Safety, Security and Performance Assessment of Security Countermeasures with SysML-Sec

Bastien SULTAN[1], Ludovic APVRILLE[1] and Philippe JAILLON[2]
[1] *LTCI, Télécom Paris, Institut Polytechnique de Paris, Sophia-Antipolis, France*
[2] *Mines Saint-Etienne, CEA-Tech, Centre CMP, F - 13541 Gardanne, France*
{*bastien.sultan, ludovic.apvrille*}*@telecom-paris.fr, jaillon@emse.fr*

Keywords: Formal verification, Impact Assessment, Countermeasures, Attacks, Safety, Security, Performance

Abstract: Deploying security countermeasures on Cyber-Physical Systems (CPS) can induce side-effects that can exceed their benefits. When CPS are safety-critical systems, performing efficiency and impact assessments of security countermeasures early in the design flow is essential. The paper introduces the W-Sec method, based on SysML-Sec. The W-Sec method consists in two interwoven formal modeling and verification cycles aiming at providing countermeasures with objective and quantitative efficiency and impact assessments in terms of safety, security and performance. The paper evaluates the W-Sec method with an autonomous rover swarm case-study, and finally discusses the method's strengths and weaknesses highlighted by the case-study results.

1 INTRODUCTION

Cyber-physical systems (CPS) can commonly act on their environment with actuators. As a result, a cyber attack on these systems can induce severe safety consequences on their environment. Thus, for these systems, bridging safety and security modeling and analysis is of prime importance to capture the inter-relations between safety and security, leading to better designs and easing maintenance. This joint safety/security modeling of complex systems has become a trending research topic over the past few years. In this context, the CAPE program[1] (one of the four scientific programs of the European Union Horizon 2020 project SPARTA) aims at providing methods for joint specification and assessment of security and safety properties for complex CPS.

Part of this program, the main contribution of the paper, is a new method (called "W-efficiency and impact assessment method", W-Sec method for short) targeting the selection of the right security countermeasures[2] for CPS. By relying on the modeling formalisms and tools supporting the SysML-Sec framework (Apvrille and Roudier, 2013), our contribution can now improve the impact assessment method introduced in (Sultan et al., 2018). The W-Sec method brings two main improvements to this method: (i) reducing models complexity while enhancing the model precision for hardware and security aspects and (ii) widening its assessment basis through more fine-grained security analyses. Formal models of W-Sec include two abstraction levels: component and overall system levels, from which formal verification of safety, security and performance can be performed.

The rest of the paper is organized as follows. Section 2 gives an overview of the related works. Then Sect. 3 introduces the W-Sec method. Section 4 presents the autonomous rover swarm case-study, including the attack scenarios and countermeasures, we used to evaluate the W-Sec method. It also presents the formal models we have designed: (1) a joint rover hardware and software model, and (2) a platoon model allowing to represent rovers' high-level behavior and interactions within the swarm. Last, Sect. 5 discusses the results of the W-Sec method applied to the rovers platoon case-study.

2 RELATED WORKS

The interest of assessing efficiency and impact of security countermeasures before their deployment has

[1] *Continuous Assessment in Polymorphous Environments.*

[2] In this paper, a *security countermeasure* is any modification brought to a system in order to mitigate one or several vulnerabilities. This modification can be related to the system's software, hardware, processes, and/or to its physical, logical and network architecture.

been discussed for a long time (Brykczynski and Small, 2003; Nicol, 2005) and several ways to select countermeasures have been proposed. (Nespoli et al., 2017) provides a survey on the optimal countermeasures selection methods proposed between 2012 and 2016. Regarding the countermeasures' (negative) impacts, these approaches mainly focus on the monetary cost of the countermeasures, yet several of them express the impacts in terms of system downtime or impacts on the provided services, e.g. in terms of confidentiality, integrity, availability and performance. Depending on the methods, these impacts can be used as inputs of the selection method (thus they are not computed on the basis of the countermeasure description but chosen on the basis of a human analysis), or computed. However, even if the "collateral damage" (Gonzalez-Granadillo et al., 2015) are assessed by some of the surveyed approaches, none of them seem to allow for a precise enough countermeasures impact assessment with respect to the behavior of the system, which is critical regarding CPS. For instance, when assessing a software patch affecting the driving controller of an autonomous car, it is crucial to quantify how the speed or steering set points can be modified as well as to evaluate the availability of the controller. Incidentally, to the best of our knowledge, few research works published after this survey address the finding of precise and objective impact assessments for countermeasures.

Among them, (Sultan et al., 2018) proposes a formal verification-based impact assessment method for security countermeasures, in the context of naval systems. This method relies on a network of UPPAAL timed automata (NTA) (Behrmann et al., 2004) used to model a cyber-physical system. When a vulnerability affecting the modeled system is discovered, NTA is mutated into a set of mutant NTAs representing (i) the vulnerable system, (ii) the vulnerable system enhanced with security countermeasures mitigating the vulnerability, (iii) the realization of successful attacks on the original vulnerable system, and (iv) the realization of these attacks on the "patched" systems. Afterwards, the NTA (modeling the CPS) and the mutant NTAs (modeling the CPS affected by the cyber events) are model-checked against a set of properties: the impacts of the cyber events and the efficiency of the countermeasures can then be deduced by comparing the model-checks results. However, as the authors explain in (Sultan, 2020), the underlying modeling framework lacks in expressiveness with respect to data security aspects (e.g., data confidentiality is modeled in a simplistic way by a boolean variable depicting an illegitimate access to the the component which processes the data). Therefore, the relative security properties verification results may be not accurate enough. In addition, establishing a fine-grained modeling with timed automata can be time-consuming and error-prone, and the resulting models can be complex to understand as they encompass heterogeneous aspects in a single modeling view.

Addressing these flaws requires requires more appropriate modeling formalisms and tools. Providing a framework for designing safe and secure embedded systems, the SysML-Sec method (Apvrille and Roudier, 2013) relies on an enhanced SysML-based modeling formalism that allows for modeling high-level system architectural and behavioral aspects, as well as fine-grained hardware ones. Moreover, SysML-Sec is tailored to produce joint safety, security and performance analyses (Apvrille and Li, 2019) and is fully supported by the toolkit TTool, that provides the system designers with a graphical and easy-to-use modeling and verification interface. In addition, it also provides distinct modeling views that can help simplifying the system models. For these reasons, we believe that the SysML-Sec language and TTool are the best underlying formalism and toolkit for enhancing the method proposed in (Sultan et al., 2018). Furthermore, another reason for choosing this formalism and tools is that our contribution can also complement the SysML-Sec method. Indeed, the attack model considered by this method is the Dolev-Yao one (Dolev and Yao, 1983), thus other kinds of attacks such as "sequence[s] of exploitation of vulnerabilities of several components" (Apvrille and Roudier, 2013) are considered out of scope. Thanks to the introduction of the modular attack scenarios like in (Sultan et al., 2018), our contribution then widens the considered attack corpus.

