifying the combination of events that can lead to the undesired event.

Consider, for example, the FTA depicted in Figure 42. The undesired event is Y placed at the root of the tree. A safety engineer is interested on the *cut sets of an FTA*, that is, the collections of events, normally component faults, that lead to the undesired event. For this FTA example, the cut sets are:

$$\{A,D\}, \{B,C\}, \{A\}, \{E,F\}$$

as any of these combinations lead to the event Y. If both A and D happen at the same time, the left-most and branch is satisfied leading to the event Y.

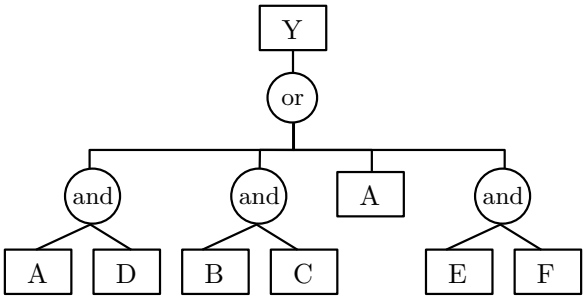


Figure 42: FTA Example.

From a FTA, one can compute the *minimum cut sets*, that is, the minimum set of cut sets that encompasses all possible combinations of triggering an undesired event. The minimum cut set for the given example is

$$\{B,C\}, \{A\}, \{E,F\}$$

Notice that the event A already triggers the event Y. Therefore, there is no need to consider the cut set {A,D}as it is subsumed by the cut set {A}.

Given the minimum cut sets, a safety engineer can, for example, show compliance with respect to the safety requirements. This may require placing control measures to reduce the probability of the corresponding undesired event.

Failure Modes and Effect Analysis (FMEA)

FMEA is a bottom-up approach (inductive method) used to systematically identify potential failures and their potential effects and causes. Thus FMEA complements FTA by instead of reasoning from top-level undesired events as in FTA, adopting a bottom-up approach by starting from faults/failures of sub-components to establish top level failures.

FMEAs are, normally, organized in a table containing the columns: Function, Failure Mode, Effect, Cause, Severity, Occurrence, Detection and the RPN value.

Failure modes are established for each function. Examples of failure modes include [3]:

- **Loss of Function**, that is, when the function is completely lost;
- **Erroneous**, that is, when the function does not behave as expected due to, for example, an implementation bug;
- **Unintended Action**, that is, the function takes the action which was not intended;
- **Partial Loss of Function**, that is, when the function does not operate at full operation, e.g., some of the redundant components of the function are not operational.

Effect and cause are descriptions of, respectively, the impact of the failure mode of the function to safety and what could be a cause for such failure be, e.g., failures of sub-components. Severity, Occurrence and Detection are numbers, ranging normally from 1-10. The higher the value for severity the higher the impact of the failure. The higher the value for occurrence the higher is the likelihood of the failure. The higher the value of detection the less likely it is to observe (and consequently activate control mechanisms) the failure.

Finally, the value RPN is computed by multiplying the values for severity, occurrence and detection. It provides a quantitative way of classifying the importance of failure modes. The higher the value of RPN of a failure the higher its importance.

Goal Structured Notation (GSN)

Safety assessments are complex, breaking an item safety goal into safety sub-goals, e.g., considering different hazards, and often applying different methods, e.g., FTA, FMEA, Safety Mechanisms. GSN [4] is a formalism introduced to present safety assessments in a semi-formal fashion.

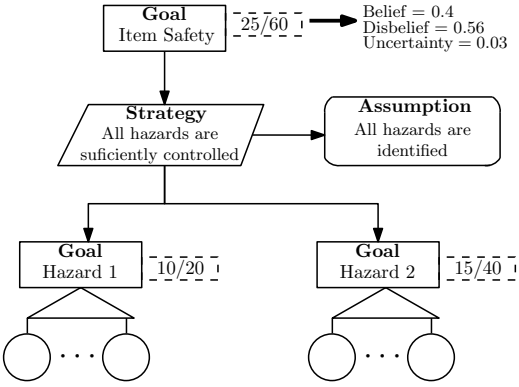


Figure 44: Example of GSN-Model with Quantitative Information. Here the pair m/n attached to goals specifies, respectively, the number of defeaters outruled and the total number of identified defeaters.

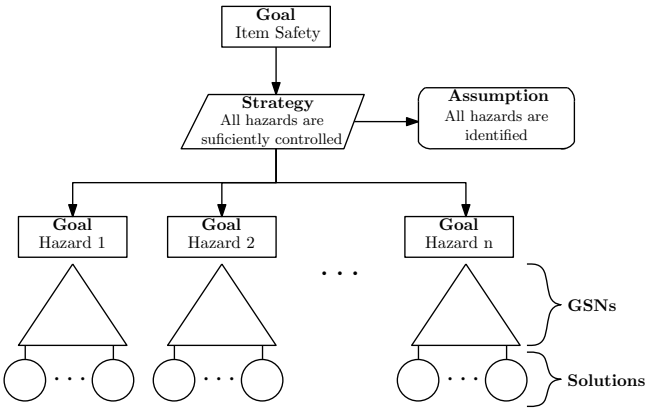


Figure 43: GSN Hazard Pattern.

Since its introduction, different safety arguments have been mapped to GSN patterns. Consider, for example, the GSN pattern depicted in Figure 43. It specifies the argument by analysing all the possible/known hazards to an item's safety. It is assumed that all the hazards are known. For each hazard a safety argument, also represented by a GSN-Model, is specified. At the leaves of the GSN-Model, one describes the *solutions* that have been taken, e.g., carry out FTA, FMEA, safety designs, etc.

Clearly, such safety arguments can provide important information for security. For example, it contains the key safety hazards of an item. It also contains what type of solutions and analysis have been carried out. However, a problem of GSN-Models is the lack of more precise semantics. The semantics is basically the text contained in the GSN-Models, which may be enough for a human to understand, but it does not provide enough structure for extracting automatically security-relevant information. We extend GSN-Models and show how to construct security models, namely, Attack Trees, from a GSN-Model.

Finally, recent works [34, 123] have proposed mechanisms for associating GSN-Models with quantitative values denoting its confidence level. These values are inspired by Dempster-Shafer Theories [33] containing three values for, respectively, the Belief, Disbelief, and Uncertainty on the safety assessment. These values may be assigned by safety experts [123] or be computed from the total number of identified defeaters[18] and the number of defeaters one was able to outrule [34].

We illustrate the approach proposed by Duan et al. [34]. Consider the GSN-Model depicted in Figure 44. It contains a main goal which is broken down into two sub-goals. GSN goals are annotated with the number of defeaters outruled and the total number of defeaters. Intuitively, the greater the

[18] A defeater is a belief that may invalidate an argument.

total number of defeaters, the lower is the uncertainty. Moreover, the greater the number of outruled defeaters the greater the belief on the GSN-Model and the lower the disbelief. In Figure 44, a total of 60 = 20 + 40 defeaters have been identified and only 25 = 10 + 15 have been outruled. These values yield a Belief of 0.4, Disbelief of 0.56 and Uncertainty of 0.03.[19] If further 20 defeaters are outruled, then the Belief is increased to 0.73, the Disbelief reduces to 0.24 and the Uncertainty remains the same value 0.03.

Intuitively, only arguments that have high belief, thus low uncertainty and low disbelief, shall be accepted. Such a quantitative information can be used to incorporate the results of security assessments in safety assessments. For example, if no security assessment has been carried out for a particular item, then the associated uncertainty shall increase. On the other hand, if a security has been carried out establishing that the item is secure, then the belief on the safety of the item shall increase. Otherwise, if an attack is found that could compromise the safety of the item, then the disbelief shall increase.

Safety Mechanisms

Safety mechanisms, such as voters, watchdogs, are often deployed in order to increase the safety of a system. For example, consider the hazard *unintended airbag deployment*. Instead of relying on a single signal, e.g., crashing sensor, to deploy an airbag, a voter can be used to decide to deploy an airbag taking into account multiple (independent) signals, e.g., crashing sensor and gyroscope, thus reducing the chances for this hazard.

However, as pointed out by Preschern et al. [96], safety mechanisms themselves can be subject to attacks. For example, an attacker may tamper the voter leading to a hazard. If security engineers are aware of the deployment of such mechanisms, they can assess how likely it is to attack them to trigger a hazard.

3.2.1.1.2 Security techniques

We review some security techniques used by engineers to evaluate and increase the security of a system, namely, Attack Trees (ATs) and Attack Defense Trees (ADTs).

Attack Trees (ATs)

First proposed by Schneier [105], attack trees and its extensions [21, 72] are among the main security methods for carrying out threat analysis. An attack tree specifies how an attacker could pose a threat to a system. It is analogous to GSN-Models but, instead of arguing for the safety of a system, an attack tree breaks down the possibilities of how an attack could be carried out.

Consider, for example, the Attack Tree depicted in Figure 45. It describes how an intruder can successfully steal a server. He needs to have access to the server's room and be able to exit the building without being noticed. Moreover, In order to access to the server's room, he can break the door or obtain the keys.

Attack Defense Trees (ADTs)

Attack defense trees [72] extend attack trees by allowing to include counter-measures to attack trees. Consider the attack defense tree depicted in Figure 46 which extends the attack tree depicted in Figure 45. It specifies counter-measures, represented by the dotted edges, to the possible attacks. For example, "breaking the door" can be mitigated by installing a security door which is harder to break into. Similarly, installing a security camera or hiring a security guard can mitigate that the attacker leaves the building undetected. Attack defense trees also allow to model how attackers could attack mitigation mechanisms. For example, a cyber-attack on the security camera causing it to record the same image reduces the camera's effectiveness.

[19] We refer to the work of Jøsang [66] on how exactly these values are computed.

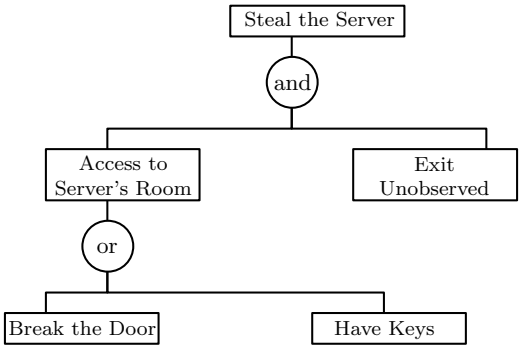


Figure 45: Attack Tree Example.

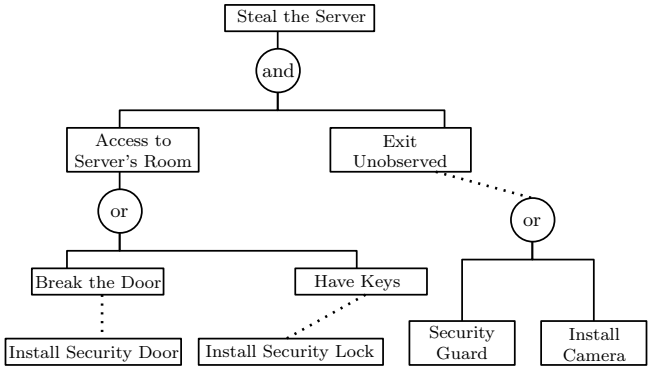


Figure 46: Attack Defense Tree Example.

3.2.1.1.3 Safety and Security co-analysis techniques

We review some safety and security co-analysis techniques used by engineers to evaluate and increase the safety and security of a system.

General Models for Both Safety and Security

A number of works [50, 76, 83] have proposed using general models encompassing both safety and security concerns. For example, GSN extensions with security features, so that in a single framework, one can express both security and safety [76].

Although it is an appealing approach, it does not take into account the different mind-sets between safety and security, which poses serious doubts on the practicality of such approach. On the one hand, security engineers do use GSNs for threat modeling and it is hard to expect them to combine security threats with solutions such as FTA, FMEA, etc. On the other hand, safety engineers are not security experts, so it is hard to expect that they would develop deep security analysis.

Safety Assessments used for Security Analysis

Instead of building a general model for both safety and security, some approaches [35, 101, 110] propose the development of safety assessments and then "passing the ball" to security engineers to carry out security analysis based on the safety assessments.

An example of this approach is the use of standard (natural) language, such as Guide Words [35], with information in safety assessments relevant for carrying out security assessments. For example, HAZOP uses guide words to systematically describe the hazards, such as under which condition it may occur. This information can provide hints for carrying out security analysis.

Automated Integration of Safety into Security Recently [71], we proposed a methodology for extracting security relevant information from safety analyses, such as the ones described above.

The overall methodology is depicted in Figure 47 and consists of the steps described below. Following

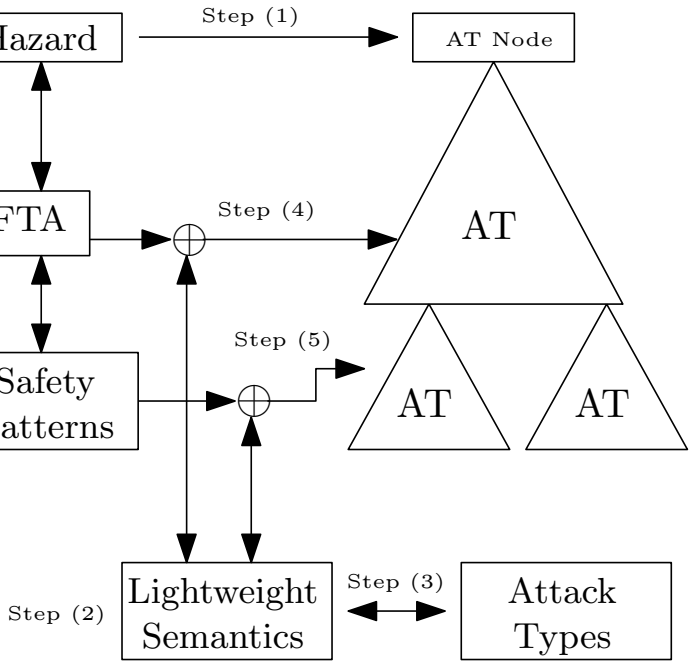


Figure 47: Methodology for translating safety models to Attack Trees

this methodology, one is able to use computer aided methods for extracting security analysis in the form of ATs from safety analysis.

1. **Hazard to Attack Tree** – For each hazard identified in the Hazard analysis, we generate the root node of the attack tree. This follows the guidelines discussed above. The idea is that each attack tree will be expanded using the information collected in the safety analysis.
2. **Lightweight Semantics** – In order to automate the extraction of information from safety analysis, however, we associate to each safety model, e.g., events in FTA, a lightweight semantics. Inspired by the work of [35], this semantics consists of simple annotations, normally already used by safety engineers, e.g., Guide Words, that specify the nature of events involved in the safety analysis and of the types of safety patterns deployed.
3. **Relating Guide Words to Attacks** – We associated with each guide word used in the lightweight semantics the possible attacks that could lead to it. Attacks can be expressed by using the STRIDE notation, for example.
4. **FTA to Attack Tree** – Based on the lightweight semantics associated to the events, an FTA analysis for triggering a hazards is translated to an AT, by reasoning over its minimal cut sets. The AT obtained from the hazards is then expanded using this AT.
5. **Safety Patterns to Attack Trees** – We then further expand the trees by reasoning over the safety patterns, i.e., architectural patterns recommended in security requirements. In particular, new sub-attack trees are constructed enumerating ways for which attackers can trigger a hazard by attacking a safety pattern.

We applied this methodology in a use case involving an Industry 4.0 element.

3.2.1.2 Requirements Engineering

As described in paragraph 3.1.1.1, the model that was identified for the ”Certification for safety and security” process is the V-Model represented by the diagram shown in figure 22. Diagram includes and compares three parallel processes:

1. Security Engineering Process
2. Safety Engineering Process
3. Cybersecurity Certification Process

In this model the phase in which the definition/analysis of the requirements takes place is that identified by the first line on the right of the diagram. In particular the elements of the three parallel processes that characterize them are (keeping the order of the previous bulleted list):

1. Requirement analysis
2. Safety Goals definition
3. ASE (Security Target Evaluation)

In particular, for the certification process, considering the current cyber security certification schemes at a national and international level, Common Criteria (ISO / IEC 15408) [26] has been taken as the starting point, a standard that has evolved and consolidated over the last decade. In this project will take inspiration to undertake the path of improvement of these processes.

Although one of the objectives of WP5 is to simplify the certification processes to make them faster and leaner, while at the same time it is necessary to consider the needs that may arise in dealing with the analysis of very different elements between them.

It is exactly in this direction that moves the proposition made in this project for the definition phase of the requirements in the definition of the ”Requirement Engineering”.

Point 3 of the previous bulleted list contains the phase of the assessment on the Security Target which is the main document of an evaluation conducted according to the Common Criteria standard.

Among the contents of this document there is the precise definition of the security requirements that the ”TOE” (Target of Evaluation - the object that is intended to certify, generally an ICT product, but can also be addressed to systems, processes and services) must satisfy. Therefore, this document has a very limited validity (only to TOE) and is not very reusable as security needs change.

