W-Sec: a Model-Based Formal Method for Assessing the Impacts of Security Countermeasures

Bastien Sultan, Ludovic Apvrille, Philippe Jaillon, Sophie Coudert

W-Sec: a Model-Based Formal Method for Assessing the Impacts of Security Countermeasures

Bastien Sultan[1][0000−0002−5031−5794], Ludovic Apvrille[1][0000−0002−1167−4639], Philippe Jaillon[2], and Sophie Coudert[1]

[1] LTCI, Télécom Paris, Institut Polytechnique de Paris, Sophia-Antipolis, France
firstname.lastname@telecom-paris.fr
[2] Mines Saint-Etienne, CEA-Tech, Centre CMP, F–13541 Gardanne, France
firstname.lastname@emse.fr

Abstract. The chapter provides a detailed description of W-Sec, a formal model-based countermeasures' impact assessment method. It also introduces a new formal definition of the two SysML profiles used in SysML-Sec and W-Sec, enabling (i) for the future automation of several W-Sec stages and (ii) for the definition of consistency rules ensuring the consistency of the models written in these two distinct modeling languages. In addition, the chapter evaluates W-Sec with a new industry 4.0 case-study and discusses the strengths and the current limitations of the approach in this new application field.

Keywords: Impact assessment · Formal methods · Mutations · SysML-Sec · TTool · Cyber-Physical Systems · Safety · Security · Performance

1 Introduction

Cyber-physical systems (CPS), including industrial control systems (ICS), control physical processes involved in safety-critical industries (energy, chemical industry, etc.) [19]. Therefore, cyber attacks targeting CPS may result in extremely harmful consequences. Maintaining a high security level for these systems is thus an essential process for CPS designers and operators. CPS security is an active research field including various challenging topics. Among them, the selection of optimal countermeasures[3] mitigating the systems' vulnerabilities is a complex issue. Indeed, any modification brought to a CPS must comply with the – often drastic – safety, security and performance requirements ensuring the system's dependability.

Two of our previous contributions proposed modeling and impact assessment approaches that help in addressing this issue. On the one hand, we proposed

[3] A *countermeasure* refers in our works to any modification brought to a system in order to mitigate one or several vulnerabilities. It can be a modification of the system's software, hardware, processes, and/or to its physical, logical and network architecture.

in [26] a countermeasures impact[4] assessment method targeting the context of CPS. Relying on CPS modeling with networks of timed automata (NTA), this method has three main drawbacks. Firstly, it is difficult to capture in a single modeling language heterogeneous aspects (such as data security, hardware architecture and discrete software behavior) at a same abstraction level. As a result, the models may lack in precision regarding some aspects (e.g., data security and low-level hardware aspects). Secondly, modeling all these aspects with a single NTA may lead to a model that is complex to understand and to verify. Last, due to the difficulty to model data security aspects with NTAs, the impact assessment of this NTA-based method does not include a fine-grained data security analysis. On the other hand, with SysML-Sec [4], we proposed a CPS design and verification method relying on two distinct SysML-based formal modeling languages. One of them targets high-level system architectural and behavioral aspects, when the other one is well suited for modeling fine-grained hardware and low-level security aspects. Yet, a drawback of SysML-Sec is that the method considers a single attacker model at the verification stage.

Therefore, we proposed in [25] a new countermeasures selection method called W-Sec, merging the strengths of our two previous contributions in order to correct the flaws we mention above. This chapter is an extended version of this work presented at the 10th ModelsWard conference. It brings to this initial paper several enhancements:

- A formal description of W-Sec is now given.
- For this purpose, the chapter provides a new mathematic definition of the models designed with the two modeling languages used in SysML-Sec and in W-Sec. This formalization is also the first step towards the automation of some W-Sec modeling stages.
- In order to ensure the modeling consistency between the two languages, the links between the both modeling views are now explicitely defined.
- W-Sec is applied to a new, more complex, real case-study, in a new context (connected factories and industrial control systems).

It is organized as follows. Sect. 2 presents the industrial case-study we use troughout the chapter to illustrate and evaluate the method. Then, Sect. 3 provides our new formalization of the two modeling languages. Sect. 4 describes both theoretical and practical aspects of W-Sec. Afterwards, Sect. 5 gives an overview of the related works. Last, Sect. 6 discusses the case-study results and the contributions W-Sec brings to our previous contributions.

2 IT'm Factory: an Industry 4.0 Case-Study

IT'm Factory is a research and training platform[5] hosted by the École des Mines de Saint-Étienne. It provides researchers and companies with a realistic connected factory case-study including typical industry 4.0 features (connected and

[4] *Impact* refers to *positive* impacts (i.e., efficiency) as well as to *negative* impacts (i.e., regressions).

[5] https://itm-factory.fr/index.php/objectif_et_visite_360/.

automated packaging chain, collaborative robots, virtual and augmented reality glasses, digital twin, etc.). The case-study we present in this paper is focused on the packaging chain.