3 W-SEC METHOD

The method introduced in this section (see Fig. 1), called W-efficiency and impact assessment method (or W-Sec method for short), relies on two interwoven modeling and verification cycles aiming at (i) designing relevant models of a complex system and (ii) providing efficiency and impact assessments of countermeasures in terms of safety, security and performance. The following paragraphs describe its successive stages.

3.1 System modeling

Like in (Sultan et al., 2018), a comprehensive modeling of the whole system is first built. However, unlike this approach, our modeling approach does not rely on

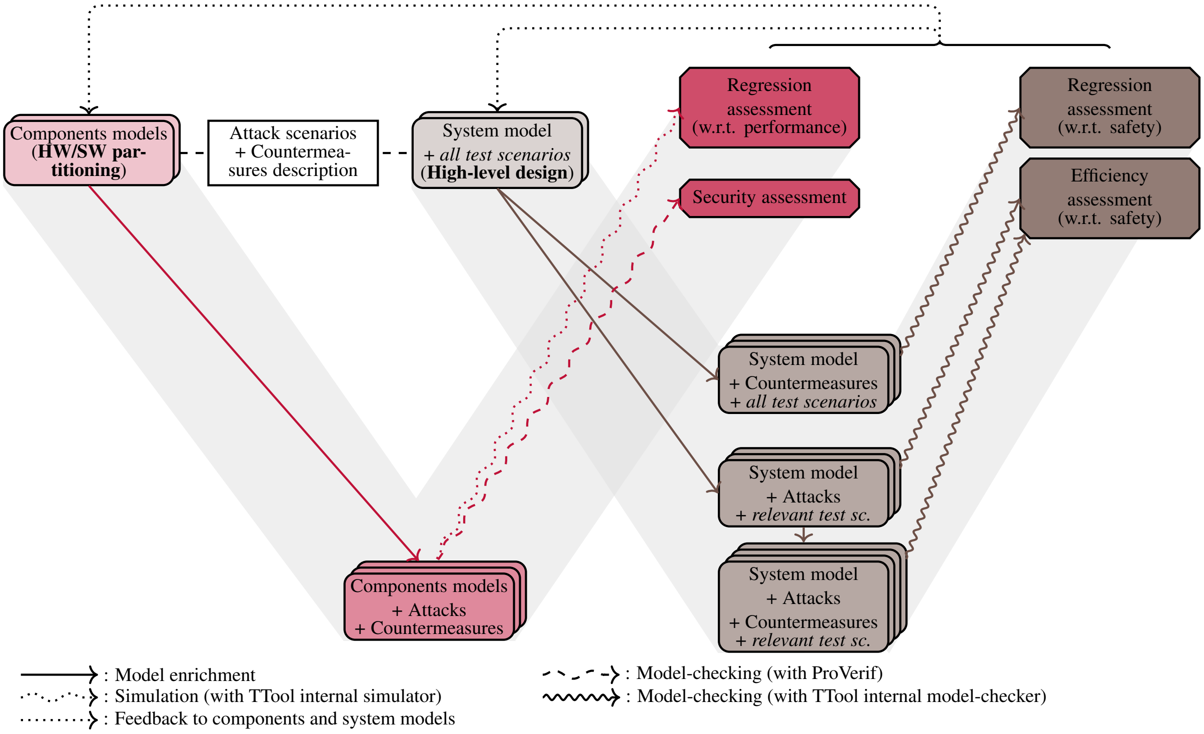


Figure 1: The W-Efficiency and Impact Assessment Method

a single network of timed automata (NTA). The single model approach indeed has two main drawbacks: first, it requires the modeling of heterogeneous items (e.g. software and hardware) and aspects (e.g., data security and system safety) with the same formalism. As a result, the modeling effort can be substantial and the model can lack in precision with respect to one of the modeled aspects, for instance data security as discussed in Sect. 2. Second, formal verification or simulation of such a model may be pointlessly complexified when evaluating safety, security and performance properties (e.g., the output values of an algorithm can be needed when analyzing its safety, while they can be useless when assessing its performance). To tackle these two issues, we have chosen to rely on two distinct modeling views provided by TTool and used in the SysML-Sec method, thus capturing the systems high-level and low-level aspects with more precision and facilitating the separation of concerns. One view targets the hardware/software (HW/SW) partitioning at a component level. We use it to precisely model the system's components, and we have chosen to carry out the performance and security assessments in this view: hardware modeling is obviously needed to assess a system's performance, and veryfing the security properties we want to assess (i.e., integrity, authenticity and confidentiality of data when transferred between two components or processed by a component) means verifying a resistance to attacks, including hardware-related attacks (e.g. putting a probe on a bus). The other view is dedicated to high-level system design. We use it to model the system overall behavior and architecture, and to perform the safety assessments.

1. The HW/SW partitioning view is used to model the system's components low-level application architecture and behavior, as well as their supporting hardware platforms. In this view, a component is modeled thanks to three models (Enrici et al., 2017) depicting its embedded software[3], its hardware platform, and the communication protocols it uses for data transfer between its hardware subcomponents (i.e., DMA transfer protocols). These three models are then linked in a fourth one (the *mapping* model) which assigns software tasks to CPUs, communication protocols between memories, etc. In addition, in order to assess the external communications security,

[3] Note that the software model focuses both on the exchange of information between applicative tasks and on the computational cost of algorithms (i.e. actual arithmetic operations are not modeled). For instance, if a task receives a value and then performs a multiplication by 2, the behavioral diagram of this task contains a read action in a channel followed by a one-multiplication complexity operator.

we integrate a simple model of the communication interfaces with the components modeling (e.g., a socket in the software model and a network interface controller in the hardware model) belonging to the external components with which the component communicates. Thus, the models of this view are built on the basis of the information that is necessary to precisely depict the software and hardware architecture of the modeled components, as well as the low-level security (i.e., cryptographic algorithms, privacy of the hardware buses, etc.) and performance (e.g., algorithm computational cost and hardware platform specification) aspects. Finally, this view also features the security and performance properties. These properties are established in parallel with the HW/SW partitioning models design.