So every time the supplier of a particular TOE intends to proceed to its certification it will have to start from scratch producing such document with a remarkable working effort.

In order to simplify this phase in a possible certification process, the idea is to adopt one of the solutions considered by the Common Criteria standard, the introduction of “Protection Profiles” (PP) for the categorization of Cybersecurity requirements.

3.2.1.2.1 Protection Profile scope

The purpose of a Protection Profile is to describe, with a well-defined scheme, a security problem for a given set of TOE and to specify security requirements to address that problem, without consider how these requirements will be implemented. For this reason it is possible to consider a PP as a security description ”independent of implementation”.

Such a document, due to its flexibility, can satisfy different needs as follows:

- A user community seeking to come to a consensus on the requirements for a given TOE type.
- A developer of a TOE, or a group of developers of similar TOEs wishing to establish a minimum baseline for that type of TOE.
- A government or large corporation specifying its requirements as part of its acquisition process.

3.2.1.2.2 Protection Profile content

The content of a Protection Profile can be summarized in the following points:

- an overview of the PP and a description of the TOE that identify, in adequate terms for users, the security problem to be addressed;

- a description of the security environment of the TOE, which specifies the expected conditions of use, identifying the threats to be countered and the security policies of an organization (OSP) which must be met in the light of specific assumptions;
- the security objectives concerning the evaluation of the TOE, that is, how and to what extent the safety aspects must be considered;
- the functional and guarantee requirements adopted to meet the declared safety objectives;
- the reasons that demonstrate that the declared safety objectives are attributable to all the aspects (hypotheses, threats, OSP) identified in the security environment of the TOE and are suitable to treat them.

In the following paragraphs, you get into more detail about the contents of the various sections of a generic Protection Profile.

a) Descriptive part

This section contains the following elements:

- Identification: sufficient information to identify and uniquely characterize the PP.
- Description: summary of the PP in discursive form, to be used also as an isolated document (to be adopted in catalogs, etc.). Among the elements to be considered in this area are the type of TOE, the functionalities in general and the boundaries of the TOE.

b) Security environment

This section contains the definition of the environment in which the TOE will be used and the way in which it will be used through the following elements:

- Hypotheses made regarding the security environment of the TOE (eg, aspects of connectivity, physical protection, personnel safety, etc.).
- The assets that must be protected and the threats that concern them (deriving from a risk analysis).
- Any organizational policies and security rules that the TOE must satisfy (eg: control rules on data flow, rules for identifying access control, etc.).

c) Security objectives

This section contains the identification and specification of the security objectives of a PP which provide the answer to the elements constituting the security problem defined above. These objectives are divided into:

- Security objectives for the generic TOE, met by technical countermeasures (IT) implemented by the TOE itself.
- Safety objectives for the environment, met by technical measures implemented by the IT environment or by non-IT measures (eg: procedures).

d) Security Requirements

This chapter provides a guide to defining IT security requirements that can be divided into the following categories:

- Security functional requirements (SFR) for generic TOE. These identify the requirements for the security functions that must be provided by the TOE to ensure that the security objectives for the TOE are achieved (these must be written, when possible, using the catalog of functional components defined in [6]).
- Security assurance requirements (SAR) for the generic TOE, which identify the levels of assurance required in the implementation of the security functional requirements (SFR) (these must be written, when possible, using the catalog of warranty components defined in [7]).

- Security requirements for the IT environment. These define the functional or guarantee requirements that must be met by the IT environment (ie from the hardware, firmware and / or external software to the TOE), in order to ensure that the TOE security objectives are achieved.

e) PP rationale

The purpose of the rationale for a PP is to demonstrate that a TOE, which is PP conformant, provides an effective set of IT security countermeasures within the TOE security environment. In particular, rational shows that the security requirements are suitable to meet the security objectives, which in turn are suitable for dealing with all aspects of the TOE security environment (which defines security requirements).

3.2.1.3 Modelling and Implementation

Model-Based System/Software Engineering (MBSE) suggests to rely on models all along the development cycles of (embedded) systems. This usually includes capturing requirements, selecting the “best” HW/SW system partitioning, designing software (and sometimes hardware), generating code and handling the maintenance of the system. Verifications and testing are assumed to be performed from models whenever possible.

Important aspects to be dealt with during the development cycles of embedded systems are the handling of potentially conflicting requirements during all other methodological stages. Requirements can be defined in the three following categories: safety, security and performance. Conflicts can concern opposite needs or goals in the same category, or antagonic goals in two different categories. For instance, adding security requirements usually results in degraded performance. MBSE intends to solve these issues by using different kinds of models, views and viewpoints, that help sharing models between different system experts and allow verifications to be performed.

This section now reviews several MBSE environments, with a focus on DSE and safety/security aspects, including two important candidates in our reseach projects: AutoFocus and TTool/SysML-Sec.

The authors of [104] introduce an abstract design space exploration (DSE) framework, and its integration into design space exploration solvers. Their tool Generic Design Space Exploration, is intended to support DSE for any domain, and allows the use of different solvers for DSE. They allow the user to specify different metrics and constraints to find an optimal solution.

MAESTRO [99] targets the design of embedded firmware, with support for automatic design space exploration and code generation. It also supports evaluation of power consumption, timing, temperature, etc. The Koski design flow models multiprocessor system-on-chips in a UML profile with automated design space exploration [67]. The entire process includes requirement description, application and architectural modeling, architecture exploration, verification by simulation, and code generation.

Capella [93] relies on Arcadia, a comprehensive model-based engineering method. It is intended to check the feasibility of customer requirements, called *needs*, for very large systems. Capella provides architecture diagrams allocating functions to components, and advanced mechanisms to model bit-precise data structures. Capella is however more business focused, and lacks formal verification capabilities.

MARTE [121] models communications, applications, and architecture. However, it intrinsically lacks a separation between control and message exchange. However, even if the UML profile for MARTE adds capabilities to model Real Time and Embedded Systems, it does not specifically support architectural exploration. Other works based on UML/MARTE, such as Gaspard2 [48], are dedicated to both hardware and software synthesis, relying on a refinement process based on user interaction to progressively lower the level of abstraction of input models. However, such a refinement does not completely separate the application (software synthesis) or architecture (hardware synthesis) models from communication.

Other toolkits are specialized for automotive systems, such as Medini, which supports safety analysis and design based on ISO 26262. It supports simulation and probabilistic analysis of faults, but not security analysis [15].

[44] relies on Architecture Analysis and Design Language (AADL) models to consider architectural mapping during security verification. The authors note that a system must be secure on multiple levels: software applications must exchange data in a secure manner, and also execute on a secure memory space and communicate over a secure channel.

[51] enhances Design Space Exploration with the ability to map security tasks in a real time multicore system with the algorithm HYDRA. Their work assumes an attacker who can intercept communications, forge messages, and prevent the availability of services. To impede the attack, security tasks must be performed periodically. Security tasks are abstracted to consider only that they must execute within a set deadline to maintain the security of the system, and not the exact mechanisms for security.

[65] also considered how to secure communications in embedded systems, with encryption performed in software or on FPGA. They considered how to ensure only the confidentiality of their internal messages, with a single encryption algorithm AES. They consider all possible mappings with static and re-configurable FPGA, and determine if the system meets timing constraints. Their work is focused on scheduling and constraint satisfaction.

Likewise, the Simple Modeling Language for Embedded Systems (SMOLES) [109] was enhanced with a Security Analysis Language [36]. SMOLES models systems as a set of components with input and output ports, and tasks are mapped onto the hardware platform. Models can be verified for schedulability, timing, including latencies, and safety properties with UPPAAL. The addition of security algorithms can secure communications across partitions, and also models the attacker capabilities in terms of the size of keys that can be cracked. Their analysis tool can then analyze the fulfillment of integrity and secrecy requirements.

AutoFocus3 [2] addresses requirement capture, software and hardware architectures, with a focus on safety and performance aspects. AutoFocus3 relies on *models*, *views* and *viewpoints*. Analysis techniques help designers investigating e.g. requirements inconsistencies and understanding system behaviour thanks to the simulation of automata. Design Space explorations, based on the Z3 SMT solver, helps deciding of a mapping of software tasks onto HW target platforms. Finally, software code can be automatically generated.

SysML-Sec [1] is an environment to design safe and secure embedded systems with an extended version of the SysML language. SysML-Sec targets both the software and hardware components of these systems. SysML-Sec is fully supported by the free and open-source toolkit: TTool. SysML-Sec has been developed in the scope of the EVITA FP7 European Project [43] and is now used in the scope of the AQUAS H2020 project [16]. Many projects and case studies have already been modeled with SysML-Sec ranging from automotive systems, drone systems, industrial systems, information systems (e.g., the analysis of malware targetting banking systems) and industrial systems (Analysis of SCADA malware), and more generally, security protocols.

The SysML-Sec method is as follows:

- The **analysis** stage intends to elaborate on system requirements, faults and threats.
- The **HW/SW partitioning** stage helps finding a good compromise between main system constraints (cost, safety, security and performance). Safety and security countermeasures can be evaluated using simulation and formal verification techniques. Models are performed at a high level of abstraction, e.g. processors are highly abstract and are customized thanks to a few parameters (e.g. cache miss, wrong branching prediction rate, bus data size, etc.)
- Finally, the **software design** stage supports the design of software components related to functions mapped to processors in previous stage. Software components can be formally verified against safety and security requirements, before executable code is generated.

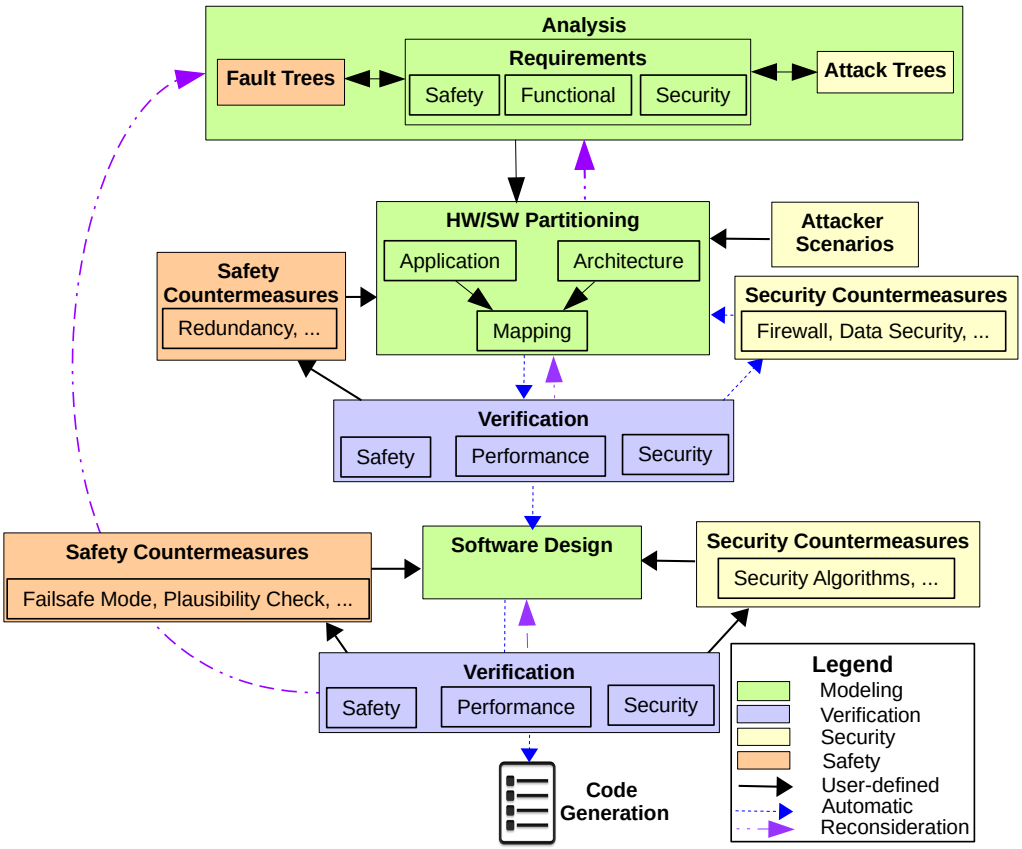


Figure 48: Overview of the SysML-Sec methodology

3.2.1.4 Safety-security Co-verification and Validation Techniques

3.2.1.4.1 Formal methods

The use of mathematical models are useful, especially, during early phase designs of development. Figure 49 illustrates with an example a methodology, that we recently developed [79], using formal methods for carrying out safety and security co-verification.

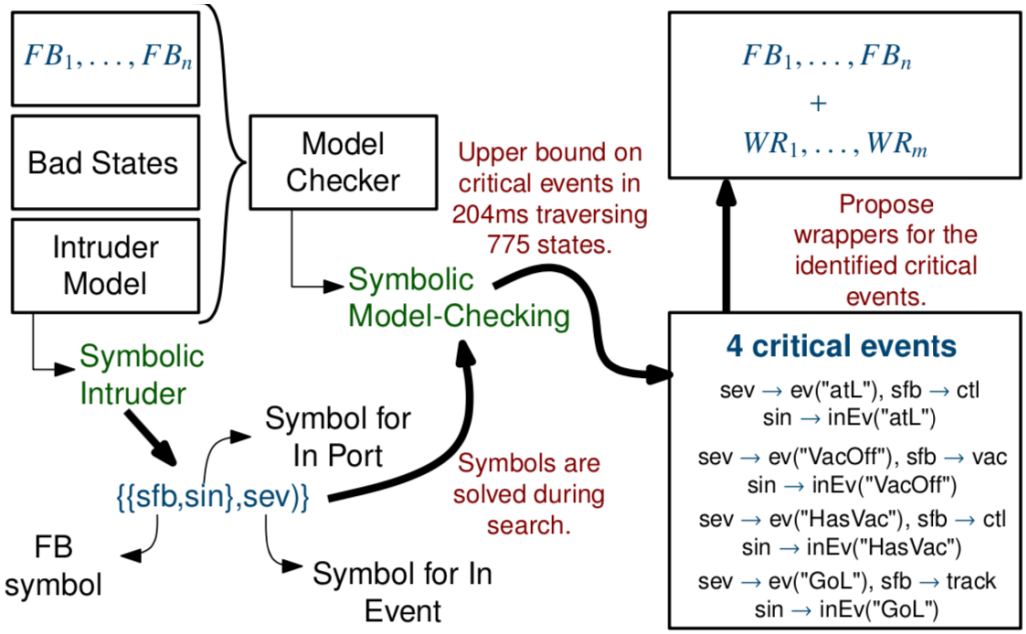


Figure 49: Illustration of an application of formal methods for automated safety and security analyses.

The methodology takes as input:

- a number of function blocks $FB_1, \ldots, FB_n$ specifying the high-level behaviour of the system;
- a number of bad states (or configurations) specifying, for example, catastrophic events;
- a formal specification of an intruder model. It specifies, for example, the capabilities of the intruders, such as capability of injecting or tampering with messages.

The intruder uses symbolic messages. This means that one does not need to inform before-hand, for example, which messages the intruder can inject. The symbols are resolved during verification in a lazy fashion.

These inputs are provided to a model-checker. For this example, we implemented a model-checker in the rewriting tool Maude [24]. The model-checker traverses all possible states and identifies which critical events that an intruder can inject or tamper with, so that it makes the system reach a critical state, i.e., lead to a catastrophic event. In the example shown in Figure 49, four critical events have been identified.

Given these critical events, a designer can propose counter-measures, such as the used of signed messages to ensure the integrity of messages.

3.2.1.4.2 Visualization Techniques

The assessment of safety and security is a primary challenge in the SPARTA project. Vulnerabilities in software influence both the safety and security of systems, e.g., in the context of self-driving cars, an exploitable vulnerability can lead to catastrophic events.

The detection of known vulnerabilities (e.g., CVE's) in software is a primarily automatic process, while the assessment of the potential impact of vulnerabilities on large software projects or organizational structures depends on the expertise of software developers and managers. A meaningful visual representation of such software vulnerabilities and their dependencies can help to identify, explore, interpret such critical safety as well as security risks.

Interactive visualizations support the assessment of such vulnerabilities as they enable us to present the data in a human-readable format, generate hypotheses, and explore the complex relationships between vulnerabilities. More precisely, the goal of such visualizations is to convey the associated risk on modules, components, projects, and organizational structures while also allowing for a detailed analysis of individual vulnerabilities, such that the associated risks can be addressed.

Visual Analytics (VA) is an interdisciplinary approach towards complex data analysis scenarios based on this combination of man and machine. Visual Analytics "combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning, and decision making on the basis of very large and complex datasets", a definition given by Keim et al. as summary of the Vis-Master EU research project [68].

Besides direct knowledge generation, following Visual Analytics principles also fosters a user's constructive reflection and correction of conducted analyses, resulting in improvements for processes and models, and ultimately, of decisions taken and knowledge generated by the users.