2.1 The Packaging Chain

This chain is composed of four autonomous machines: a warehouse, a filling machine, a cobot (a collaborative autonomous articulated robotic arm) and a packer. The warehouse contains empty pots and, when operating, places them at regular intervals on the filling machine's conveyor belt. The filling machine detects the pots and fills them with grains. When a pot arrives at the extremity of the conveyor belt, the collaborative robot grabs it, closes the pot with a cap and places it on the packer conveyor belt. The packer then places it in a crate. When a crate is filled with six pots, the packer ejects it and continues the process with a new empty crate.

For supervision reasons, these machines are connected to a local network and may be supervised from a remote SCADA[6] console as well as from local control panels. Thanks to these HMIs, users can define the setpoint values for the physical processes the machines perform. Users can also supervise in real time the parameters of the machines and the number of filled pots, etc. Some values are also shared with a remote supervision system that can be connected to external networks. Fig. 2.1 shows the physical architecture and network topology of the packaging chain, with a focus on the internal architecture of the filling machine (the warehouse's and the packer's architectures are very close to this one). The data flows are also depicted in this figure with colored/decorated arrows.

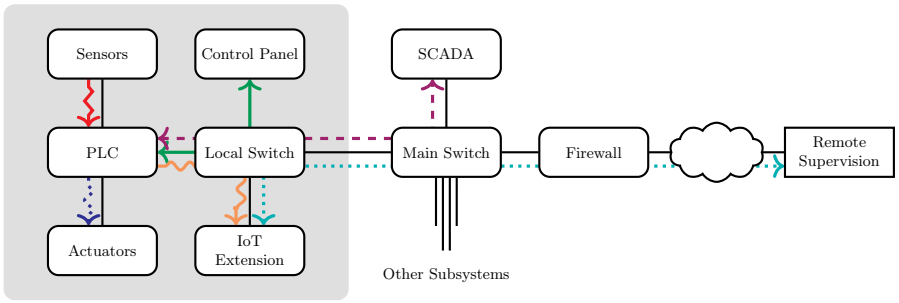


Fig. 1. Architecture of the filling machine (in the grey enclave) and its network flows

In the case-study presented in this paper, we specifically focus on the filling machine, the main switch and the SCADA remote console.

[6] Supervisory Control and Data Acquisition system.

2.2 Scenarios, Attack and Countermeasures

Our case-study integrates the two main functional scenarios of the filling machine: a scenario **Sc1** where the machine is supervised from its local control panel, and a scenario **Sc2** where it is supervised from the SCADA remote console. In the both scenarios the user sends a start command to the machine and then defines the following setpoints: the speed of the conveyor belt, the filling duration of each pot (i.e., the desired grains volume), and the centering offset of the filling machine's nozzle with respect to the center of each pot during the filling phases.

In this case-study, we also consider one attack we want to protect the system from. This attack aims at falsifying the centering offset setpoints in order to make the filling machine discharging the grains on the conveyor belt. To achieve this goal, the attack scenario consists of two consecutive steps. The first one is an ARP poisoning attack targeting the SCADA console in order to reroute its outgoing traffic to the attacker's host. The second step consists in modifying the centering offset setpoint with a value greater to the pots radius, and then in forwarding the modified message to the PLC (via the main and local switches). Note that the platform is actually already protected against this attack since the control algorithms executed by the PLC perform a consistency check on the received setpoint values (contermeasure **C1**, see below). Moreover, we assume for this case-study that the SCADA console and the main switch are vulnerable to ARP poisoning. However, for the needs of our paper we consider that the platform has not its embedded countermeasures yet.

In addition, we evaluate four countermeasures aiming at mitigating this attack. **C1**, an offset setpoint check: each time the PLC receives a centering offset setpoint, it checks if it exceeds the radius of a pot. If it is the case, the received value is replaced with the the value of the radius minus 1 cm. **C2** is a cryptographic countermeasure, consisting in symmetric encryption of the communications between the PLC and the SCADA console and the local control panel. **C3** consists in defining static ARP tables and denying ARP *is-at* requests. Last, **C4** is an emergency countermeasure consisting in unpluging the filling machine from the main switch.

The following sections will describe W-Sec both theoretically and practically, by relying on models and results coming from this case-study.

3 Preliminary Concepts: TTool's SysML Profiles

W-Sec relies on two modeling languages provided by the toolkit TTool [7]. These languages are two formally defined SysML profiles: one of them, AVATAR [23], used in TTool's *System View* (or S-View), is well suited for the high-level behavioral modeling of a system. The other one, DIPLODOCUS [13], used in TTool's *Hardware/Software Partitioning View* (or HSW-View), targets the joint modeling of hardware and software aspects of a system and is well suited for a low-level

[7] https://ttool.telecom-paris.fr/.

modeling of a component. This section provides new mathematical definitions of the models these two SysML profiles enable to design, that enable for a formal definition of W-Sec in Sect. 4.

3.1 Preliminary Definitions

The two modeling views share some common concepts. Although these concepts have not exactly the same meaning in the both cases, we present them in a common definition for simplicity reasons.

Definition 1 (Types, Attributes, Expressions and Signals).