2. The High-level system design view is used to model the system high-level architecture and behavior, with a focus on the interactions between system components and on the system's descriptive variables (e.g., for a rover, its speed and coordinates) evolution. Then the relevant arithmetic parts of the components algorithms can be modeled and even system dynamics (e.g. simplified modeling of speed and position) can be abstracted through integer variables[4]. Therefore, the information basis used to build the system model encompasses software architectural and functional information at system-level, and, if needed, at component-level in order to depict the high-level behavior of the components algorithms (i.e., the evolution of their output parameters depending on their inputs). In other words, we exclude from these models the hardware, the low-level security and algorithmic complexity aspects. In addition, the safety properties, established in parallel with the system model design, are captured in this view. Note that if several test scenarios are needed to evaluate all these properties, then several system models must be built, e.g. one per test scenario. For instance, if we want to check if the gap between two rovers in a swarm is always positive regardless of the swarm's speed variations, three models will be designed: a model in which the swarm keeps a constant speed, another one in which it accelerates, and a third one in which it decelerates.

Last, note that in the current state of our research, the semantic links between the HW/SW partitioning models and the high-level system design models are not explicit, thus it falls to the user to ensure the consistence of the models of both views while designing them.

3.2 Integrating attacks and countermeasures through enrichment of models

The second stage of our method consists in integrating the attacks and countermeasures with the models of both views (high-level system design view, HW/SW partitioning view). This integration consists in modeling the attacks and countermeasures in SysML-Sec, and then, in the current state of our works, to manually compose these new models with the pre-existing system and components models. For instance, a countermeasure consisting in encrypting a given communication channel can be modeled with two *encrypt* and *decrypt* SysML-Sec actions: this countermeasure is thus integrated to the models by adding the *encrypt*/*decrypt* actions before and after the relevant pre-existing *send* and *receive* actions in the pertinent SysML-Sec activity diagrams. In the same way, attacks can be modeled through a SysML-Sec block (which activity diagram models the successive attack stages) that is then bound to the relevant pre-existing blocks in the enriched models, and if needed through a modification of the state machines of these blocks (in order to model the behaviors that are now possible due to the attack). Note that due to the absence of integer variables in the HW/SW partitioning view, the repercussions of attacks on the outputs of the system algorithms, or on the system dynamics, can only be modeled in the high-level system design view.

Like the NTA in (Sultan et al., 2018), the high-level system models are turned into the following three classes of models (see Fig. 1):

- System models with countermeasures. As we want to assess the potential regression due to the countermeasures with respect to each test scenario, we produce one model per countermeasure and per scenario, i.e., if we have m test scenarios and n countermeasures, $m \times n$ models are built.
- System models with attacks. For each attack, we select the relevant test scenarios[5] then one model is produced per selected test scenario.
- System models with attacks and countermeasures. For each countermeasure (as a patch can actually mitigate several attacks), we build a "patched" model for every element of the *system models with attacks* subset. In other words, if we have p system models with attacks and n countermeasures, $p \times n$ models are built.

[4] Note however that only simple dynamics can be modeled in this way.

[5] i.e., the system configurations on which we want to assess (i) the attack safety impacts and (ii) the related countermeasures efficiency).

In addition, the components models of the HW/SW partitioning view are also turned into a set of new models. For each countermeasure, and for each component targeted by the countermeasure, the component's software model is enriched to integrate the countermeasure. Then, an attack scenario leading to the countermeasure triggering is integrated to this new model.

3.3 Performing safety, security and performance assessment

The third and last stage of the method consists in performing simulation and formal verification of the previously enriched models. First, the system models are checked with the TTool internal model-checker (Calvino and Apvrille, 2021) against the safety properties. As in (Sultan et al., 2018), these safety verification operations aim at:

- evaluating functional regressions induced by countermeasures by comparing the results of the verifications of the *system models with countermeasures* and the "nominal" system models designed at the first step.
- and then, from the comparison of the results of the model-checking of the *system models with attacks* and the *system models with attacks and countermeasures*, assessing the countermeasures ability to mitigate the attacks.

Since in our method the verification of the system models (thus, the two aforementioned regression and efficiency assessments) is only safety-focused, we perform further operations on the components models. First, security verifications are performed with ProVerif (Blanchet, 2001) thanks to an integrated *HW/SW partitioning models to ProVerif specification* translation (Li, 2018; Lugou et al., 2016). Second, simulations are performed with the TTool HW/SW simulator (Apvrille et al., 2006). These simulations provide, both for the "nominal" component model and for the component with countermeasures models, a set of performance assessments (e.g., the number of CPU clock cycles for a given algorithm). Thanks to the comparison of these results the additional complexity of the modeled countermeasures can be evaluated, with respect to the hardware platform and in terms of clock cycles or elapsed time.

At the end of this stage, we obtain a set of evaluations that enables to determine the subset of the input attacks that can be effective on the system, and to determine the optimal countermeasure or combination of countermeasures. If the impact and efficiency assessments show that no countermeasure is satisfactory, the models of both views can incrementally be modified and re-assessed until they verify the desired properties: in this way, the modeled countermeasures can be improved and then modified/developed to be deployed on the system.

3.4 Improvements brought by the method

Table 1: Comparison with SysML-Sec and (Sultan et al., 2018)

Modeling formalism	From SysML-Sec and (Enrici et al., 2017)
Two-views modeling	From SysML-Sec
Separation in two views of safety vs. security and performance aspects	W-Sec method contribution
Use of HW/SW partitioning view for low-level component modeling	W-Sec method contribution
Attacks and countermeasures modeling through model enrichment	From (Sultan et al., 2018)
Impact assessment approach	From (Sultan et al., 2018)

As explained hereinabove, the W-Sec method uses elements from SysML-Sec and (Sultan et al., 2018) (see Table 1). By merging these two approaches, the method brings several improvements to each of them. Indeed, with regards to (Sultan et al., 2018):

- The W-Sec method reduces the models complexity because it does not rely on a single NTA but on several models that can separately be simulated and verified. These models are based on two distinct views which only contain the information needed for their respective purposes (i.e., safety or security and performance assessment), and do not aggregate all this information in a single view.
- Hardware modeling relies on configurable templates already defined in TTool (CPU, memories, DMA, etc.) so it helps reducing the modeling time and effort, while giving more precision to the models with respect to the NTA approach.
- Low-level security aspects can be captured in a more fine-grained way. In addition, this low-level security modeling is facilitated thanks to TTool predefined security-related patterns (e.g., cryptographic algorithm models or hardware firewall blocks).

- Thanks to the simulation and verification tools provided by TTool, our approach assesses the impact of countermeasures and attacks, with respect to a widened property basis. Indeed, we can now evaluate fine and low-level security properties (e.g., related to the integrity of a data transfer, or the confidentiality of a component data).

Furthermore, the W-Sec method completes the SysML-Sec method thanks to the integration of modular attack scenarios like in (Sultan et al., 2018). When designing a countermeasure, the W-Sec method can therefore be used before the SysML-Sec design stage.