VA combines multiple research areas and subjects, including data management and analysis, spatio-temporal data processing, statistics, human-computer interaction, and visualization [69]. It is intended to allow us to derive insights from large, in-homogeneous, and ambiguous datasets and enables both to confirm expected results as well as finding unexpected coherence. Users can quickly come to comprehensible, correct results, and can communicate their findings and derived consequences for action efficiently.

Applied to the SPARTA project, the VA process will be used to improve the decision-making process of analysts in complex use cases that require to simultaneously oversee vulnerabilities in own and third-party project, organizational structures, and their inter-dependencies.

The visual analytics prototype, therefore, will improve the communication of the impending consequences of collaboratively made decisions and their effect on affected organizational units. The knowledge generated by decision-making results and reviews can be used to develop long-term

strategies in software organizations. The knowledge and findings resulting from the visual analysis of vulnerabilities can be additionally used as input for penetration tests refsec:PT.

3.2.1.4.3 Simulation-based Fault Injection

In order to better understand the role of FI on safety assessment, a theoretical background on this field is essential. The FI technique either evaluates or validates the dependability of systems. Dependability of a computer-based system is the ability to avoid service failures that are more frequent and more severe than acceptable. By exploiting such a testing technique, controlled experiments are conducted by the deliberate injection of faults into the system and the reaction is observed. Its main objectives are to:

- Understand the systems behaviour under the effects of real faults
- Evaluate the system fault tolerance
- Forecast the faulty behaviour of the target system
- Identify weak links on the design
- Estimate the coverage and latency of Fault Tolerance Mechanisms (FTM). Actually, as they are not triggered under normal conditions, FI is used to activate those exceptional conditions and to remove FTM design faults.

A detailed description of the different FI techniques and tools is presented in [124]. One of the techniques with more relevant benefits is the so-called **Simulation-based Fault Injection** which allows full observability and controllability. To get meaningful and accurate FI experiment results, a representative fault model is required. Different types of faults can appear depending on its nature during the system design process or during its operational life.

It is beneficial to use simulation technologies before the construction of physical models, as the build-up of virtual model concepts need fewer resources than the preparation of a physical prototype. These techniques also highly recommended across the verification and validation phases of the V-Cycle development process.

As illustrated by Figure 50, the user starts by modelling the system architecture with its corresponding components in a system design tool. After doing so, traditional safety analysis techniques and fault injection are put together in order to perform a combined analysis of the system. The next step is defining the safety concept/safety mechanisms. Then, fault Injection techniques are performed by including saboteurs at component inputs and monitors at component outputs. Finally, the user checks if the safety mechanisms are correctly implemented and a sufficient level of safety has been achieved.

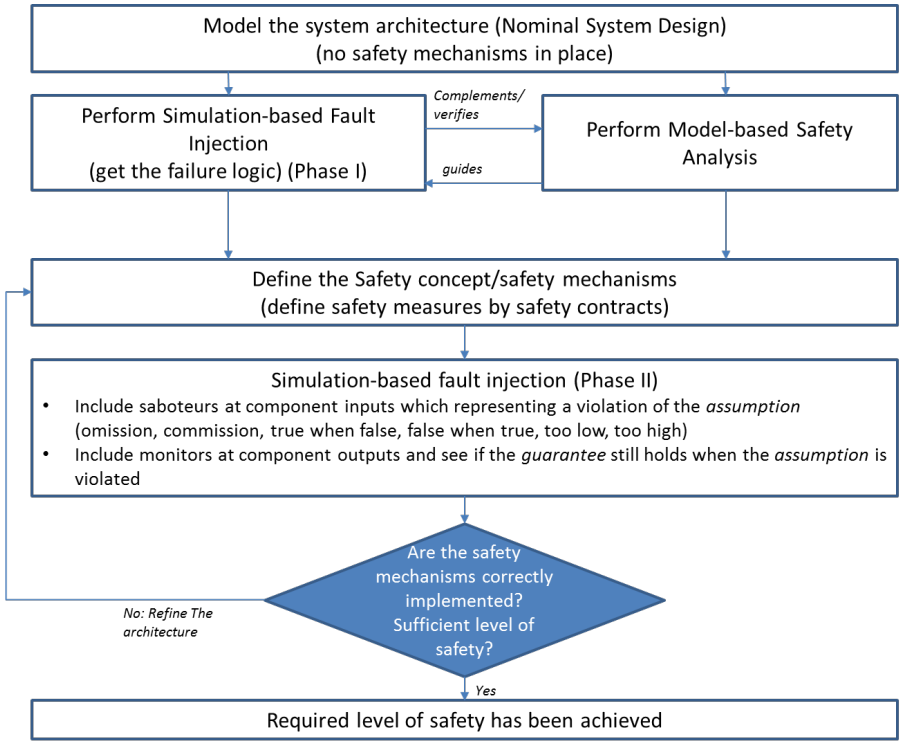


Figure 50: Simulation-based Fault Injection workflow.

3.2.1.4.4 Penetration Testing

Once the system is designed using all the above mentioned methods and techniques it has to be evaluated in terms of security. To do this we will use common methodologies of Ethical Hacking. The ultimate goal of this process is to obtain an objective assessment of the resilience of the system to cyber-attacks. With this cybersecurity assessment, the necessary corrective measures, firmware and software updates, detection mechanisms, prevention, mitigation and recovery of the tested system will be defined.

To achieve this objective, we must analyze each ECU searching for any vulnerability, trying to exploit the vulnerability in order to determine whether unauthorized access or other malicious activity that may compromise ECU security and occupants' safety are possible.*Penetration Testing* is defined as the live test process of the effectiveness of security defenses through mimicking the actions performed by an Ethical Hacker. For the execution of a penetration testing the ethical hacker simulates a criminal attack under controlled conditions:

- **Definition of entry points.** The first step is to determine which are the connectivity devices that may be attacked.
- **Ability to perform an attack.** Once entry points are identified, these will be tried to compromise using several cyber-attacks, obtaining the possible security breaches.
- **Define the breach level.** If any of the performed attacks succeeds, the next step is to determine its level of penetration. Attacks can be passive (attacker can only eavesdrop, intercept messages or resend it) or active (attacker can manipulate data or generate new data), but their impact can have a very broad range.

In conclusion, a penetration test to the system will allow to check that the security requirements involving safety but also other possible security concerns (such as privacy) are satisfied.

One of the tools which is planned to be used here is the Visual Analytics tool from UKON, which due to the complete overview of the SW vulnerabilities of each ECU and their internal relations in the system, it can provide a really useful information to the penetration tester in order to perform complex attacks that take into account not only one, but more ECUs or interfaces.

Moreover, it will analyze the possibilities to add as well HW vulnerabilities of each of the units to support this visualization with a more complete overview of the system in terms of architecture.

3.2.1.5 Updates

Modern automotive vehicles are composed of a large number of cyber-physical systems containing millions of lines of code. Due to the connectivity of the cyber-physical systems and the automotive vehicles it is possible to update the software and firmware for maintenance purposes or for enhancing functionality. The update process is a source of evolution of the embedded software and firmware. From the security point of view this raises questions related to (1) securing the update process to avoid distributing malware within the automotive vehicle and (2) determining whether some certified components need to be recertified (incremental cybersecurity certification). In this section we address (1) by summarising The Update Framework (TUF) and its implementation for the automotive domain Uptane.

In the past, update software repositories have been the subject of attacks. The main aim of attackers was to use the update process as a means of distributing their malware. TUF is a security framework designed to protect update software repositories from attacks. The basic concept in TUF is to guarantee integrity of the repository contents by signing file sizes and hash metadata. Attacks can then be detected and prevented by verifying the metadata before performing updates. TUF is based on four design principles:

- Responsibility separation: metadata is signed using different roles in order to increase resilience to compromise.
- Multi-signature trust: any metadata must be signed using a minimum of x out of y keys.
- Minimal risk to individual keys and roles: offline keys are used for the high impact roles such as root, whereas on-line keys are only used for low risk roles.
- Explicit and implicit revocation: revocation is done explicitly by signing metadata and done implicitly by defining expiration dates for access rights

The four basic roles in TUF are root, time-stamp, snapshot and target. Root is the certificate authority that distributes and revokes public keys that are used to sign the meta-data. Timestamp indicates availability of new updates (new meta-data or new images). The release role identifies the updates that have been released at the same time. The targets role defines the meta-data such as file sizes or hashes.

UPTANE [73] is a software update framework for automotive vehicles. It extends TUF by adding new types of meta-data to be signed thus making attacks more difficult and making the update system more resilient to attacks. The new security principles that UPTANE add to TUF are the following:

- Providing recovery from attacks by making available backups for Electronic Control Units (ECU) sofware compromised by an attack.
- Broadcasting meta-data to prevent attacks where ECU received different versions of meta data at the same time.
- Preventing compatibility attacks by signing a vehicle version manifest that identifies installed software.
- Preventing attacks where ECU updates are delayed indefinitely.

The threat model for UPTANE assumes that attackers have the following goals: (1) Read updates: steal intellectual property by reading updates and reverse-engineering ECU firmware, (2) Prevent fixing vulnerabilities and problems by denying updates, (3) trigger abnormal vehicle behaviour by temporarily or permanently stopping correct ECU behavior, and (4) take control of the vehicle. Attackers are assumed to be capable of performing man in the middle attacks and intercept or change communications. This can be done by an external attack that takes control of a cellular network used for distributing updates or by an internal attack to control communications over a gateway. Attackers are assumed to be capable of compromising vehicle ECUs. Attackers are also assumed of being

capable of compromising the cryptographic keys used to sign meta-data or the server that store the keys.

Software/firmware updates are a key process for software/firmware evolution that needs to be taken into account during cybersecurity assessment. Any updates could require triggering cybersecurity re-assessment of some key functionalities.

3.2.1.6 Assessments

Organizations are spending considerable resources in building proper information security risk management programs that would eventually address the risks they are exposed. These programs should be established on solid foundations, which is the reason why companies look for accepted standards and frameworks.

In the automotive domain three major standardization activities are currently ongoing in the field of safety and cybersecurity, as illustrated by Figure 51:

- **Functional Safety**: The first version of ISO 26262 was published in November 2011. While the standard was a huge success and adopted by the automotive industry, technological developments like the increased usage of assistant functions, increased connectivity and the rising importance of software required a revision and update of the standard. This process is now finalized and ISO 26262: 2018 was published December 2018 [60].
- **Safety of The Intended Functionality – SOTIF**: For automated or autonomous vehicles safety is not only endangered by failures in the classical understanding, e.g. a hardware element is failing, or a software has a design error, but also by misinterpretations of sensor signals or lacking combination of sensor data and processing. SOTIF is a newly developed standard which addresses such issues [59].
- **Automotive Cybersecurity**: Due to the increasing connectivity, V2X communication and the shift of functionality towards software and more complexity that increases the need for Over the Air Updates (OTA), cybersecurity is increasingly important for dependable automotive systems. Recently demonstrated hacker attacks on automotive control systems via maintenance or entertainment channels have shown the necessity as well. Therefore SAE, who already created SAE J3061 as Guideline for Automotive cybersecurity engineering, and ISO have joined forces towards an Automotive Cybersecurity Standard (ISO/SAE 21434) [58].

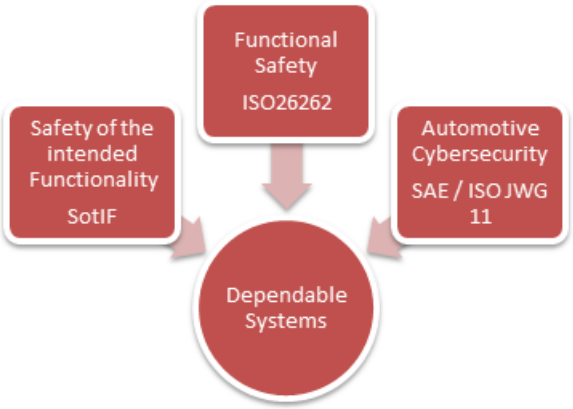


Figure 51: Automotive domain standards.

3.2.1.6.1 Functional Safety according to ISO 26262

The automotive industry implements the ISO 26262 ”Functional Safety Road Vehicles” standard [60] for functional safety, which is the adaptation of the IEC 61508 standard for the automotive industry. The ISO 26262 standard defines what is required to avoid unreasonable risks due to hazards caused by malfunctioning behaviour of Electrical/Electronic systems.

ISO 26262 was published in 2011 and was designed to attend the specific safety risks of the automotive industry and the road vehicles, ensuring the design and build of functionally safe vehicles and efficient safety management through the supply chain. The standard has been considerably reworked and ISO 26262:2018 was published in December 2018.

The Ed. 2 (ISO 26262: 2018) consists of 12 parts – a new part 11 on Semiconductors and another new on for motorcycles. The major goals of the rework have been:

- Increase consistency between parts.
- Adapt standard to evolving technologies and industrial developments.
- Ease adaption and application of standard.
- Extension of the standard for other road vehicles like motorcycles, trucks and busses.
- Extension regarding semiconductors.

As a sub goal for the second edition, it contains some guidance on how to harmonize automotive system engineering with safety and security engineering on the interaction between safety and security teams/activities.

3.2.1.6.2 Safety of the Intended Functionality-SOTIF

ISO 26262 addresses possible hazards caused by malfunctioning behaviour of E/E safety-related systems, including interaction of these systems. The ISO 26262 does not address the nominal performance of E/E systems, but the development of safety-related automated functions needs rules, which are outside of the direct scope of ISO 26262.

New automated functionalities are planned to be introduced in automotive vehicles and such kind of systems rely on information data from the environment provided by different kind of sensor technologies. Such sensors could provide wrongly interpretable data of the environment that could lead to safety violations, even by fault-free systems (e.g. wrong operation of a processing algorithm on environment sensor inputs).

The ISO/TC22/SC32/WG8 was working on a standard under development called SOTIF, which was finally released in January 2019 as ISO PAS 21448:2019 ”Road vehicles-Safety Of The Intended Functionality“ [59], which provides guidance to avoid such kind of violations. It was published as a Publicly Available Specifications (PAS), because, while there was the need for increased guidance, it was also the acknowledged that there is not yet a sufficient state of the art to prescribe requirements.

3.2.1.6.3 Automotive Cybersecurity

Since security is not included on the ISO 26262 standard, ongoing work to extend ISO 26262 to include cybersecurity is been developed, resulting in a new standard for cybersecurity/information security called ISO/SAE 21434 ”Road Vehicles - Cybersecurity engineering” [61]. The designation indicates that the ISO/SAE 21434 standard should be jointly developed by an ISO and SAE working group and then released by both organizations. The ISO/SAE 21434 standard reached the status of Committee Draft in September 2018 and its publication is scheduled for November 2020.

The ISO/SAE 21434 standard is focused on a common terminology and some key aspects of cybersecurity. It aims to help companies demonstrate responsible and careful handling of vehicle development and threat prevention. The activities are controlled on the basis of risk assessment, for this purpose measures for the organizational anchoring are demanded. Although processes are required, the standard only describes the task of a process, leaving the design process to the users. No specific technology or solution is proposed, and no special status has been given to the highly automated vehicles.

Meanwhile, the J3061 ”Cybersecurity guidebook for cyber-physical vehicle systems” [103], was published in January 2016 by the Society of Automotive Engineers (SAE). It is an anticipated standard to fill this gap in security engineering of modern vehicles and provides high-level guidance and information on best practice tools and methods related to cybersecurity in the automotive domain, which can be adapted to existing development processes in an organization.

The J3061 guide builds on many existing works on security engineering and secure system development methodologies and has a wide relation to the ISO 26262, that is why the security lifecycle defined in J3061 is strongly influenced by the safety lifecycle defined in ISO 26262. Moreover, interaction points between the security and safety process are explicitly defined in J3061 to coordinate the two engineering processes. The system lifecycle of the J3061 is divided into concept phase, product development (including system, hardware, and software), production, operation, and service. It also suggests supporting processes such as requirement, change, and quality management.

3.2.2 Task Roadmap

The Task 5.2 is developing and applying the techniques described above in the Connected Car vertical described in Section 2.1. We are also following the steps of the V-model development process as described in Section 3.1.1.1.

Figure 52 depicts the roadmap of activities that we will take. At the moment of writing this deliverable, we have finished the description of the Car Cybersecurity scenario, including the investigation of possible demonstrators. We are currently carrying out the activities to safety and security analysis. This should finish by February 2020. These analyses will lead to requirements, that can be refined by applying safety and security co-analyses techniques, such as trade-off and security/safety by design techniques. The final set of requirements shall be completed by April 2020. Selected features/requirements will be modelled and implemented using the Model-Based Engineering approaches described above that shall end by July 2020. Verification and validation techniques will be applied on the models developed ending at October 2020. Update techniques will be investigated until December 2020 and Assessment until January 2021.

Finally, the task T5.2 ends by contributing to the deliverables 5.2 and 5.3.

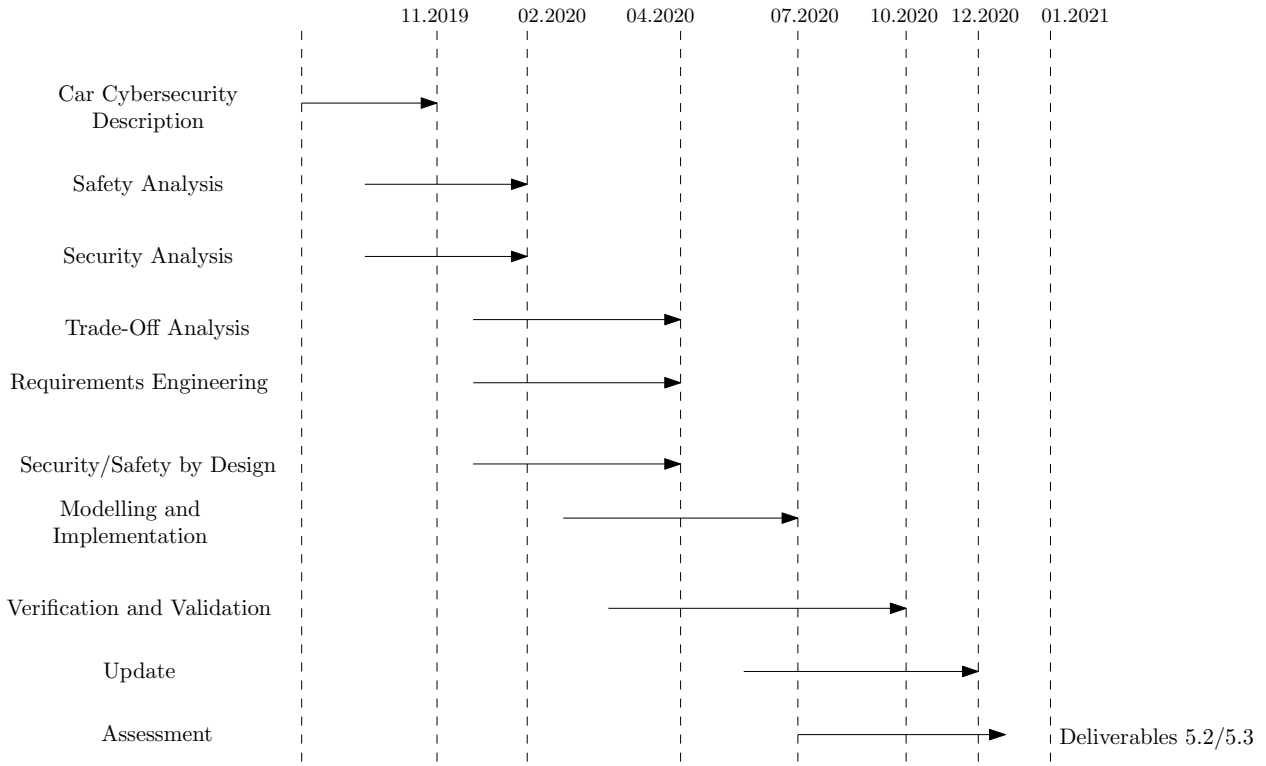


Figure 52: Roadmap for Task 5.2 activities.

Differences to the Proposal Descriptions: The Roadmap above includes a wider range of concerns than the task description in the proposal. For example, the proposal did not consider important points, such as modelling and implementation, co-verification and validation and updates. The jus-

tification of the roadmap above is that it follows the V-development model, considering additionally updates. We believe that including the additional activities will have a great impact on the connected car vertical and help validate the developed methods.

In order to carry out these activities, we will use the tools described in Table 44.

Name	Description	Phase
KAOS/objectiver	Requirements modelling tool that we use to model security and safety goals, and their interaction, identify threats and select countermeasures.	Safety Analysis, Security Analysis, Trade-off Analysis, Requirements Engineering, Safety and Security by Design
EVITA	Threat Assessment Risk Analysis using attack probability, severity level and safety impact parameters.	Security Analysis, Requirements Engineering
AutoFOCUS (3.1.16)	Model-Driven Tool for Embedded Systems. It supports a number of features, including security aspects, e.g., Attack Defense Trees, and safety, e.g., Goal Structured Notation, Formal Verification.	All phases
Maude	Using Model-Checking and SMTs to reason about the security of systems, in particular, to determine whether cyber-attacks can cause catastrophic events	Verification and Validation
SysML-Sec/TTool	SysML-base method to design safe and secure embedded systems, starting from requirements, then system analysis (e.g. fault and attack trees), then HW/SW system partitioning, then software design and finally executable code generation.	All phases
Sabotage (3.1.12)	Model-driven and simulation-based fault injection tool to accomplish an early evaluation dependability evaluation of safety-critical systems. The framework sets up, configures, executes and analyses the simulation results. It includes a fault model library and it is possible to connect to virtual environments such as a virtual vehicle or a robot.	Safety Analysis, Verification and Validation
OpenCert (3.1.11)	PolarSys OpenCert is an integrated and holistic solution for safety-security assurance/certification management of Cyber-Physical Systems (CPS) spanning the largest safety and security-critical industrial markets, such as aerospace, space, railway, manufacturing, energy and health.	Safety Analysis, Trade-Off Analysis, Assessment
VIS SENSE	Visual Analytics Representation of Large Datasets for Enhancing Network Security	Validation and Verification
VALCRI	Visual Analytics for Sense-Making in Criminal Intelligence Analysis	Security Analysis
VASA	Visual Analytics for Security Applications	Security Analysis, Verification and Validation

Table 44: List of Tools to be used by Task 5.2 and respective phase of the roadmap.

3.3 T5.3 - Risk Discovery, Assessment and Management for Complex Systems of Systems

At high-level, software security requirements reflect security objectives, whereby some of those objectives are specific to a given vertical, industry or use-case, while others are largely independent of such. One example of a generic, high-level requirement is that the program code of software products and services shall be free from security vulnerabilities, both the code developed by the given vendor (own code) as well as 3rd party code reused by the product or service in question (3rd party code).

This generic requirement is described in comparable ways in industry-specific standards, sometimes in conjunction with recommended controls. For instance, the security standards of the Payment Card Industry (PCI) enumerate *"technical and operational requirements [...] to protect cardholder data [and which apply] to all entities that store, process or transmit cardholder data – with requirements for software developers and manufacturers of applications and devices used in those transactions."* In this context, requirement 6 of the PCI Data Security Standard (DSS) v3.2.1 [91] demands to *"Establish a process to identify security vulnerabilities"*, and to *"protect all system components and software from known vulnerabilities by installing applicable vendor-supplied security patches"*.

The scope of CAPE task 5.3 is to address this generic security requirement in the light of current software engineering methodologies and technologies. Those trends include, in particular, the ever-increasing use of open source platforms and components through the development life cycle, both heavy-weight application platforms developed by industry consortia as well as libraries and snippets developed as one man show. Another trend is the increasing agility of software development, which results in very short development cycles, as well as semi-automated and automated deployments into production environments, e.g., cloud platforms, through CI/CD pipelines.

The goal of CAPE task 5.3 is to develop a set of tools that can be used by software development organizations to comply with the above-mentioned high-level requirements (a) to keep own source code free of security vulnerabilities, and (b) to detect the presence of known security vulnerabilities in 3rd party software, especially open source components, which is commonly mitigated by the integration of vendor-supplied patches in the software application at hand. Moreover, the task aims at (c) addressing so-called supply chain attacks, in particular ones where attackers try to inject malicious code into upstream open source components.

3.3.1 Context and Background

Figure 53 provides a high-level overview about common systems, actors and activities of a typical development environment. With the exception of *contributors*, all of those elements exist in comparable form both for commercial as well as for open source development projects, no matter the industry or vertical.

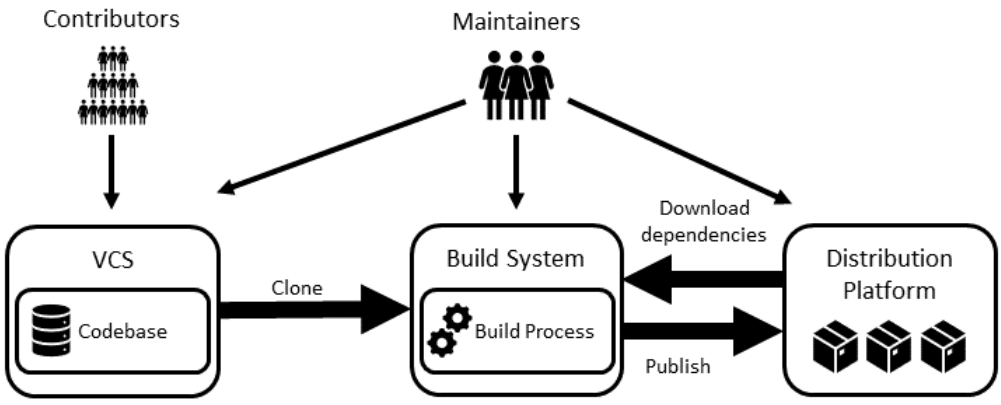


Figure 53: High-level development, build and distribution activities in software projects.

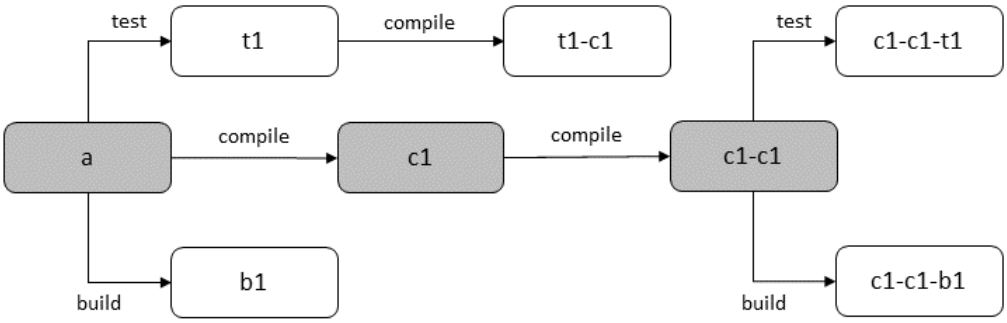


Figure 54: Example Dependency Tree

In this context, *Maintainers* are members of a development project who administer the depicted systems, provide, review and approve contributions, or define and trigger build processes. Open source projects also receive code contributions from *contributors*, which may be reviewed and merged into the project's codebase by maintainers.

The *build process* ingests the source code and other resources of a project, and has the goal to produce software artifacts. These artifacts are subsequently published such that they become available to end-users and other development projects.

The project resources reside in a *VCS*, e.g. Git, and are copied to the local file system of the *build system*. Among those resources is a declaration of direct dependencies, which is analyzed at the start of the build process by a *dependency manager* in order to establish the complete dependency tree with all direct and transitive dependencies. As all of them are required during the build, for instance, at compile time or during test execution, they are downloaded (pulled) from *package repositories* such as PyPI[20] for Python, npm[21] for Node.js, or Maven Central[22] for Java.

Obviously, the number of open source components pulled in this manner into a development project depends on the application at hand. However, it is not uncommon for a commercial application to depend on dozens or even hundreds of open source components, whereby some programming languages and their respective ecosystems tend to pull in more and smaller components than others. Accordingly, a significant share of the overall code base stems from open source projects, up to 80% of typical commercial software applications.

Figure 54 presents a simple supply chain or dependency tree of an imaginary application. Here, the application `a` has three direct dependencies on the software components `t1`, `c1` and `b1`. Those components are downloaded in some automated fashion by package managers during the development of application `a`, and executed at different times according to the respective dependency scope. All of those components are developed, built and distributed according to the illustration of Figure 53. The component `c1-c1` is a transitive dependency of `a`, as it is needed by `a`'s direct dependency `c1`.

At the end of a successful build, program code and other resources are assembled into one or more build artifacts (packages), which are potentially signed and eventually published. Either to distribution platforms like app stores such that they may be consumed by end-users or to *package repositories* for other development projects.

3.3.2 Controls Specification

As mentioned before, the contributions of task 5.3 aim at the detection and mitigation of security vulnerabilities in own and 3rd party code, as well as the detection of so-called supply chain attacks.

The majority of contributions correspond to tools, many of which target specific technologies, e.g.,

[20] https://pypi.org
[21] https://www.npmjs.com
[22] https://search.maven.org/

programming languages, devices or operating systems. Thus, a given software development organization will need to select those tools that match its respective set of technologies. Typically, the integration of those tools happens in the context of an application's build process, which calls one tool after another, and publishes their results in management dashboards.

The tools of this toolbox can be applied to all elements of a supply chain as the one depicted in Figure 54. In the ideal case, this is done by the respective project maintainers, and initiatives like the CII Badge Program [23] reflect the presence of such controls in the maturity rating of open source projects. However, vendors of commercial software products or services cannot assume that all maintainers of all their dependencies are security-aware and follow security best-practices, thus, they need to perform tool-supported security scans by themselves. This can happen in the context of a single application project, e.g., during its respective build process, or by scanning entire code or package repositories.

Moreover, the tools differ in respect to the assessment target. Some focus on source code, e.g., code fragments or entire programs as maintained in VCS, while others consider the binary package that is produced towards the end of a build process.

3.3.2.1 Known and Unknown Vulnerabilities

A common classification of tools supporting the detection of security bugs or vulnerabilities distinguishes between static approaches - static code analysis or static application security testing (SAST) - and dynamic approaches - dynamic application security testing (DAST). Static approaches analyze source or compiled versions of code and do not require the actual execution of the software in question, whereas dynamic approaches have in common to observe actual program execution. Over the course of the last decades, numerous techniques have been developed for both approaches, with different degrees of automation and covering techniques as diverse as formal (mathematical) methods (as an example for SAST) or fuzzing (DAST).

The quality of dynamic approaches generally depends on the test coverage, thus, the share of the program code actually executed by, for instance, penetration testers or automated test cases. Dynamic approaches are therefore prone to false-negatives, i.e., actual problems that remain undetected. While static approaches can consider the whole program code, they generally suffer from false-positive, i.e., wrongly reported findings, due to the fact that corresponding techniques create abstract and simplified representations of software code.

The following contributions of task 5.3 fall into the category of static code analyzers:

First, CEA further extends the open source platform Frama-C (cf. Section 3.1.3), a static analyzer able to verify that source code complies with a formal specification written in a dedicated language, ACSL. The focus of those contributions will be on the integration of Frama-C into automated build pipelines as well as the support of security audits.

Second, SAP continues to work on the open source tool Steady (cf. Section 3.1.9), which aims at detecting the presence of known vulnerabilities in application dependencies. This class of vulnerabilities is among the OWASP Top-10 application security risks [89], and has been the root causes for numerous data breaches. The focus of the contributions will be on developments to improve precision as well as to facilitate tool operation and exchange of vulnerability information.

Third, both UNILU and SAP work on AI-based classification of source code commits with the intention to understand whether a given commit introduces or fixes a security problem (cf. Section 3.1.15). This information can be used, for instance, to down vote pull requests created by project contributors or maintainers, or to populate code-centric vulnerability databases such as the one required by Steady [95]. Of course, those works can be applied to source code changes of both a given application and its upstream dependencies.

Approver, on the other hand, employs both static and dynamic techniques in order to find security vulnerabilities in Android applications. CINI will further extend Approver in the context of SPARTA (cf.

[23] https://www.coreinfrastructure.org/programs/badge-program/

Section 3.1.4), with a focus on the integration into automated build pipelines.

Last, UKON will implement a tool that integrates and visualizes security and supply chain information, and which allows for manual analysis and exploration (cf. Section 3.1.13).

3.3.2.2 Supply Chain Attacks

In general, software supply chain attacks aim to inject malicious code into a software product. Frequently, attackers tamper with the end product of a given vendor such that it carries a valid digital signature, as it is signed by the respective vendor, and may be obtained by end-users through trusted distribution channels, e.g., download or update sites.

A prominent example of such supply chain attacks is NotPetya, a ransomware concealed in a malicious update of a popular Ukrainian accounting software [22]. In 2017, NotPetya targeted Ukrainian companies but also hit global corporations, caused damage worth billions of dollars and is said to be one of the most devastating cyberattacks known today [78]. In the same year, a malicious version of CCleaner, a popular maintenance tool for Microsoft Windows systems, was downloadable from the vendor's official website, and remained undetected for more than a month. During this period it was downloaded around 2.3 million times [70].

Another flavor of supply chain attacks aims at injecting the malicious code into a dependency of a software vendor's product. This attack vector was already predicted by Elias Levy in 2003 [75], and recent years saw a number of real-world attacks following that scheme. Such attacks become possible, because modern software projects commonly depend on multiple open source packages, which themselves introduce numerous transitive dependencies [19]. Such attacks abuse the developers' trust in the authenticity and integrity of packages hosted on commonly used servers and their adoption of automated build systems that encourage this practice [18].

A single open source package may be required by several thousands of open source software projects [64, 125], which makes open source packages a very attractive target for software supply chain attacks. A recent attack on the npm package `event-stream` demonstrates the potential reach of such attacks: The alleged attacker was granted ownership of a prominent npm package simply by asking the original developer to take over its maintenance. At that time, `event-stream` was used by another 1,600 packages, and was in average downloaded 1.5 million times a week [54].