4 THE AUTONOMOUS ROVER PLATOON CASE-STUDY

4.1 Description of the Case-Study

4.1.1 The Platoon and its Rovers

The case-study relies on the SPARTA/CAPE Connected and Cooperative Cars system (Dupont et al., 2020). In concrete terms, we consider a platoon composed of three vehicles (one leader and two followers) driving in cooperative adaptative cruise-control (CACC) mode. The three vehicles we model are autonomous rovers designed by Fortiss (Dupont et al., 2020; Martinez et al., 2021; Lúcio et al., 2018). The architecture of these rovers relies on two Raspberry Pi 3B+ (one for the driving algorithms and the other one for processing images produced by the camera), three kinds of sensors (a Pi camera, a LIDAR and ultrasonic distance sensors), a DC motor and a steering servo motor. In a CACC platoon mode, a rover can act as a leader. In that case, the rover regulates its speed and trajectory using information received by its sensors. If a rover is a follower, then it has to adapt its speed according to the speed value received from the leader and gap information received from its sensors, and its trajectory according to the sensed information. It shall also brake whenever it receives an emergency brake (EB) message from the leader. Note that in our model we assume that a follower only relies on the speed value received from the leader to adapt its speed.

In order to assess the impact of attacks and countermeasures on safety, we have selected three scenarios derived from the *Basic Scenario* described in (Martinez et al., 2021) ("the platoon [...] navigates on a straight road" and "the goal of [an attacker] is to cause a crash between two legitimate vehicles"). We also assume that communications between vehicles are not signed nor timestamped, and that the leader only sends speed update or emergency brake (EB) messages to the followers. Our three platooning scenarios are depicted in Fig. 2 and can be summarized as follows:

(a) a simple scenario where the three rovers run in a straight lane. The leader (Rover1) continuously sends its speed value to the followers (Rover2 and Rover3) so that they can adapt their speed on the basis on the received value. In this scenario, the leader can keep a constant speed (scenario (a_1)), decelerate (scenario (a_2)) or accelerate (scenario (a_3)).

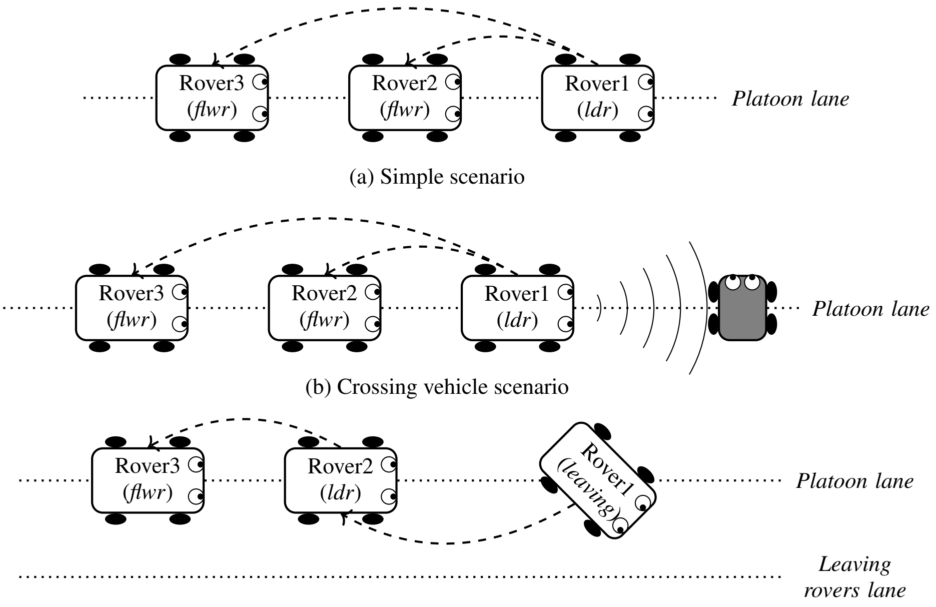
(b) a scenario where a vehicle crosses the platoon lane in front of the leader. In the first steps of the scenario, the leader sends its speed value to the followers like in scenario (a). Then, when the vehicle crosses the lane, the leader detects it, brakes and sends EB messages to the followers. Last, when the vehicle has finished to cross the lane, the leader restarts and sends its speed value to the followers.

(c) a scenario where the leader (Rover1) leaves the platoon. It joins the *leaving rovers lane* and sends a leaving message to the followers. The immediate following vehicle (Rover2) becomes the new leader and continuously sends its speed value to its follower (Rover3).

4.1.2 Attack Scenarios and Countermeasures

In addition to these platooning scenarios, we have implemented four of the attack scenarios described in (Martinez et al., 2021). We consider a Dolev-Yao (Dolev and Yao, 1983) adversary model: an attacker can thus eavesdrop, modify, block or inject messages. Attacks are:

- **Attack 1**: the attacker modifies the speed update messages sent by the leader to the followers by adding a large deviation to the legitimate value.
- **Attack 2**: the attacker modifies the speed update messages sent by the leader to the followers by adding a small deviation to the legitimate value.
- **Attack 3**: the attacker injects false EB messages bound for the first follower. In our implementation, the attacker performs this attack by changing the speed update messages sent to the first follower into EB messages.
- **Attack 4**: the attacker blocks the EB messages sent by the leader to the followers. In our implementation, this attack is performed by changing the EB messages into speed update messages.



(a) Simple scenario

(b) Crossing vehicle scenario

(c) Platoon splitting scenario

Figure 2: CACC Platoon Scenarios

We also have implemented several countermeasures in order to mitigate these attacks. They can be classified into two groups:

- **Plausibility checks**

  Speed Plausibility Check: this countermeasure has been introduced in (Martinez et al., 2021) and consists in checking if the speed update value received by the followers "deviates from a given percentage w.r.t. the average of the last *n* speed values received by the vehicle" (Martinez et al., 2021). If it is the case, the message is discarded and the rover relies on the last legitimate received speed value to elaborate its actuators commands.

  Emergency Brake Plausibility Check: (Martinez et al., 2021) mentions a sensor-based plausibility check. We have chosen to implement such a countermeasure, working as follows: (1) when the first EB message is received, the follower checks if the three next gap measurements decrease. If it is not the case, the EB message is discarded. If it is the case, the EB message is considered as legitimate, the rover brakes and the current gap measurement is recorded. (2) When the next EB message is received, it is considered as legitimate if the current gap measurement is less than or equal to the previously recorded gap measurement[6]. If it is not the case, the recorded gap measurement will be set to 0 and the successive gap checks will be performed as for the first iteration.