Open source software supply chain attacks are comparable to the problem of vulnerable open source packages which may pass their vulnerability to dependent software projects. This is known as one of the OWASP Top-10 application security risks [89]. However, in case of supply chain attacks, malicious code is deliberately injected and attackers employ obfuscation and evasion techniques to avoid detection by humans or program analysis tools.

Project environments as visualized in 53 are subject to numerous trust boundaries, and many threats target the respective data flows, data stores and processes. Managing those threats may be challenging even when considering only the environment of a single software project. When considering supply chains with dozens or hundreds of dependencies, it is important to notice that such an environment exist for every single dependency, making it obvious that the combined attack surface of such projects is considerably larger than that of software entirely developed in-house.

Taking the perspective of attackers, malevolent actors have the intention to compromise the security of the build or runtime environment of software projects through the infection of one or more upstream open source packages, each one of which is developed in environments comparable to 53. How to reach this goal is described in the following sections by means of two attack trees that provide a structured overview about attack paths to inject a malicious code into dependency trees of downstream users and to trigger its execution at different times or under different conditions.

The attack tree illustrated by 55 is an extension and refinement of the graph presented by Pfretzschner and Othmane [92], and has as top-level goal to inject malicious code into the dependency tree of downstream packages. Thus, the goal is satisfied once a package with malicious code is available on a distribution platform, e.g. package repository, and it became a direct or transitive de-

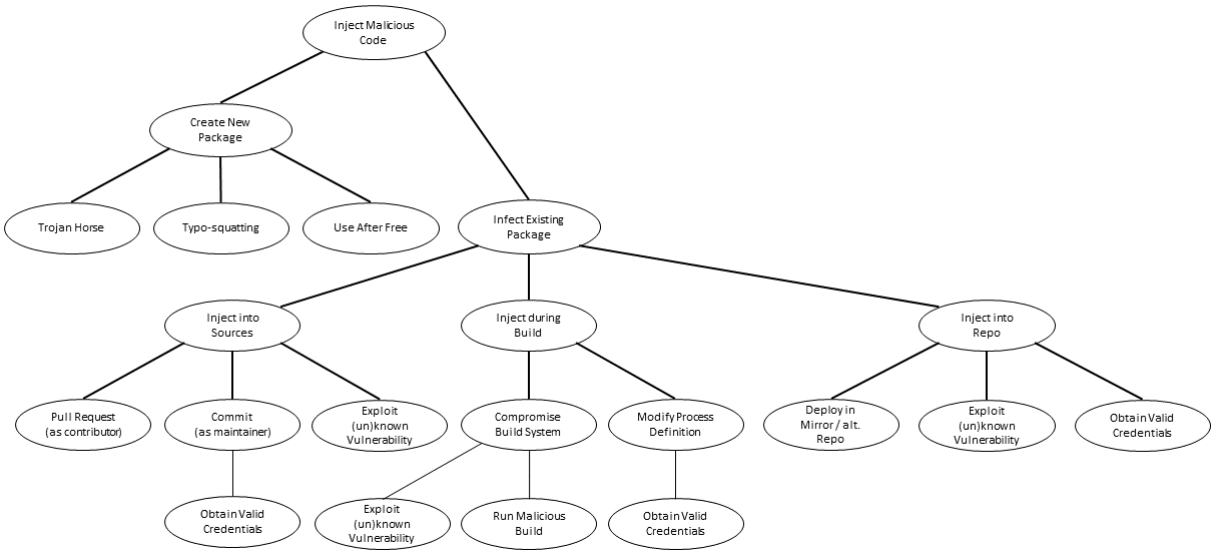


Figure 55: Attack Tree for Open Source Supply Chain Attacks

pendency of one or more other packages. As such, this type of code injection differs from other injection attacks, many of which exploit security vulnerabilities at application runtime, e.g. buffer overflow attacks that become possible due to a lack of user input sanitation.

To inject a package into dependency trees an attacker may follow two possible strategies, he may either *infect an existing package* or submit a *new package*.

Obviously, developing and publishing a new rogue package using a name that is not used by anybody else avoids interference with other legitimate project maintainers. However, such a package has to be discovered and referenced by downstream users in order to end up in the dependency trees of victim packages. This may be achieved using a name similar to existing package names (*typosquatting*) [115], or by developing and promoting a *trojan horse*. An attacker might also use the opportunity to reuse the identifier of an existing project, package, or user account withdrawn by its original and legitimate maintainer (*use after free*).

Another possibility is to infect an *existing package* that already has users, contributors and maintainers. The attacker might choose packages for different reasons, e.g. a significant number or specific group of downstream users. Once the attacker choose a package to infect, the malicious code may be injected *into the sources*, *during the build*, or *into the package repository*.

In the context of such supply chain attacks, SPARTA partners will provide a number of countermeasures, each one addressing particular attacks and/or technologies.

First, given a package, downloaded from a distribution platform such as a website or package repository, SAP will provide a tool to identify (and assess) code of the package that is not present in the respective version control system (cf. Section 3.1.10). As such, the tool addresses attacks aiming to inject code during build time or into the package repository. Obviously, this approach is limited to programming languages and systems where executable code exists in identical form both in the source code repository and the distributed artifact. Most notably, this is not the case for compiled languages such as C/C++. It remains to be clarified whether the approach can be adapted to support interpreted languages whose code is minimized, generated, transpiled or otherwise transformed when being included in a distributed artifact.

Second, UNILU will provide a tool to detect logic bombs in Android applications by leveraging static analysis methods (cf. Section 3.1.14). It addresses malicious code per se, no matter whether it has been introduced through malicious dependencies or in the application code itself.

Third, UBO will provide a sandbox environment to monitor the behavioral changes of build processes over time, e.g., from one commit or tag to another (cf. Section 3.1.17). It can be used by developers when building their application in order to see whether any of their dependencies, as downloaded

from package repositories, perform suspicious activities at installation time or during the execution of tests. And it can also be used to spot such behavioral changes when dependencies are built from scratch, e.g., to overcome the use of 3rd party package repositories altogether.

3.3.3 Task Roadmap

As mentioned before, the contributions of task 5.3 are largely independent of a given vertical, but address generic requirements. Nevertheless, where possible according to the technologies used, the task 5.3 tools will be presented and demonstrated in the context of the connected cars vertical (cf. Section 2.1) and the e-government vertical (cf. Section 2.2). Where this is not possible, due to technology constraints, task 5.3 tools will be demonstrated outside of the context of a specific use-case.

Moreover, since those contributions target different technology stacks, the corresponding tools will not interface one another. The “integration” of multiple tools happens by means of an automated build pipeline, which invokes a subset of the tools according to the technology stack of the assessment target.

A rough 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 and described, the latter of which will be included in D5.3. For each tool, a demo specification for the respective scenario exists and can be included in D5.2 (cf. Table 45).
- January 2022 (M35): *Final prototypes* have been evaluated in the respective demonstration scenarios, its summary is ready to be included in D5.4.

Depending on the respective demo scenario of each tool, the individual contributions may need further alignment with the roadmaps of the respective vertical (cf. Figure 52), or have their own, independent roadmap where the demonstration happens outside the context of a vertical. To that end, the following table provides and overview about all task 5.3 tools and the current status with regard to their demonstration in different verticals and contexts.

Partner	Contribution	Section	Technologies Covered	Use-case
UBO	Build Sandbox	3.1.17	Agnostic	Connected cars
CEA	Frama-C	3.1.3	C	Connected cars
CINI	Approver	3.1.4	Java (Android)	E-government
SAP	Steady	3.1.9	Java, Python	E-government
SAP	Package Scanner	3.1.3	Python	Stand-alone
UNILU	Logic Bomb Detector	3.1.14	Java (Android)	E-government
UNILU	Commit classifier	3.1.15	C/C++	Connected cars
UKON	Supply chain visualization	3.1.13	Agnostic	To be discussed

Table 45: Overview about tools extended/developed in the context of task 5.3

3.4 T5.4 - Integration on Demonstration Cases and Validation

3.4.1 Definition of Certification Requirements Derived from Assessment Procedures and Tools

One of the objectives of task 5.4 is to identify the various phases of the process that allows a potential evaluation facility belonging to a generic certification scheme to demonstrate the evaluability/certifiability of a product/system/process which for convenience will be identified as ”Target of Evaluation” (ToE) below. In proposing this potential process, of course, the following improvement elements, related to currently existing processes, must be taken into account:

1. developing more agile assessment and certification frameworks
2. automation, supporting developers in writing requirements and executing tests
3. assessing systems of systems, beyond individual components, and modularizing assessment to enable assessment of complex systems and services
4. lifetime dynamicity of environments who may have long lifespans
5. execution elasticity, particularly for services

The phases identified for the process are those defined in the following diagram and are detailed in D5.4 (Demonstrators evaluation).

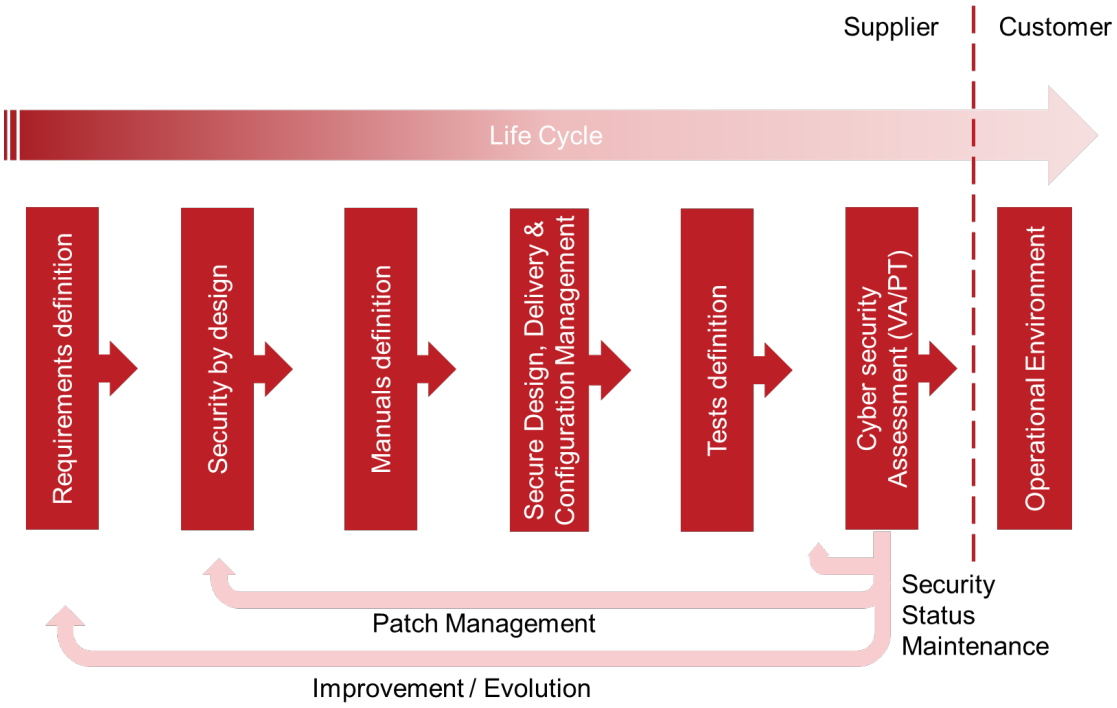


Figure 56: Evaluability process phases

Task 5.1 aims to provide a set of tools/processes that support the developers of a generic ToE in order to be able to generate elements that are already intrinsically secure and that allow them to be securely maintained throughout the life cycle of the same. For this purpose, the validity of the tools/processes identified in terms of security must be demonstrated in this deliverable. Once approved, this set can be used, in the various national or international certification schemes, to simplify the evaluation process of a generic ToE. Therefore the description of these processes/tools must adequately delineate which are the security features that their use allows to introduce in one or more phases of the evaluation/certification process indicated above. On the basis of the indications present in this deliverable, the various certification bodies and the evaluation laboratories will be able to lighten the activities that will be carried out in the various phases of the certification process of a generic ToE. Based on these inputs, the evaluation laboratories in collaboration with the certification bodies will be able to validate

the correctness of the choices made on the three proposed scenarios, thus highlighting the value of this approach in the certification activity.

3.4.2 Task Roadmap

Even if the task 5.4 is not yet started (it will start M18 and finish M36) it is important for the collaboration with other WP5 tasks during their evolution.

In fact they will provide all the necessary requirement that will allow a potential evaluation facility belonging to a generic certification scheme to demonstrate the evaluability/certifiability of the verticals defined in the SPARTA project.

In particular the output of the task 5.1 will define the phases of the V-model process (fig. 22), adopted in this project, where task 5.1 identified tools that are used.

The mapping of these tools will be recognized/approved by accreditation authorities, and could be considered in a hypothetical evaluation scheme to simplify evaluation/certification activities.

In particular the adoption of certain tools in certain phases and their application with a defined process could avoid part of the usual activity of evaluators and reduce time for the whole evaluation process execution.

An approximate task 5.4 development roadmap can be defined by the following list:

1. November 2020 M21: for each phase indicated in figure 56, evidences/deliverables to be provided to the Evaluators to verify the evaluability of the verticals at all stages of the process will be defined in collaboration with the managers of the vertical
2. March 2021 M25: for each vertical, the respective manager will provide all the material available that is necessary to the evaluators for the checks that the evaluation team will carry out to define verticals evaluability
3. January 2022 M35: for each vertical, on the basis of the material received, the evaluability of the prototypes proposed for the two verticals will be verified. Regarding elements not covered by the material delivered, the missing deliverables, necessary in the various stages of the process in order to reach the assessment of the individual vertical, will be defined by the Evaluators.

Chapter 4 Roadmap

In this chapter, the goals and objectives of the various WP5 tasks will be summarized, related activities detailed and a planning in line with the D5.2, D5.3 and D5.4 deliverables will be established.

4.1 Goals and Objectives

The goal of Task 5.1 is to improve the automation of the (self-)assessment process by providing tools and methods to the assessment activities developed in T5.2 and T5.3. The objectives of the task are to propose a framework for automated cybersecurity assessment through integration of various partners' (and external) tools developed or extended in the CAPE program. The tools use cases identified in this deliverable will enable their integration into the CAPE framework and will be developed in the following activities:

- M12-M15: **detailed design** of the various tools based on the identified uses cases
- M14-M18: **implementation** of a first prototype version of the tools
- M14-M23: **verification and validation** of implementation with regard to the framework tools software requirements (unit, integration, system, ... testing)
- M16-M23: **integration** of the tools to obtain first prototype versions.

The timeline for the final prototypes of the tools will follow a similar development cycle, with a focus on integration in the verticals demonstrations:

- M25-M26: **refined design** of the final 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 demonstration scenarios

Task 5.2 goal is to study, develop and apply the techniques and specifications for integration of security and safety in the Connected Car vertical. The following activities are foreseen:

- M9-M12: **safety and security analysis** will produce requirements
- M10-M14: **safety and security co-analyses**: trade-off analysis, requirements engineering, security/safety by design to refine the requirements
- M13-M17: **modelling and implementation** of selected features and requirements
- M13-M20: **verification and validation** of the models
- M16-M22: **updates**
- M17-M23: **assessment**

Task 5.3 goal is to address security requirements on complex systems of systems using modern software engineering methods. The objectives of T5.3 are three fold: detection/mitigation of security vulnerabilities in both own and 3rd party code, and detection of supply chain attacks:

- **Known and unknown vulnerabilities** using SAST and DAST tools: Frama-C, Steady, UniLu VA2, Approver and UKON Supply chain visualization tool
- **Supply chain attacks** with the help of SAP Package Scanner, logic bomb detection (UniLu) inside a sandbox environment (UBO)

The goal of Task 5.4 is to demonstrate the validity of the tools/processes described in T5.1, T5.2 and T5.3 in the CAPE verticals. Those requirement will allow a potential evaluation facility belonging to a generic certification scheme to demonstrate the evaluability/certifiability of the verticals. The following activities are foreseen:

- **Prototypes evaluability definition** - M18-M21 - for each phase of the evaluability process (requirements definition, security by design, tests definition, ...), evidences/deliverables will be produced
- **Prototypes evaluability materials production** - M22-M25: for each vertical, materials necessary to define verticals evaluability will be produced
- **Prototypes evaluability verification** - M26-M35: on the basis of the material received, the evaluability of the prototypes proposed for the two verticals will be verified

4.2 Responsibilities

The work that will be performed in the next steps of the WP5 can be regrouped by demonstration case studies in the verticals:

- vertical 1 - CCCC - TEC, UBO, FTS, IMT, CEA, UNILU
  - Demo 1.1 - Tecnalia Case Study
  - Demo 1.2 - fortiss Case Study
  - Demo 1.3 - IMT Case Study
- vertical 2 - eGovernment - CINI, SAP, UNILU
  - Demo 2.1 - CINI Desktop Scenario
  - Demo 2.2 - CINI Mobile Scenario
- Misc
  - Misc Demo - SAP, CETIC, UKON

4.3 Timeline

The roadmap for the CAPE program is based on the demonstration use cases for the CCCC and e-governement verticals as well as the various independent use-cases, and is realised in a timeframe aligned with the next deliverables:

- D5.2 Demonstrators specifications, M24, TEC: This deliverable provides the specifications of the demonstrators coming out of T5.4. They include contributions on integration mechanisms coming out of the three other tasks.
- D5.3 Demonstrator prototypes, M24, CNIT: This deliverable provides prototypes coming out of T5.4. They include contributions on integration mechanisms coming out of the three other tasks.
- D5.4 Demonstrators evaluation, M36, LEO: This deliverable provides the demonstration of the three verticals described in T5.4. It is supported by an evaluation document.

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

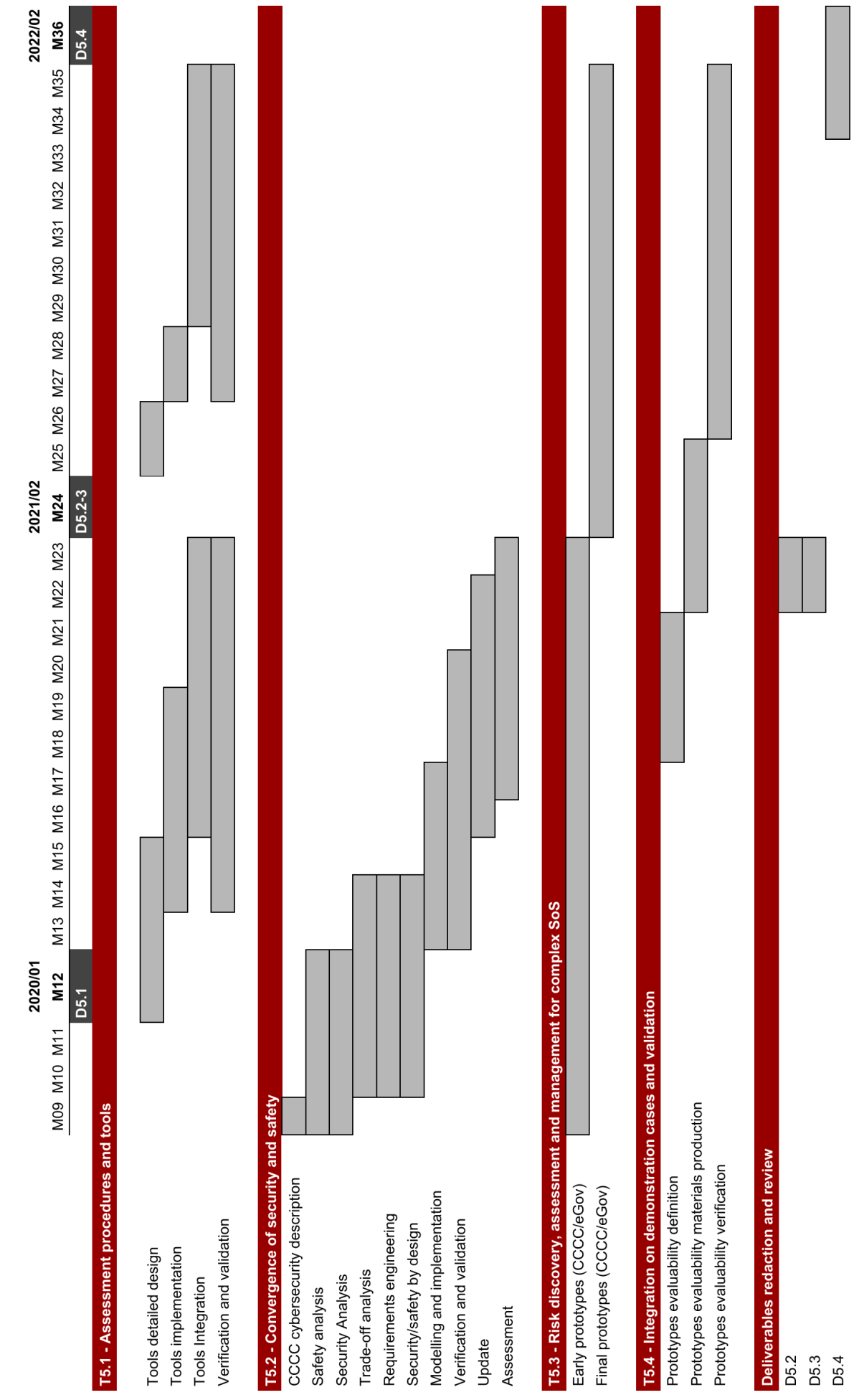


Figure 57: CAPE task roadmap planning

Chapter 5 Summary and Conclusion

This deliverable contains two key contributions for the CAPE program.

The first contribution is a clear description of our use cases, associated with the challenges that we wish to address. This description will be carried on further for the following CAPE deliverables, with the expectation that we will demonstrate most of our tools on one of these two use cases. As justified early on, the third use case (financial platform) was deemed redundant with the other two, which are sufficient to demonstrate tools. Focusing our activities enable a more efficient contribution towards these final demonstrations.

The second contribution is the generic assessment framework, extending the well-known V-Model so that we can clearly and easily position our tools in classic software development. This is extremely important for the acceptance of our tools, as well as for ensuring that we are not leaving empty areas without tooling. The expected coverage will be further demonstrated towards the end of the program.

The assessment tools described in the V-Model assessment framework were developed independently and were not designed to work as an integrated assessment environment. The deliverable proposes a loosely coupled integration of the tools in the form of continuous integration to make the tools available for the automotive and e-government verticals. The approach will be validated with the verticals during the second year of the program. The framework is designed to facilitate self- and continuous assessment for end-users trough automation of the various assessment activities. It will produce original assessment procedures, tools and methods such as simulation-based fault injection, formal verification, software verification, penetration testing, security analysis, etc. On the connected cars vertical for instance, early dependability of the system will be assessed using simulation-based fault injection techniques and behavioral models, security/safety by design will be ensured through model driven approaches (for example using Attack Defense Trees). On the e-Government vertical, static and dynamic analysis methods will be demonstrated for vulnerability detection (through dependency analysis and logic bomb detection) in Android applications.

The deliverable concludes by presenting roadmaps for the second and third years of the research program. Roadmaps are given for each of the tasks, detailing objectives, activities and expected results that will be reported in D5.2, D5.3 and D5.4.

Chapter 6 List of Abbreviations

Abbreviation	Translation
ACC	Adaptive Cruise Control
ADAS	Advanced Driver-Assistance System
ASIL	Automotive Safety Integrity Level
CACC	Cooperative Adaptive Cruise Control
CI/CD	Continuous Integration and Continuous Delivery
CCCC	Connected & Cooperative Car Cybersecurity
CIE	Italian Electronic Identity Card
CPS	Cyber-physical system
CRC	Cyclic Redundancy Check
DoS	Denial of Service
DAST	dynamic application security testing
DC	Direct Current
DSS	Data Security Standard
EAL	Evaluation Assurance Level
ECU	Engine Control Unit
eIDAS	electronic IDentification Authentication and Signature
FI	Fault Injection
FMVEA	Failure Modes, Vulnerabilities and Effects Analysis
FTA	Fault Tree Analysis
FTM	Fault Tolerance Mechanisms
GSN	Goal Structuring Notation
HARA	Hazard Analysis and Risk Assessment
HW	Hardware
IACS	Industrial Automation and Control Systems
IdP	Identity Provider
IPZS	Istituto Poligrafico e Zecca dello Stato
ISMS	Information Security Management System
ISO	International Organization for Standardization
LAN	Local Area Network
MiM	Man in the Middle

Abbreviation	Translation
NFC	Near Field Communication
NFV	Network Function Virtualisation
PAS	Publicly Available Specifications
ROS	Robot Operating System
SAE	Society of Automotive Engineering
SARIF	Static Analysis Results Interchange Format
SAST	static application security testing
SFC	Service Function Chaining
SP	Service Provider
SSO	Single Sign-On
SW	Software
TARA	Threat Analysis and Risk Assessment
TOSCA	OASIS Topology and Orchestration Specification for Cloud Applications
VM	virtual machine
VCS	version control system
WLAN	Wireless Local Area Network

Chapter 7 Bibliography

[1] SysML-Sec. More information at https://sysml-sec.telecom-paris.fr/.

[2] AF3 – AutoFOCUS 3. More information at https://af3.fortiss.org/.

[3] Standard ARP 4761: Guidelines and methods for conducting the safety assessment. Available from https://www.sae.org/standards/content/arp4761/.

[4] *GSN Community Standard Version 1*. 2011. Available at http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf.

[5] Hackers remotely kill a Jeep on the highway—with me in it, 2015. Available at https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[6] Ccmb-2017-04-002 - common criteria for information technology security evaluation, part 2 - security functional components. Technical report, April 2017.

[7] Ccmb-2017-04-003 - common criteria for information technology security evaluation, part 3 - security assurance components. Technical report, April 2017.

[8] A deep flaw in your car lets hackers shut dowm safety features, 2018. Available at https://www.wired.com/story/car-hack-shut-down-safety-features/.

[9] Protection profile - automotive-thin specific tpm - tcg tpm 2.0 automotive thin profile , family 2.0, level 0, version 1.0. Technical report, December 2018.

[10] AMASS. AMASS D3.3 deliverable, pages 57-62, . https://www.amass-ecsel.eu/sites/amass.drupal.pulsartecnalia.com/files/documents/D3.3_Design-of-the-AMASS-tools-and-methods-for-architecture-driven-assurance-%28b%29_AMASS_Final.pdf.

[11] AMASS. AMASS Platform Developers' Guide, . https://www.polarsys.org/opencert/resources/documentation/AMASS-Platform-P2-Developers-Guide_23112018.pdf.

[12] AMASS. AMASS Platform User Manual, . https://www.polarsys.org/opencert/resources/documentation/AMASS-Platform-P2-User-Manual_23112018.pdf.

[13] AMASS. Project website, . URL https://amass-ecsel.eu/.

[14] ANSSI-SV. Anssi security visa website. URL https://www.ssi.gouv.fr/en/security-visa/.

[15] ANSYS. Medini analyze. http://www.medini.eu, 2017.

[16] AQUAS. Project website. URL https://aquas-project.eu/.

[17] J. Arlat, A. Costes, Y. Crouzet, J. Laprie, and D. Powell. Fault injection and dependability evaluation of fault-tolerant systems.

[18] J. West B. Chess, F. DeQuan Lee. Attacking the build through cross-build injection: How your build process can open the gates to a trojan horse. https://www.fortify.com/downloads2/public/fortify_attacking_the_build.pdf, 2007. Accessed: 2019-03-06.

[19] Grey Baker. Keep your dependencies secure and up-to-date with github and dependabot. https://github.blog/2019-01-31-keep-your-dependencies-secure-and-up-to-date-with-github-and-dependabot/, 2019. Accessed: 2019-10-08.

[20] BEACON. Project website. URL https://cordis.europa.eu/project/id/644048.

[21] Stefano Bistarelli, Fabio Fioravanti, and Pamela Peretti. Defense tree for economic evaluations of security investment. In *ARES 06*, pages 416–423, 2006.

[22] Catalin Cimpanu. Petya ransomware outbreak originated in ukraine via tainted accounting software. https://www.bleepingcomputer.com/news/security/petya-ransomware-outbreak-originated-in-ukraine-via-tainted-accounting-software/, 2017. Accessed: 2019-02-24.

[23] CIS-SC. Cis security controls 7.1. URL https://learn.cisecurity.org/cis-controls-download.

[24] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude: A High-Performance Logical Framework*, volume 4350 of *LNCS*. Springer, 2007.

[25] James S Collofello. Introduction to software verification and validation. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1988.

[26] CommonCriteria. Project website portal. URL https://www.commoncriteriaportal.org/.

[27] SEC Consult. Authentication bypass in eIDAS-node, 2019. Available at https://sec-consult.com/en/blog/advisories/15587/.

[28] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1773–1788. ACM, 2017. ISBN 978-1-4503-4946-8. doi: 10.1145/3133956.3134063. URL https://doi.org/10.1145/3133956.3134063.

[29] CSA-CCM. Working group website. URL https://cloudsecurityalliance.org/research/working-groups/cloud-controls-matrix.

[30] CyberEssentials. Crest cyber essentials overview. URL https://www.cyberessentials.org/.

[31] Trey Darley and Ivan Kirillov. *STIX™ Version 2.0. Part 3: Cyber Observable Core Concepts*. OASIS Open, committee specification 01 edition, 2017. https://oasis-open.github.io/cti-documentation/resources.html.

[32] DECODER. Project website, 2019. URL https://cordis.europa.eu/project/id/824231.

[33] A. P. Dempster. Upper and lower probabilities induced by a multivalued mapping. *The Annals of Mathematical Statistics*, 1967.

[34] Lian Duan, Sanjai Rayadurgam, Mats Heimdahl, Oleg Sokolsky, and Insup Lee. Representation of confidence in assurance cases using the beta distribution. 2016.

[35] Jürgen Dürrwang, Kristian Beckers, and Reiner Kriesten. A lightweight threat analysis approach intertwining safety and security for the automotive domain. In Stefano Tonetta, Erwin Schoitsch, and Friedemann Bitsch, editors, *SAFECOMP*, volume 10488 of *LNCS*, pages 305–319. Springer, 2017. ISBN 978-3-319-66265-7. doi: 10.1007/978-3-319-66266-4_20. URL https://doi.org/10.1007/978-3-319-66266-4_20.

[36] M. Eby, J. Werner, G. Karsai, and A. Ledeczi. Integrating security modeling into embedded system design. In *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, pages 221–228, March 2007. doi: 10.1109/ECBS.2007.45.

[37] ECSO-sota. Ecso state of the art syllabus overview of existing cybersecurity standards and certification schemes v2. URL https://ecs-org.eu/documents/publications/5a31129ea8e97.pdf.

[38] eITUS. Project website. URL https://robmosys.eu/e-itus/.

[39] Tamas Elteto and Sandor Molnar. On the distribution of round-trip delays in tcp/ip networks. pages 172–181, 11 1999. ISBN 0-7695-0309-8. doi: 10.1109/LCN.1999.802014.

[40] Nils Engelbertz, Nurullah Erinola, David Herring, Juraj Somorovsky, Vladislav Mladenov, and Jörg Schwenk. Security analysis of eidas – the cross-country authentication scheme in europe. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, Baltimore, MD, August 2018. USENIX Association. URL https://www.usenix.org/conference/woot18/presentation/engelbertz.

[41] ENISA. Enisa factsheet on eu framework for cybersecurity certification. URL http://ec.europa.eu/newsroom/document.cfm?doc_id=46999.

[42] EU-CSA. Press release - eu negotiators agree on strengthening europe's cybersecurity. URL https://ec.europa.eu/commission/presscorner/detail/en/IP_18_6759.

[43] EVITA. Project website. URL https://www.evita-project.org.

[44] Peter H. Feiler, Bruce A. Lewis, Steve Vestal, and Edward Colbert. An overview of the SAE architecture analysis & design language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering. In *IFIP-WADL*, volume 176 of *IFIP*, pages 3–15. Springer, 2004. ISBN 978-0-387-24589-8.

[45] FINCSC. Fincsc website. URL https://www.fincsc.fi/en/services/.

[46] Marcus S Fisher. *Software verification and validation: an engineering and scientific approach*. Springer Science & Business Media, 2007.

[47] Stichting Cuckoo Foundation. Cuckoo Sandbox - Automated Malware Analysis. https://cuckoosandbox.org, 2019. Accessed: 2019-11-29.

[48] Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Rabie Ben Atitallah, Anne Etien, Philippe Marquet, and Jean-Luc Dekeyser. A model-driven design framework for massively parallel embedded systems. *ACM Trans. Embedded Comput. Syst*, 10(4):39, 2011.

[49] Benjamin Glas, Carsten Gebauer, Jochen Hänger, Andreas Heyl, Jürgen Klarmann, Stefan Kriso, Priyamvadha Vembar, and Philipp Wörz. Automotive safety and security integration challenges. In Herbert Klenk, Hubert B. Keller, Erhard Plödereder, and Peter Dencker, editors, *Automotive - Safety & Security 2014 (2015), Sicherheit und Zuverlässigkeit für automobile Informationstechnik, Tagung, 21.-22.04.2015, Stuttgart, Germany*, volume 240 of *LNI*, pages 13–28. GI, 2014. ISBN 978-3-88579-634-3. URL https://dl.gi.de/20.500.12116/2456.

[50] Edward Griffor, editor. *Handbook of System Safety and Security*. 2016. ISBN 9780128037737.

[51] Monowar Hasan, Sibin Mohan, Rodolfo Pellizzoni, and Rakesh B Bobba. A design-space exploration for allocating security tasks in multicore real-time systems. *arXiv preprint arXiv:1711.04808*, 2017.