  Emergency Brake Plausibility Check with Gap Check: as the previous countermeasure can result in platoon crashes under certain conditions (see Sect. 5), we have designed an alternative version enhanced with an additional safety check. Before each speed order is sent to the actuators, this safety check verifies if the gap measured ahead of the rover is greater than a given threshold. If it is not the case, an emergency brake is performed.

- **Cryptographic countermeasures**: these countermeasures have been proposed by (Li, 2018), in a context of autonomous connected vehicles security.

  MAC: for each message sent by the leader, a message authentication code (MAC) is added in order to allow the followers to check the messages authenticity and integrity.

  Symmetric Encryption with Nonce Exchange: for each message sent by the leader to a given follower, a nonce is exchanged and then the message is encrypted with a symmetric encryption algorithm (AES) and the nonce is concatenated. This countermeasure ensures confidentiality of the messages, provides anti-replay protection and allows the followers to check their authenticity and integrity.

  If the integrity/authenticity check fails, the rover relies on the last legitimate received message to elaborate its actuators commands.

[6] This check is performed in order to avoid a crash between the leader and the first follower if the leader sends two successive legitimate EB messages (e.g., if a slow vehicle crosses the road ahead of the leader).

4.2 Modeling the Platoon

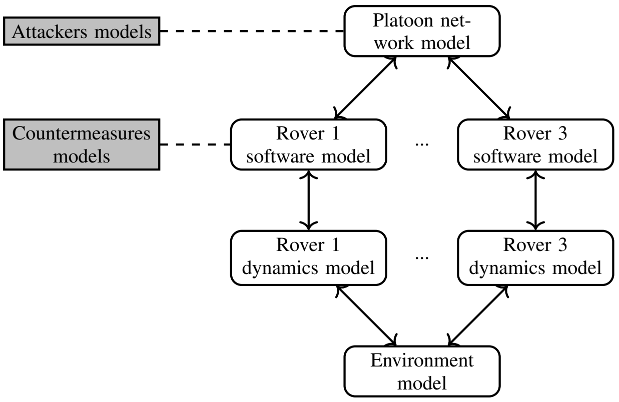


Figure 3: Structure of the platoon model (High-level system design profile)

The platoon model we have built (in the high-level system design view) relies on four kinds of SysML-Sec blocks (see Fig. 3):

- **The rover software models**. These blocks depict the high-level logic behavior of the rovers: their state-machines model the network communications and the driving algorithms. But since they are high-level models, the algorithms are not represented in a fine-grained way: e.g., we do not model the PID[7] control loops and we consider that, for a given controlled parameter (e.g., speed), the command values are directly equal to the final set point values and the actuators directly reach these command values.
- **The rover dynamics models**, used to abstract the evolution of the rover's coordinates and gap value (ahead of the robot). For each rover, the software model block and the dynamics model block exchange signals through their SysML ports, in order to simulate the rover's sensors acquisition cycle.
- **The platoon network model**. This block models the communications between the leader and its followers. To this end, the *rover software model* blocks exchange signals with the network block through their bound SysML ports: in our model, a communication between two robots $Rover_a$ and $Rover_b$ will be modeled by two successive signals exchanges $Rover_a$ *software model* → *platoon network model* → $Rover_b$ *software model*.
- **The environment model**, intended to link the *rover dynamics model* blocks through bound SysML ports. Thanks to this block, the rover dynamics models exchange their coordinate values in order to update their gap value.

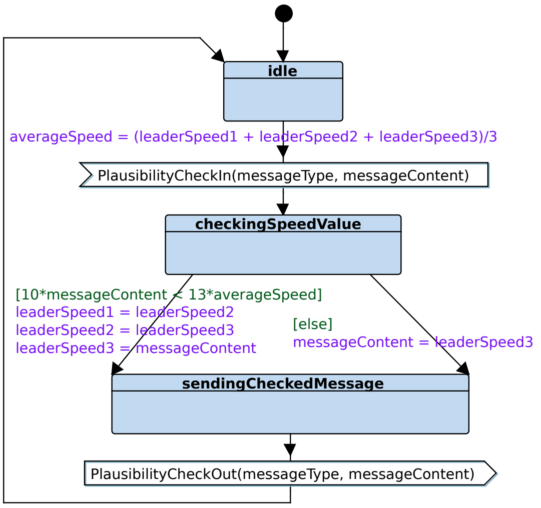

idle
averageSpeed = (leaderSpeed1 + leaderSpeed2 + leaderSpeed3)/3
PlausibilityCheckIn(messageType, messageContent)
checkingSpeedValue
[10*messageContent < 13*averageSpeed]
leaderSpeed1 = leaderSpeed2
leaderSpeed2 = leaderSpeed3
leaderSpeed3 = messageContent
[else]
messageContent = leaderSpeed3
sendingCheckedMessage
PlausibilityCheckOut(messageType, messageContent)

Figure 4: State machine of a block modeling the Speed Plausibility Check

Note that in order to make the platoon model easily scalable, we have defined two libraries (i.e., block patterns that can be instantiated through "regular" blocks) depicting the rover-related models. Actually, the *rover software model* and *rover dynamics model* blocks are instances of these two libraries.

In addition, the attacks have been modeled by enriching the *platoon network model* block, and the countermeasures have been depicted through enrichments of the *rover software model* library (see Fig. 3), or by adding countermeasure blocks bound to the *rover software model* instances (see Fig. 4). Given our test scenarios, attacks and countermeasures, we have built 50 distinct platoon models.

4.3 Modeling the Rover

As seen in Sect. 3, a SysML-Sec HW/SW partitioning model consists in four distinct "sub-models". Therefore, we have designed the following models in order to depict our rover:

- **The rover software model**. We have modeled the driving algorithms, including message exchanges and interpretation, data fusion from the sensors, PID controllers for speed and heading and emergency brake controller. The software viewpoint also includes an abstraction of the sensors' behavior (i.e., continuous data sending) as well as an abstraction of the actuators behavior (i.e., continuous data reception).
- **The rover platform model**. Our hardware model mainly focuses on the Raspberry Pi dedicated to ex-

[7] **P**roportional, **I**ntegral, **D**erivative control, elaborating a command relying on the current, past and foreseen error values.

ecute the driving algorithms[8]. We have modeled the Raspberry SoC (with a 4-core CPU, a GPU, a DMA controller, a bus and a memory block) as well as the sensors and actuators (with CPU, bus and memory blocks). The I/O devices (sensors and actuators) are linked to the SoC through a main bus and bridges.

- **The communication model**. We assume that the Raspberry Pi SoC and the I/O devices communicate through DMA transfer protocols.
- **The mapping model**, bridging the three previous ones.