[52] Ian i. Mason, Vivek Nigam, Carolyn L. Talcott, and Alisson Vasconcelos De Brito. A framework for analyzing adaptive autonomous aerial vehicles. In *Software Engineering and Formal Methods - SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA, Trento, Italy, September 4-5, 2017, Revised Selected Papers*, pages 406–422, 2017. doi: 10.1007/978-3-319-74781-1_28. URL https://doi.org/10.1007/978-3-319-74781-1_28.

[53] IEC62443. Iec 62443-4-1:2018 - security for industrial automation and control systems - part 4-1: Secure product development lifecycle requirements. URL https://webstore.iec.ch/publication/33615.

[54] Th. Hunter II. Compromised npm package: event-stream. https://medium.com/intrinsic/compromised-npm-package-event-stream-d47d08605502, 2018. Accessed: 2019-03-06.

[55] IISF. Industrial internet security framework technical report. URL https://www.iiconsortium.org/IISF.htm.

[56] INCSF. 2015 italian cyber security report a national cyber security framework, research center of cyber intelligence and information security sapienza universita di roma, cini cyber security national laboratory national interuniversity consortium for informatics. URL https://www.cybersecurityframework.it/sites/default/files/CSR2015_ENG.pdf.

[57] ISKE. Three-level it baseline security system iske website. URL https://www.ria.ee/en/cyber-security/it-baseline-security-system-iske.html.

[58] ISO21434. Iso/sae cd 21434 road vehicles - cybersecurity engineering website. URL https://www.iso.org/standard/70918.html.

[59] ISO21448. Iso/pas 21448:2019 - road vehicles — safety of the intended functionality. URL https://www.iso.org/standard/70939.html.

[60] ISO26262. Iso 26262-1:2018 - funtional safety road vehicles. URL https://www.iso.org/standard/68383.html.

[61] ISO27034. Iso/iec 27034-1:2011 - information technology - security techniques - application security. URL https://www.iso.org/standard/44378.html.

[62] ISO27K. Project website. URL https://www.iso.org/isoiec-27001-information-security.html.

[63] ITGrundschutz. Germany - federal office for information security website. URL https://www.bsi.bund.de/EN/Topics/ITGrundschutz/itgrundschutz_node.html.

[64] Michał Janaszek. State of package.json dependencies. https://medium.com/warsawjs/state-of-package-json-dependencies-de99828b6c3f, 2018. Accessed: 2019-10-08.

[65] K. Jiang, P. Eles, and Z. Peng. Co-design techniques for distributed real-time embedded systems with communication security constraints. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 947–952, March 2012. doi: 10.1109/DATE.2012.6176633.

[66] Audun Jøsang. A logic for uncertain probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2001.

[67] Tero Kangas, Petri Kukkala, Heikki Orsila, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen, Jouni Riihimäki, and Kimmo Kuusilinna. UML-based Multiprocessor SoC Design Framework. *ACM Trans. Embed. Comput. Syst.*, 5(2):281–320, May 2006. ISSN 1539-9087. doi: 10.1145/1151074.1151077. URL http://doi.acm.org/10.1145/1151074.1151077.

[68] Daniel Keim. Mastering the information age: solving problems with visual analytics. 2010.

[69] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. Visual analytics: Definition, process, and challenges. In *Information visualization*, pages 154–175. Springer, 2008.

[70] Swati Khandelwal. Ccleaner attack timeline - here's how hackers infected 2.3 million pcs. https://thehackernews.com/2018/04/ccleaner-malware-attack.html, 2018. Accessed: 2019-02-24.

[71] Antoaneta Kondeva, Carmen Carlan, Harald Ruess, and Vivek Nigam. On computer-aided techniques for supporting safety and security co-engineering. In *The 9th IEEE International Workshop on Software Certification WoSoCer*, 2019.

[72] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. Foundations of attack-defense trees. pages 80–95, 2010. doi: 10.1007/978-3-642-19751-2_6. URL https://doi.org/10.1007/978-3-642-19751-2_6.

[73] Trishank Karthik Kuppusamy, Lois DeLong, and Justin Cappos. Securing software updates for automotives using uptane. *;login:*, 42(2), 2017. URL https://www.usenix.org/publications/login/summer2017/kuppusamy.

[74] Sonali S. Lagu and Sanjay B. Deshmukh. Raspberry Pi for Automation of Water Treatment Plant. In *Computing Communication Control and Automation (ICCUBEA), 2015 International Conference on*, pages 532–536, February 2015. doi: 10.1109/ICCUBEA.2015.109.

[75] Elias Levy. Poisoning the software supply chain. *IEEE Security & Privacy*, 1(3):70–73, 2003.

[76] Per Håkon Meland, Elda Paja, Erlend Andreas Gjære, Stéphane Paul, Fabiano Dalpiaz, and Paolo Giorgini. Threat analysis in goal-oriented security requirements modelling. *Int. J. Secur. Softw. Eng.*, 5(2):1–19, 2014. ISSN 1947-3036. doi: 10.4018/ijsse.2014040101. URL http://dx.doi.org/10.4018/ijsse.2014040101.

[77] Modbus Organization. Official Modbus Specifications, 2016, http://www.modbus.org/specs.php, Last access: April 2019.

[78] Alfred NG. Us: Russia's notpetya the most destructive cyberattack ever. https:

//www.cnet.com/news/uk-said-russia-is-behind-destructive-2017-cyberattack-in-ukraine/, 2018. Accessed: 2019-02-25.

[79] Vivek Nigam and Carolyn Talcott. Formal security verification of industry 4.0 applications. In *ETFA*, 2019.

[80] Vivek Nigam, Carolyn Talcott, and Abraão Aires Urquiza. Towards the automated verification of cyber-physical security protocols: Bounding the number of timed intruders. In *European Symposium on Research in Computer Security (ESORICS)*, 2016.

[81] Vivek Nigam, Alexander Pretschner, and Harald Ruess. Model-based safety and security engineering. White Paper, 2018.

[82] NIST-CSF. Us gsa website. URL https://www.gsa.gov/technology/technology-products-services/it-security/nist-cybersecurity-framework-csf.

[83] Nicola Nostro, Andrea Bondavalli, and Nuno Silva. Adding security concerns to safety critical certification. In *Symposium on Software Reliability Engineering Workshops*, 2014.

[84] Thomas Novak, Albert Treytl, and Peter Palensky1. Common approach to functional safety and system security in building automation and control systems. 2007.

[85] OASIS. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, 2005. Available at http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.

[86] OASIS. SAML V2.0 Technical Overview, 2005. Available at http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02.pdf.

[87] DPCM of 24 October 2014. Sistema Pubblico per la gestione dell'Identità Digitale (SPID), 2014. Available at http://www.agid.gov.it/agenda-digitale/infrastrutture-architetture/spid.

[88] OPENCOSS. Project website. URL http://www.opencoss-project.eu/.

[89] OWASP. OWASP Top 10 security risk A9, Using Components with Known Vulnerabilities. https://www.owasp.org/index.php/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities.

[90] European Parliament. Regulation 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC, 2014. Available at http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014R0910&from=EN.

[91] PCI Security Standards Council. Payment Card Industry (PCI) Data Security Standard v3.2.1, Last Access: November 2019. Available at https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf?agreement=true&time=1574868634382.

[92] Brian Pfretzschner and Lotfi ben Othmane. Identification of dependency-based attacks on node.js. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, page 68. ACM, 2017.

[93] Polarsys. ARCADIA/CAPELLA (webpage). In *https://www.polarsys.org/capella/arcadia.html*, 2008.

[94] Christophe Ponsard and Jeremy Grandclaudon. Survey and guidelines for the design and deployment of a cyber security label for smes. In Paolo Mori, Steven Furnell, and Olivier Camp, editors, *Information Systems Security and Privacy*, pages 240–260, Cham, 2019. Springer International Publishing. ISBN 978-3-030-25109-3.

[95] Serena E. Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cedric Dangremont. A manually-curated dataset of fixes to vulnerabilities of open-source software. In *Proceedings of the 16th International Conference on Mining Software Repositories*, May 2019. URL https://arxiv.org/pdf/1902.02595.pdf.

[96] Christopher Preschern, Nermin Kajtazovic, and Christian Kreiner. Security analysis of safety patterns. PLoP '13, pages 12:1–12:38, USA, 2013. ISBN 978-1-941652-00-8. URL http://dl.acm.org/citation.cfm?id=2725669.2725684.

[97] Carlos Queiroz, Abdun Mahmood, and Zahir Tari. Scadasim—a framework for building scada simulations. *IEEE Transactions on Smart Grid*, 2(4):589–597, 2011.

[98] Mark Rollins. *Beginning LEGO MINDSTORMS EV3*. Apress, 2014.

[99] Rafael Rosales, Michael Glass, Jürgen Teich, Bo Wang, Yang Xu, and Ralph Hasholzner. MAESTRO— Holistic Actor-Oriented Modeling of Nonfunctional Properties and Firmware Behavior for MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.*, 19(3):23:1–23:26, June 2014. ISSN 1084-4309. doi: 10.1145/2594481. URL http://doi.acm.org/10.1145/2594481.

[100] Jose Rubio-Hernan, Juan Rodolfo-Mejias, and Joaquin Garcia-Alfaro. Security of cyber-physical systems — from theory to testbeds and validation. In *Security of Industrial Control Systems and Cyber-Physical Systems – Second International Workshop, CyberICPS 2016, Heraklion, Crete, Greece, September 26-30, 2016, Revised Selected Papers*, pages 3–18. Springer, September 2016. doi: 10.1007/978-3-319-61437-3_1. URL https://doi.org/10.1007/978-3-319-61437-3_1.

[101] Giedre Sabaliauskaite, Lin Shen Liew, and Jin Cui. Integrating autonomous vehicle safety and security analysis using STPA method and the six-step model. *International Journal on Advances in Security*, 11, 2018.

[102] SABOTAGE. Sabotage: A Simulation-Based Fault Injection Tool Framework. https://www.cyberssbytecnalia.com/node/271.

[103] SAEJ3061. Sae j3061 - cybersecurity guidebook for cyber-physical vehicle systems. URL https://www.sae.org/standards/content/j3061_201601/.

[104] Tripti Saxena and Gabor Karsai. *MDE-Based Approach for Generalizing Design Space Exploration*, pages 46–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-16145-2.

[105] B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's Journal of Software Tools*, 24: 21–29, 1999.

[106] Mariana Segovia, Ana Cavalli, Nora Cuppens, Jose Rubio-Hernan, and Joaquin Garcia-Alfaro. Reflective attenuation of cyber-physical attacks. In *Security of Industrial Control Systems and Cyber-Physical Systems – 5th International Workshop, CyberICPS 2019, Luxembourg, September 2019, Revised Selected Papers, ESORICS workshops 2019*. Springer, September 2019.

[107] SPARTA. Deliverable D.11.1, Mapping of International and national cybersecurity certification initiatives, January 2020.

[108] STANCE. Project website, 2011. URL https://cordis.europa.eu/project/id/317753.

[109] Tivadar Szemethy and Gabor Karsai. Platform modeling and model transformations for analysis. *Journal of Universal Computer Science*, 10(10):1383–1407, 2004.

[110] Kenji Taguchi, Daisuke Souma, and Hideaki Nishihara. Safe & sec case patterns. In *SAFECOMP 2015 Workshops, ASSURE, DECSoS, ISSE, ReSA4CI, and SASSUR*, 2015. doi: 10.1007/978-3-319-24249-1_3. URL https://doi.org/10.1007/978-3-319-24249-1_3.

[111] André Teixeira, Iman Shames, Henrik Sandberg, and Karl Henrik Johansson. A secure control framework for resource-limited adversaries. *Automatica*, 51:135–148, 2015. ISSN 0005-1098. doi: http://dx.doi.org/10.1016/j.automatica.2014.10.067.

[112] The OMNeT++ Network Simulation Framework, Last Access: April 2019. Available at http://www.omnetpp.org/.

[113] The OMNeT++/INET Framework, Last Access: April 2019. Available at http://inet.omnetpp.org/.

[114] TOSCA. Project website. URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca.

[115] Nikolai Philipp Tschacher. Typosquatting in programming language package managers. Master's thesis, Universität Hamburg, Fachbereich Informatik, 2016.

[116] U3CAT. Project website, 2010. URL https://frama-c.com/u3cat.

[117] Abraão Aires Urquiza, Musab A. AlTurki, Max I. Kanovich, Tajana Ban Kirigin, Vivek Nigam, Andre Scedrov, and Carolyn L. Talcott. Resource-bounded intruders in denial of service attacks. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, pages 382–396, 2019. doi: 10.1109/CSF.2019.00033.

[118] András Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *1st International conference on Simulation tools and techniques for communications, networks and systems & workshops (Simutools)*, 2008.

[119] VDS3473. Vds website - cyber-security. URL http://vds-global.com/en/vds-cyber-security/.

[120] VESSEDIA. Project website, 2017. URL https://cordis.europa.eu/project/id/731453.

[121] Jorgiano Vidal, Florent de Lamotte, Guy Gogniat, Philippe Soulard, and Jean-Philippe Diguet. A co-design approach for embedded system modeling and code generation with UML and MARTE. In *Design, Automation and Test in Europe*, pages 226–231, Dresden, Germany, April 2009.

[122] Dolores Wallace and Roger U Fujii. Software verification and validation: Its role in computer assurance and its relationship with software project management standards. Technical report, 1989.

[123] Rui Wang, Jérémie Guiochet, and Gilles Motet. Confidence assessment framework for safety arguments. In *SAFECOMP*, 2017.

[124] Haissam Ziade, Rafic A. Ayoubi, and Raoul Velazco. A survey on fault injection techniques. 1: 171–186, 2004. http://ccis2k.org/iajit/PDF/vol.1,no.2/04-Hissam.pdf.

[125] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. Small world with high risks: A study of security threats in the npm ecosystem. *arXiv preprint arXiv:1902.09217*, 2019.

Chapter 8 Appendix

The appendix contains detailed tools description (requirements and specifications) in relation with the CAPE Assessment Framework described in T5.1.

NeSSoS risk assessment tool

User requirements description

The tool can be used in order to evaluate existing risks for the e-government system.

UC1	Evaluation of e-government risks
Description	A risk analyst assesses risks of the system.
Actors	• Risk analyst • A person responsible for security • IT (security) team • Various responsible persons (e.g., HR head, Physical security head, etc.)
Basic flow	Once the scope of the system is defined, the analyst (with the help of the tool) will identify the key assets to protect, existing security practices and the threats relevant for the considered system. Next, risks will be computed and possible improvements identified.

Table 46: NeSSoS - Use Cases

NeSSoS will identify the main security risks and the security controls which should be installed to ensure the relevant Security Assurance Level.

CR1	Identification of risks and relevant security controls
Description	A risk analyst identifies the main security risks and the security controls that are required for the the selected Security Assurance Level.
Actors	• Risk analyst/certifier • A person responsible for security • IT (security) team • Various responsible persons (e.g., HR head, Physical security head, etc.)
CR2	**Continuous risk assessment/certification**
Description	The identified risks are updated upon receiving new information.
Actors	• A monitoring module [external] • Risk consumer module (e.g., risk analyst/certifier)

Table 47: NeSSoS - Certification requirements

SR1	A stand alone on-line tool
Description	The tool is to be available on-line as a separate service
Actors	• Tool developers (CNR) • User (e.g., Risk Analyst)
Basic flow	Make the tool developed on-line.
SR2	**Identification of (additional) countermeasures**
Description	The tool should help to identify (additional) countermeasures to be installed.
Actors	• Tool developers (CNR) • User (e.g., Risk Analyst)
Basic flow	The tool computes the risk. Next, it helps the user to identify (additional) security controls to mitigate the identified risks.
SR3	**Continuous assessment**
Description	The tool should consume inputs and provide responses automatically.
Actors	• Tool developers (CNR) • Monitoring module • Risk consumer module
Basic flow	The tool API is triggered by an external monitoring module providing the information about the current state of security configuration. The tool re-computes the risk and sends it to the risk consumer.

Table 48: NeSSoS - Software requirements

Technical specifications

The NeSSoS tool consists of the following components:

- User interface. A web-based GUI for the user to insert the information about the system, as well as for receiving the results of risk assessment.
- Risk computation unit. The core unit which computes (and re-computes) the risks and identifies suggested countermeasures.
- Communication unit. A unit that manages communication of machine-readable information (e.g., receiving it from a monitoring module and sending it to a risk consumer module).
- Database. A database with the expert knowledge stored and used for simplifying the analysis.

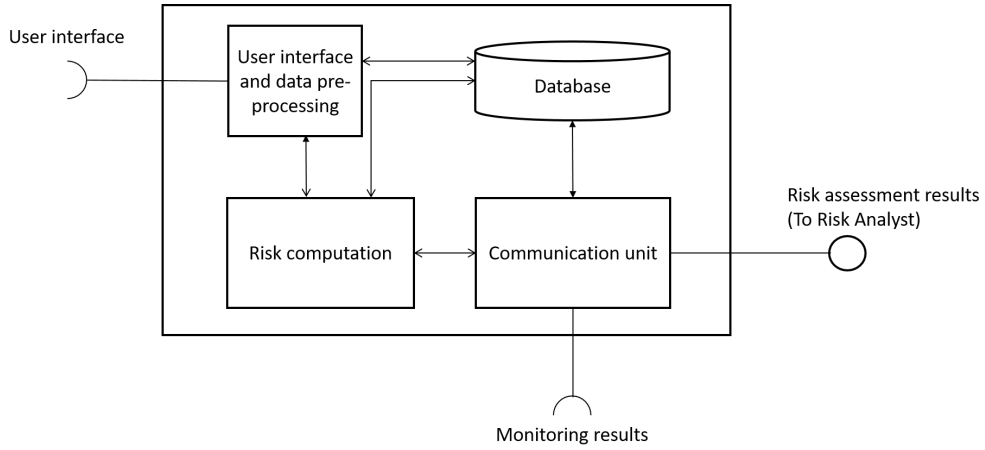


Figure 58: NeSSoS Risk Assessment Architecture

In short, the tool is to work as follows (see Figure 58). A user (e.g., Risk Analyst) enters the required data (the information about available security controls, key cyber assets and expected impact). The user interface passes this information to the risk computation unit, which, with the help of the knowledge stored in the database, identifies relevant threats and compute risk levels. This information is provided to the user through the user interface. If the tool is to be used for continuous assessment, the risk computation unit triggers the communication unit to send the aggregated risk information in a machine-readable format to any risk consumer module (e.g., a tool working on behalf of risk analyser). At this point, a monitoring module must be set up (based on the information previously generated by the NeSSoS tool (e.g., credentials and tokens for access, the ID for the system under evaluation, etc.). Once the monitoring module provides the up to date information about the state of security practices to the communication unit, risk is re-computed and the updated risk results are provided to the risk consumer module.

Development roadmap

Use Case	Architecture components	Realisation	Involved partners
UC1	User interface, risk computation, Database	Use the tool to compute the risks and suggest the additional controls	CNR, CINI

Table 49: NeSSoS - Use cases, realisations and architecture

Software verification and validation plan

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 automatically	Check if the tool is able to receive the input from the monitoring module and recompute risk levels without human intervention	Use the tool through the machine interface

Table 50: NeSSoS - Demo scenarios and verification methods

Buildwatch

User requirements description

The main use case of Buildwatch is the inspection of alternations to the host systems state, originating from a dependency during one of the aforementioned development processes. The inspections result may influence the decision on the roll-out of a dependency update to the dependent project.

UC1	Build Host State Introspection
Description	The build host state introspection entails the capture of the build host system state information generated by a software development process (i.e. build, test, install) and the reporting.
Actors	Developer
Basic flow	A build script is submitted to the Buildwatch Sandbox. The sandbox triggers the script and monitors the operating system calls made by the resulting process (and child processes) during the execution. The resulting is preprocessed and reported back for review by the developer. The review might either entail a manual assessment of the resulting host state changes or a automated comparison of different versions, of the monitored build process.

Table 51: Buildwatch - Use Cases

The Buildwatch Sandbox use case requires a deployed *dependency management process*. This process needs to define under which circumstances a dependency is to updated or held back, given an update is available. The Buildwatch Sandbox provides insight into the dependencies behavior. Based on this insight, a decision on holding an update back may be made. Hence, dependency updates have to be pulled into the project in a controlled fashion, that allows for a review of the changes within these dependencies.

SR1	Process Automation
Description	The process to be monitored needs to be fully automated.
Actors	• Developer • DevOps Engineer
Basic flow	The script that automates the process (i.e. build, test, install) is passed to the Buildwatch Sandbox and executed.
SR2	**Version Control**
Description	The software dependency needs to be version controlled. Either by commits or semantic versioning depending on monitoring granularity.
Actors	• Developer • DevOps Engineer
Basic flow	Comparing the differences with the *diff tool* is only feasible if two versions of the same product are evaluated. Both need to pass the sandbox, the *diff tool* is subsequently used to compute the differences.

Table 52: Buildwatch - Software requirements

Technical specifications

The Buildwatch system consist of three parts:

1. The Monitor
2. The Reporting Module
3. The Diff Tool

The Buildwatch Sandbox is based on the Cuckoo Sandbox [47] which has experimental support for Linux-based guest systems. Hence, the *monitor* is based on the Cuckoo agent. Recorded system calls are passed to the *reporting module*. The *reporting module* computes an abstraction based on cyber observable objects [31]. The *diff tool* allows the computation of differences between two of these reports in an object position (in terms of order) independent manner.

In order to use the Buildwatch Sandbox in a continuous integration pipeline supported development process, the required interfaces have to be added.

Development roadmap

Use Case	Architecture components	Realisation	Involved partners
UC1	Monitor	Extend Cuckoo monitoring capabilities for Linux-based guests	UBO
UC1	Reporting Module	Implement a custom reporting module the ingests the Cuckoo reported data and aggregates them to cyber observable objects	UBO
UC1	Diff tool	Implement a script that computes the differences between two Buildwatch reports	UBO
UC1	Integration	Implementation of the interfaces to ingest a ci job and report the result	UBO

Table 53: Buildwatch Sandbox - Use cases, realisations and architecture

Software verification and validation plan

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	CCCC
SR2	Compute 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.	CCCC

Table 54: Buildwatch Sandbox - Demo scenarios and verification methods

IDS and SIEM assessment tool

User requirements description

UC1	Synthetic traffic generation from existing traces
Description	Traffic is generated from features learned from network traffic traces including variations of protocol distribution and respective amplitudes over time. Protocol traffic can be generated from activities graph for improved realism. The size of the network traffic can be adapted to stress the network, while preserving the protocols distribution. The resulting synthetic trace is expected to confuse the system under test by hiding injected attack traffic, or at least evaluate the performance limitations of the system under test.
Actors	• Tester • Pentester
Basic flow	The tester recovers a sample network trace to be reproduced. The tool processes the trace to extract features in a privacy preserving manner, anonymizing sensitive identifiers. Once the features of the trace are learned, these features are given as input to the traffic generator to output a similar traffic respecting the protocols distribution and associated amplitudes. The generated traffic can also be amplified to fit stress test scenarios by arithmetically transforming the learned values of the model features. The pentester is then able to inject attacks during the play of the generated traffic to conceal her activities.
UC2	**Attack traffic mutation**
Description	Attack traffic is generated and confronted to the system under test to assess its ability to bypass it. Test results are fed back to the generator to improve the generation by mutating it. The resulting traffic should enable the discovery of possibly new attacks.
Actors	Tester
Basic flow	The tester starts the generator with or without an attack seed. The generated attack traffic is sent through the system under test. If the traffic is accepted, the generator reinforces the features that allowed the bypass. If the traffic is rejected, the generator degrades the weights of features that raised an alert. Traffic is then generated again with the newly computed features. The workflow is repeated until the situation does not evolve anymore. The outputs of the test may result in new attacks and therefore avenues to improve the robustness of the system under test.

Table 55: IDS - Use Cases

UR1.1	Availability of network traffic trace
Description	Network traffic traces for the system under test is available to be processed and reproduced.
Actors	System owner
UR1.2	**Privacy-preserving traffic generation**
Description	Selection of features avoid sensitive information to prevent learning it. If needed, some features are anonymized using
Actors	Data protection officer
UR2.1	**Availability of network intrusion traffic**
Description	Optionally, generation of intrusions is driven by network intrusion features or traces.
Actors	Pentester
UR2.2	**Interpretability of results**
Description	Bypasses enable the tester to understand how vulnerable the system under test is and to decide how to improve it.
Actors	Tester

Table 56: IDS - User Requirements

SR1.1	Metrics to measure realism
Description	Design of a set of metrics able to capture the realism of generated traffic, in order to guarantee the fairness of evaluation.
Actors	• Network expert • Tester
Basic flow	Study network traffic datasets to retain a number of features necessary to generate life-like traffic. Metrics should be derived from these features in order to measure the realism of generated traffic. The network expert assists the tester in selecting features and designing appropriate metrics.
SR1.2	**Anonymization functions**
Description	Select a set of functions to anonymize network trace to reproduce.
Actors	• Privacy expert • Tester
Basic flow	Traces to be reproduced are pre-processed to extract features necessary to the traffic generation stage. The privacy expert assists the tester in choosing features that are privacy-preserving. If needed, a set of anonymization functions is provided to process the trace before feature extraction.
SR2.1	**Metrics to measure malice.**
Description	Design of a set of metrics able to capture the malice of generated attack traffic, in order to guarantee the fairness of evaluation
Actors	• Pentester • Tester
Basic flow	Study attack traffic datasets to retain a number of features necessary to generate malicious life-like traffic. Metrics should be derived from these features in order to measure the malice of generated attack traffic. The pentester assists the tester in selecting features and designing appropriate metrics.
SR2.2	**Mutation functions**
Description	Select a set of functions to mutate attack traces.
Actors	• Pentester • Tester
Basic flow	Attack traces are mutated to evade the system under test. Mutation functions should not alter the malice of the attack trace. The pentester assists the tester in choosing mutation functions to transform the attack traces into seemingly innocuous network traces.

Table 57: IDS - Software requirements

Technical specifications

The proposed tool is constituted of at least two main modules:

- a network traffic parser to process captured traffic inputs;
- a network traffic generator to generate traces for IDS/SIEM evaluation.

In CAPE, IMT aims at improving the parser to extract new features that will enable a more faithful modelling of traffic traces. By reliably modelling real traffic traces, we believe that we will be able to generate more realistic network traffic. The models learned from a single traffic trace allows the generator to reproduce traffic for this particular trace. One particular challenge is the feasibility of producing real traffic features from certain model features.

The traffic generator takes as input the model features learned from the parser and generates a synthetic network traffic. A legacy and more heavy approach was to set up a number of agents supporting the protocols identified in the input and to have them generate a traffic envelope that fulfills the distribution of protocols and their respective amplitudes over time. While this work is interesting to pursue particularly with respect to agent orchestration, in CAPE, we aim at developing novel approaches based on generative networks. In particular, a first attempt using autoencoders – a type of neural networks that efficiently learn and reproduce inputs – has been studied. IMT will focus on improving the realism of generated traffic, with respect to packet contents and flow behaviour. Indeed, the autoencoder actually output a feature vector and not network traffic per se. IMT needs to develop a function (translator) to generate life-like traffic from the outputted feature vector.

Secondly, in order to generate malicious vector able to challenge the systems under evaluation (IDS, SIEM), IMT will develop a generative adversarial network-based (GAN) approach. Using a GAN, we aim at improving concurrently two aspects of the generated traffic: its realism and its malice, so that it becomes difficult for the system under evaluation to discriminate real, legitimate traffic from the malicious, synthetic one.

Finally, a human interface module should summarize the results of the test and assist the tester in identifying the weaknesses of the system under evaluation to make recommendations on how to improve it.

Development roadmap

Use Case	Architecture components	Realisation	Involved partners
UC1	Parser	Design new features set to extract	IMT
UC1	Generator	Design and implement autoencoder-based feature generator and translator	IMT
UC1	User interface	Produce explainable outputs for IDS/SIEM evaluation	IMT
UC2	Generator	Design and implement a GAN-based attack traffic generator	IMT
UC2	User interface	Produce explainable outputs for IDS/SIEM evaluation	IMT

Table 58: IDS - Use cases, realisations and architecture

Software verification and validation plan

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 59: IDS - Demo scenarios and verification methods

VaCSInE

User requirements description

UC1	**Enforce security policy on the edge infrastructure based on certification criteria**
Description	A security policy can be applied on the edge infrastructure to provide security properties, the SFC securing the edge infrastructure is adapted to satisfy the security policy. The security policy and corresponding NFCs will be described using the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [114] and aligned on a certification scheme property.
Actors	Security officer
Basic flow	Admin provides a security policy at the federation level, the SFC protecting the underlying infrastructure is automatically reconfigured to satisfy the security policy
UC2	**Continuous self-assessment for adaptive security with service function chaining**
Description	Monitor and detect: Ensure the edge infrastructure is protected through an automated reconfiguration of the service function chains . This can involve adding/removing or updating existing NFV's
Actors	security officer
Basic flow	The intrusion detection triggers a firewall re-configuration, remediation is checked against the system's security policy (derived from certification criteria) and applied to the SFC's protecting the infrastructure

Table 60: VaCSInE - Use Cases

CR1	Minimal network attack surface
Description	Ensure only the allowed network ports are open for the various components of the system.
Actors	• security officer • federated security agents • federated security manager
Basic flow	A security policy is applied to the edge system, which is then continuously stressed by simulated intrusions in a continuous integration sandbox environment. The Adaptive security component will detect the security breach, provide feedback to the security manager component that will adapt the security policy and apply it automatically to all the parts of the system.

Table 61: VaCSInE - Certification requirements

UC1 and UC2 will be developed in D5.3, UC2 will be further developed to study more complex SFC configurations before D5.4.

Foreshadow-VMM Assessment Tool (CNIT/University of Rome Tor Vergata)

User requirements description

UC1	Detect security vulnerabilities in Android applications packages
Description	The security analyst of an organization uses Approver to scan all mobile application packages, by submitting the APK packages either manually using the web front-end or using a set of REST APIs. Then, he uses the Web front-end to understand and assess findings and find the best mitigation option.
Actors	Security Analyst
Basic flow	The analysis of the Android application packages is triggered either manually, by uploading the APK package through the tool web interface or using the available REST APIs. Analysis results are made available through Approver's Web front-end. Furthermore, from the front-end, it is also possible to download a security report in PDF format.
UC2	**Detect security vulnerabilities in Android applications during development and suggest mitigations**
Description	An organization uses Approver to scan Android application projects. To do that, application developers can integrate Approver in their software development pipelines to detect security vulnerabilities in each build step of the application and to obtain mitigation strategies to solve those problems.
Actors	Developer & Security Analyst
Basic flow	The security analysis of the Android application is automatically triggered by the CI software, e.g., Jenkins. Once a new build is submitted, the Approver CI plugin will automatically report the security findings to the developer and, if available, automatically opens ad-hoc issues on the software repository.

Table 62: Approver - Use Cases

SR1.1	Implementation of Approver CI Plugins
Description	The current architecture of Approver supports the submission on the Android application package directly from the Web front-end or through REST APIs. However, it has no support for the automated submission of the application package during the development phase, especially in a continuous integration scenario. To this aim, the SR1 requirement is to develop the integration plugin for at least a state-of-the-art automation server (e.g. Jenkins [1]) and a version control system (e.g. Github [2]).
Actors	Not applicable
Basic flow	Description of the use case flow

Table 63: Approver - Software requirements

Technical specifications

At high-level, cf. Figure 59, 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 security of the application 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 i) view the detailed results of each application analyzed, ii) download all the artefacts produced during the analysis, and iii) download a security report which contains all the identified issues.

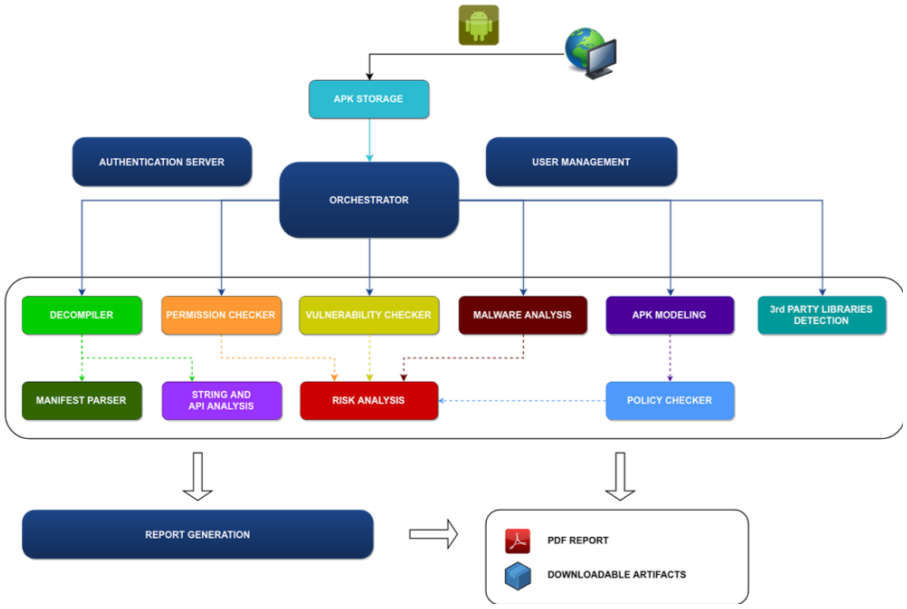


Figure 59: Approver architecture

Software verification and validation plan

The high-level development roadmap is to implement the software requirement SR1 within 2020-21 such that it can be demonstrated at project end, as explained in Table 64.

SR id	Description	Verification method	Demonstration scenario
UC1	Scan mobile app(s) packages related to e-government vertical	Check if scans succeed and findings are correct	e-government
UC2	Scan mobile app(s) projects developed for the e-government vertical	Check if scans succeed and findings are correct	e-government

Table 64: Approver - Demo scenarios and verification methods