In addition, we have modeled the attack scenarios and countermeasures by adding tasks to the rover software model and by modifying the pre-existing tasks. Note that the countermeasures algorithms have been modeled according to the worst-case performance, i.e. if, for a given algorithm, the number of executed instructions depend on the value of a given input variable, we always consider that this variable is set to a value leading to the highest number of executed instructions. Given our test scenarios, attacks and countermeasures, we have built 6 distinct rover models.

5 RESULTS AND DISCUSSION

Table 2: Verification results for security properties

Property	*Weak*	*Strong*
Contermeasure	*auth.*	*auth.*
No countermeasure	✗	✗
MAC	✓	✗
Symm. Encr. + Nonce	✓	✓
Speed Plausibility Check	✗	✗
Emergency Brake PC	✗	✗
EB PC with Gap Check	✗	✗

This section discusses the application of the third stage of our method, i.e. assessing the impacts and efficiency of countermeasures thanks to the simulation and verification of these models. We have verified and simulated the behavior of our rovers on 21 consecutive operating cycles[9]. This number of operating cycles has been chosen to suit the hardware limitations of the computer we used to verify the models: as the models of the high-level system design view rely on unbounded integer variables (coordinates and gaps), the verification computational cost increases with the number of rovers operating cycles. Apart from limiting the number of operating cycles, other strategies could have been used to handle the combinatory explosion, such as bounding the evolution of variables. However, the 21 operating cycles were enough to simulate all platooning scenarios, to identify one rover collision for each attack, and to observe impacts of the selected countermeasures, thus providing the safety, security and performance results to evaluate our approach.

5.1 Performance Assessment

Performance assessment aims at comparing the overhead cost, in terms of operating cycles duration, of the countermeasures. In order to carry out performance measurements, two breakpoints have been added in the rover software model of the HW/SW partitioning view. The first one is placed on the “read in channel” action modeling a leader message reception, and the second one on the “read in channel” action modeling the reception of a power command by the actuators. Thus, if the first breakpoint is reached after n ns and the second breakpoint is reached after m ns, then it takes $m-n$ ns for the rover to perform an operating cycle. We have then measured this difference for each of the 21 operating cycles and for each rover configuration (i.e. the “unpatched” and the 5 “patched” configurations).

To have a fair comparison, we have also enforced the following for the patched rover model:

- to integrate to each rover model an attack scenario systematically leading to the use the countermeasure
- to model the countermeasures so that they always lead to the driving actions (i.e., always elaborate speed and trajectory commands).

Table 3: Average elapsed time between the two consecutive breakpoints (ns)

Without any countermeasure	274
With MAC	1,019
With Symm. Encryption + Nonce	646
With Speed Plausibility Check	357
With Emergency Brake PC	344
With EB PC with Gap Check	391

Table 3 gives, for each rover configuration (i.e., with or without countermeasures), the average duration of the operating cycles, and thus the induced overhead. These durations are consistent with our expectations: the cryptographic countermeasures induce more operations, thus they imply has a higher computation cost than the plausibility checks. Yet,

[8] Since the countermeasures will be mapped to this Raspberry Pi, its SoC constitutes our target of evaluation.

[9] An *operating cycle* is the sequence of actions starting at the reception of sensed data and/or a leader message and ending at the next command execution by the actuators.

these overhead costs are all acceptable since acquisition cycles of the rover's sensors are far greater than these values (the minimal acquisition cycle being 3,704 µs (Dupont et al., 2020; Garmin, 2016)). Therefore, with respect to performance aspects, all the countermeasures are compatible with the performance requirements of the rovers.

5.2 Security Assessment

Given the chosen attack scenarios (i.e., modifications of the messages exchanged between the leader and the followers), we have decided to evaluate the security of the followers' incoming communications. To this end, we have integrated a simple software task simulating the leader's network behavior, and a simple leader hardware model on which this task is executed, with the models of the HW/SW partitioning view. This hardware model is linked to the rover's one through a public bus: this bus depicts the unprotected platoon network. The software task is linked to the rover's one through two communication channels. We have assessed the two following properties related to the downlink channel: *Weak authenticity* (i.e. integrity), and *Strong authenticity* (i.e. data integrity and data origin authenticity). Indeed, these two properties indicate, respectively, if an attacker could falsify a message, and if an attacker could falsify and/or replay a message, without the follower being able to detect any of these attacks.

The verification results are given in Table 2. They confirm that the implementation of the MAC countermeasure provides the rover's incoming network communications with weak authenticity, while the implementation of the Symmetric Encryption with Nonce Exchange ensures a strong authenticity. Therefore, with respect to the security requirements, the latter should be selected.

5.3 Safety Assessment

We have chosen to focus our safety assessment on the two most critical safety properties of the platoon: (i) the gap between Rover1 and Rover2 is always strictly positive and (ii) the gap between Rover2 and Rover3 is always strictly positive (i.e., no crash can occur between Rover1 and Rover2, and between Rover2 and Rover3). The verification of these properties enables to evaluate if the attacks (which aim is to cause a platoon crash) are successful on the rovers, if the countermeasures can actually mitigate them, and if they could lead to the worst regression (i.e., a crash under nominal driving conditions)[10]. Table 4 summarizes the verification results for each rover configuration. Each cell depicts the results for a given ⟨platooning scenario, attack scenario⟩ pair, and is divided into two subcells: the first subcell represents the property 1 verification, and the second the property 2 verification. If the subcell is green, then the property is verified; if the subcell is red, the property is not verified. For instance, Table 4 (a) indicates that when attack 3 occurs in platoon scenario (b), there is no crash between Rover1 and Rover2, but a crash occurs between Rover2 and Rover3. According to these results:

Table 4: Safety properties verification results

Attack Scenario	No att.	Att. 1	Att. 2	Att. 3	Att. 4
Scenario (a_1)	🟩🟩	🟥🟥	🟥🟩	🟩🟥	—
Scenario (a_2)	🟩🟩	🟥🟥	🟥🟩	—	—
Scenario (b)	🟩🟩	—	—	🟩🟥	🟥🟥
Scenario (c)	🟩🟩	—	—	—	—

(a) Without countermeasure

Attack Scenario	No att.	Att. 1	Att. 2	Att. 3	Att. 4
Scenario (a_1)	🟩🟩	🟩🟩	🟥🟩	🟩🟥	—
Scenario (a_2)	🟩🟩	🟥🟥	🟥🟩	—	—
Scenario (b)	🟩🟩	—	—	🟩🟥	🟥🟥
Scenario (c)	🟩🟩	—	—	—	—

(b) With Speed Plausibility Check

Attack Scenario	No att.	Att. 1	Att. 2	Att. 3	Att. 4
Scenario (a_1)	🟩🟩	🟥🟥	🟥🟩	🟩🟩	—
Scenario (a_2)	🟩🟩	🟥🟥	🟥🟩	—	—
Scenario (b)	🟩🟩	—	—	🟩🟥	🟥🟥
Scenario (c)	🟩🟩	—	—	—	—

(c) With Emergency Brake Plausibility Check

Attack Scenario	No att.	Att. 1	Att. 2	Att. 3	Att. 4
Scenario (a_1)	🟩🟩	🟩🟩	🟩🟩	🟩🟩	—
Scenario (a_2)	🟩🟩	🟩🟩	🟩🟩	—	—
Scenario (b)	🟩🟩	—	—	🟩🟩	🟩🟩
Scenario (c)	🟩🟩	—	—	—	—

(d) With Emergency Brake PC + Gap Check

Attack Scenario	No att.	Att. 1	Att. 2	Att. 3	Att. 4
Scenario (a_1)	🟩🟩	🟩🟩	🟩🟩	🟩🟩	—
Scenario (a_2)	🟩🟩	🟥🟥	🟥🟥	—	—
Scenario (b)	🟩🟩	—	—	🟩🟥	🟥🟥
Scenario (c)	🟩🟩	—	—	—	—

(e) With MAC or with Symm. Encr. + Nonce Exchange

None of the countermeasures leads to a safety regression under normal conditions (see the "No attack" column of each table).

Speed Plausibility Check only mitigates attack 1 in scenario (a_1). That seems consistent since this countermeasure makes the followers rely on the last legitimate leader message: as in scenario (a_2), the leader decelerates while the attack is carried out, the followers rely on a speed value higher than the actual leader speed.

EB Plausibility Check only mitigates attack 3 in scenario (a_1), but not in scenario (b). These verifica-

[10] However, these two properties are insufficient for performing a comprehensive safety regression assessment and for presenting cases where a countermeasure deteriorates safety.

tion results led us to design the next countermeasure.

EB Plausibility Check with Gap Check mitigates all the attacks. These results are obvious since the countermeasure makes the followers brake when their measured gap value is smaller than a given threshold.

The two cryptographic countermeasures mitigate attacks 1, 2 and 3 in scenario (a_1) but do not mitigate them, nor attack 4, in the other tested scenarios. That seems logical since they make the followers rely on the last legitimate leader message if they detect a received message alteration. Indeed, in ⟨scenario (b), attack 4⟩ Rover2 and Rover3 detect that the incoming speed message has been modified. But since they rely on the last legitimate message that has been a speed update message, they continue to run while Rover1 is braking when the intruder crosses the lane. Then in ⟨scenario (b), attack 3⟩ Rover2 detects that the incoming EB message has been modified. But after the intruder has crossed the lane, the last legitimate message received by Rover2 is an EB message. So it continues to brake and that leads to a collision with its follower Rover3. Finally, in ⟨scenario (a_2), attacks 1 and 2⟩ the followers rely on a last legitimate speed value higher than the actual leader speed as for Speed Plausibility Check. Note that for ⟨scenario (a_2), attack 2⟩, property 2 is not verified while it is verified for the "unpatched" rover (see Table 4 (a)). This seems to indicate a safety regression but it is actually due to the 21 operating cycles limitation: a simulation on 42 operating cycles also leads to a crash between Rover2 and Rover3 in the "no countermeasure" configuration. Thus, for ⟨scenario (a_2), attack 2⟩ the cryptographic countermeasures only make this crash occur sooner. This can easily be explained since in the "unpatched" configuration, Rover3 will adapt its speed on the basis of the received speed value which is the leader speed slightly increased by the attacker. For instance, if the leader runs at 10 mph then decelerates down to 5 mph, Rover2 and Rover3 will run at 12 mph and then decelerate down to 7 mph. But with cryptographic countermeasures enabled, Rover2 and Rover3 will elaborate their driving commands on the basis of the last legitimate received speed value, i.e. 10 mph: in the end, they will run faster than in the "unpatched" configuration so the collision between the leader and Rover2, and then Rover2 and Rover3, will occur sooner. The fact that the 21 operating cycles were not enough to observe this crash constitutes a weakness of our choice to rely on unbounded integer variables for modeling the rovers' dynamics.

5.4 Discussion

This platoon case-study is interesting to evaluate the W-Sec method, e.g. to analyze the relations between safety, security and performance requirements and mechanisms. Indeed, thanks to safety verification, we were able to show that all the attack scenarios were successful on the platoon and that the designed countermeasures were not as efficient as expected: for instance, the Speed Plausibility Check was designed to mitigate attack 1 but we showed that under certain conditions (i.e., if the leader decelerates while the attack is carried out), it can lead to a platoon crash. In addition, the method helped us in improving the EB Plausibility Check: safety assessment results showed that the improved EB Plausibility Check was more efficient, and performance assessment results showed that its overhead cost was still acceptable. This shows the importance of the joint safety-performance analysis, thus of the interleaving of the two modeling and verification cycles. Broadly speaking, these two examples show the interest of the W-Sec method for designing security countermeasures or merely assessing the impact of existing ones. Also, the case-study highlighted the interest of the W-Sec method regarding the performance assessment: thanks to the measurements, we were indeed able to establish that the rover model enhanced with countermeasures still respects performance properties. Finally, the case-study fully illustrates the relevance of joint safety-security analysis: the assessment of cryptographic countermeasures showed that even if the attacked communication channel is provided with integrity, authenticity and anti-replay protection, the attacks could still lead to a crash depending on the platooning scenario and on the countermeasures implementation. Here, the main implementation problem lies in the fact that the rover systematically relies on the last legitimate received message when it detects that the current message has been altered.

But this case-study also identified several remaining weaknesses of the method. Firstly, depending on the modeling choices for the models of the high-level system design view, we can face a combinatorial explosion when verifying safety properties. In our case-study, this was due to the unbounded integer variables used to model the rovers' coordinates and gap values. To tackle this issue, we have decided to limit the number of operating cycles. However, this leads to incomplete safety verifications with respect to the whole attack sequence: for instance, as explained in the previous subsection, depending on the ⟨platooning scenario, attack scenario⟩ pair, a crash between Rover2 and Rover3 can occur after 21 operating cycles. The simulation then helped to analyse and complete the verification results. Yet, this shows that it is important to analyze the scope of the verification results.

Secondly, if our safety property basis was relevant for assessing the impacts of the attacks and then the efficiency of the countermeasures, it was insufficient to obtain a comprehensive countermeasures safety regression assessment. In general, further safety properties are needed to perform a complete regression assessment. For instance, it should have been interesting to systematically check reachability and liveness properties for all states of the SysML-Sec state-machines, and to compare, for each countermeasure, the results of this verification performed on the "unpatched" and on the "patched" model.

Thirdly, as the model enrichment (i.e., their composition with attack scenarios and countermeasures models in the second stage of the method) is not automated, human intervention is needed to carry them out. Depending on the initial system and components models, attack scenarios and countermeasures, that could be time-consuming —especially as we produce up to three enriched system models per test scenario as explained in Sect. 3: for instance, we produced 47 enriched models for this case-study. However, the transformations were simple (add a block, change variables updates or guards in the state-machines, etc.) and as TTool provides the user with convenient functions (e.g., models or objects cloning), we were able to perform the needed models' enrichment quickly. Still, the reduction in the number of needed enriched models shall be studied in future works: for instance, we could design a single test scenario encompassing all the atomic test cases and so assess all the countermeasures at once.

Finally, if we can assess the safety and performance impacts of a wide variety of attacks, the security formal verification only evaluates the impact of a Dolev-Yao attacker. As a result, verifying the confidentiality of data processed by a component is done by verifying the confidentiality of data transfers within this component (i.e., between its subcomponents). That is enough to model eavesdropping attacks targeting the component's buses, yet we still cannot evaluate the *security* impact of attacks directly targeting a hardware subcomponent.

6 CONCLUSIONS

The W-Sec method, introduced in this paper, allows CPS designers and maintenance engineers to assess the efficiency and impacts of security countermeasures. It relies on two distinct modeling views helping in reducing the models complexity, and on formal methods for providing objective and quantitative assessments regarding three aspects: safety, security and performance. By combining their strengths, the W-Sec method enhances the method presented in (Sultan et al., 2018) and complements the SysML-Sec method. Its relevance has been illustrated by an autonomous rover platoon case-study.

Yet further works are needed to address its limitations. As mentioned in Sect. 5, we intend to study the reduction in the number of needed enriched models while still correctly assessing safety, security and performance impacts. We will also investigate the automation of the enrichment stage. Furthermore, we will work on the identification of the links between the two modeling views. Regarding the assessment stage, the security assessment of hardware attacks at subcomponent level, the automated generation of properties and the design of metrics allowing to easily compare the verification and simulation results are three improvement perspectives we intend to explore. Finally, we intend to evaluate the method on other case-studies that can help in comparing the assessment results with the impacts observed on real systems.

ACKNOWLEDGEMENTS

This work has been funded by the EU H2020 project SPARTA. We gratefully thank Fortiss and Yuri Gil Dantas for their kind help and support.

REFERENCES

Apvrille, L. and Li, L. W. (2019). Harmonizing safety, security and performance requirements in embedded systems. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1631–1636. IEEE.

Apvrille, L., Muhammad, W., Ameur-Boulifa, R., Coudert, S., and Pacalet, R. (2006). A uml-based environment for system design space exploration. In *2006 13th IEEE International Conference on Electronics, Circuits and Systems*, pages 1272–1275. IEEE.

Apvrille, L. and Roudier, Y. (2013). SysML-Sec: A SysML Environment for the Design and Development of Secure Embedded Systems. In *APCOSEC 2013*, Yokohama, Japan.

Behrmann, G., David, A., and Larsen, K. G. (2004). A tutorial on uppaal. *Formal methods for the design of real-time systems*, pages 200–236.

Blanchet, B. (2001). An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE*

Computer Security Foundations Workshop (CSFW-14), pages 82–96, Cape Breton, Nova Scotia, Canada. IEEE Computer Society.

Brykczynski, B. and Small, R. A. (2003). Reducing internet-based intrusions: Effective security patch management. *IEEE software*, 20(1):50–57.

Calvino, A. T. and Apvrille, L. (2021). Direct model-checking of sysml models.

Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208.

Dupont, S., Maroneze, A., Massonnet, P., Nigam, V., Plate, H., Sykosch, A., Cakmak, E., Thanasis, S., Jiménez, V., Amparan, E., Martinez, C., López, A., García-Alfaro, J., Segovia, M., Rubio-Hernan, J., Blanc, G., Debar, H., Carbone, R., Ranise, S., Verderame, L., Spaziani-Brunella, M., Yautsiukhin, A., Morgagni, A., Klein, J., Bissyande, T., and Samhi, J. (2020). Assessment specifications and roadmap. Technical report.

Enrici, A., Apvrille, L., and Pacalet, R. (2017). A model-driven engineering methodology to design parallel and distributed embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(2):1–25.

Garmin (2016). Lidar lite v3 operation manual and technical specifications. Technical report.

Gonzalez-Granadillo, G., Garcia-Alfaro, J., Alvarez, E., El-Barbori, M., and Debar, H. (2015). Selecting optimal countermeasures for attacks against critical systems using the attack volume model and the rori index. *Computers & Electrical Engineering*, 47:13–34.

Li, L. (2018). *Safe and secure model-driven design for embedded systems*. PhD thesis, Université Paris-Saclay.

Lúcio, L., Kanav, S., Bayha, A., and Eder, J. (2018). Controlling a virtual rover using AutoFOCUS3. In *Proceedings of the MDETools Workshop co-located with MODELS 2018*, volume 2245 of *CEUR Workshop Proceedings*, pages 356–365.

Lugou, F., Li, L. W., Apvrille, L., and Ameur-Boulifa, R. (2016). Sysml models and model transformation for security. In *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 331–338. IEEE.

Martinez, C., Maroneze, A., Massonnet, P., Dupont, S., Grandclaudon, J., Nigam, V., Dantas, Y.-G., Plate, H., Sykosch, A., Ohm, M., Cakmak, E., Athanasios, S., Jiménez, V., Amparan, E., López, A., Apvrille, L., Blanc, G., Debar, H., Bisegna, A., Carbone, R., Verderame, L., Ranise, S., Bernardinetti, G., Palamà, I., Pellegrini, A., Restuccia, G., Sirbu, G., Yautsiukhin, M. S.-B. A., Poretti, C., Klein, J., and Samhi, J. (2021). Demonstrators specifications. Technical report.

Nespoli, P., Papamartzivanos, D., Mármol, F. G., and Kambourakis, G. (2017). Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks. *IEEE Communications Surveys & Tutorials*, 20(2):1361–1396.

Nicol, D. (2005). Modeling and simulation in security evaluation. *IEEE Security & Privacy*, 3(5):71–74.

Sultan, B. (2020). *Maîtrise des correctifs de sécurité pour les systèmes navals*. PhD thesis, Ecole nationale supérieure Mines-Télécom Atlantique Bretagne Pays de la Loire.

Sultan, B., Dagnat, F., and Fontaine, C. (2018). A methodology to assess vulnerabilities and countermeasures impact on the missions of a naval system. In *Computer Security*, pages 63–76, Cham. Springer International Publishing.