- $\mathsf{Types} = \{\mathbb{B}ool, \mathbb{Z}, \mathbb{N}\}$
- Attr *is a set of* attributes*, typed by type* : $\mathsf{Attr} \rightarrow \mathsf{Types}$.
- Expressions *are usual integer and boolean expressions over attributes. They are typed in the usual way.*
- $\mathsf{Profiles} = \{(t_1, \cdots, t_n) \mid n \in \mathbb{N} \wedge \forall\, 1 \leq i \leq n,\ t_i \in \mathsf{Types}\}$.
- $\mathsf{Sign} = \mathsf{InSign} \sqcup \mathsf{OutSign}$ *is*[8] *a set of* signals*, typed by* $profile$: $\mathsf{Sign} \rightarrow \mathsf{Profiles}$. InSign *contains* input *signals.* $\mathsf{OutSign}$ *contains* output *signals.*

3.2 The System View (S)

In S-View, models rely on SysML blocks exchanging signals and having a behavior modeled with state-machine diagrams. Definition 5 provides the mathematical definition of a whole model designed in this view. Definition 4 describes a SysML block, and definition 3 defines its state-machine diagram. Last, the syntactic correctness of such a model is given in definition 6. These four definitions rely on basic concepts that are provided in definition 2.

Definition 2 (S-View Basic Sets and associated Abstract Syntax).

- Meth *is a set of* methods*, typed by* $profile$: $\mathsf{Meth} \rightarrow \mathsf{Profiles}$.
- $m(e_1, \ldots, e_n)$ *is a* method call*, where* m *is a method and* $e_1, \ldots, e_n$ *are expressions respecting the profile of* m.
- *There are four kinds of* actions*:*
  - Affectations $a := e$ *where* a *is an attribute and* e *an expression of the same type.*
  - Random affectations $a :=?$ *where* a *is an attribute.*
  - Send $send_s(e_1, \ldots, e_n)$*, where* s *is an output signal and* $e_1, \ldots, e_n$ *are attributes respecting the profile of* s.
  - Receive $receive_s(e_1, \ldots, e_n)$*, where* s *is an input signal and* $e_1, \ldots, e_n$ *are attributes respecting the profile of* s.
- Port *is a set of* ports.

Definition 3 (State Machine Diagram). *A* state machine diagram *is a directed (control flow) graph* $smd = (s_0, S, T)$ *where*

[8] "$\sqcup$" denotes the disjoint union.

- *S is a set of* states, $s_0 \in S$ *is the initial state.*
- *T is a set of* transitions $t = \langle s_{start}, after, condition, actions, s_{end} \rangle$ *where:*
  - $after = (t_{min}, t_{max})$*, in* $\mathbb{N}^2$*, constrains the delay before firing* t*.*
  - $s_{start}, s_{end} \in S^2$ *are respectively the source and target states of* t*.*
  - *condition is a boolean expression that must be true to enable* t*.*
  - *actions is a sequence of actions/method calls, executed when* t *is fired.*

attr(smd) (resp. meth(smd), sign(smd), insign(smd), outsign(smd)) denotes the set of attributes (resp. methods, signals, input signals, output signals) used in smd.

A state machine diagram is syntactically correct if all states are reachable from s_0 *(by some syntactic path on transitions).*

Definition 4 (Block Description).

A block description *is a 6-uple* $D = \langle A_D, M_D, P_D, S_{iD}, S_{oD}, smd_D \rangle$ *where* $A_D \subset \mathsf{Attr}$*,* $M_D \subset \mathsf{Meth}$*,* $S_{iD} \subset \mathsf{InSign}$*,* $S_{oD} \subset \mathsf{OutSign}$*,* $P_D \subset \mathsf{Port}$*, smd is a state machine diagram, and all these sets are finite.*

It is syntactically correct if smd_D *is syntactically correct,* $attr(smd_D) \subseteq A_D$ *and* $meth(smd_D) \subseteq M_D$*.*

The following definition derives from the *SysML block instance diagram* defined in [5].

Definition 5 (S-Model).
A system model*, or* S-Model*, is a 5-uple* $\langle \mathcal{B}, d, \mathcal{L}, \mathcal{C}, \mathcal{R} \rangle$ *where:*

- $\mathcal{B}$ *is a finite set of* blocks*.*
- d *is a function which associate a block description to each block of* $\mathcal{B}$*.*
  for $X \in \{A, M, P, S_i, S_o, smd\}$ *and* $B \in \mathcal{B}$*,* X_B *abbreviates* $X_{d(B)}$*, and* $\mathcal{P} = \bigsqcup_{B \in \mathcal{B}} P_B$*,* $\mathcal{S}_o = \bigsqcup_{B \in \mathcal{B}} S_{o_B}$ *and* $\mathcal{S}_i = \bigsqcup_{B \in \mathcal{B}} S_{i_B}$*.*
- $\mathcal{L} \subset \mathcal{P} \times \mathcal{P}$ *contains* links*. It is an irreflexive and antisymmetric partial injection.*
- $\mathcal{C} \subseteq \mathcal{L} \times \mathcal{S}_o \times \mathcal{S}_i$ *is a set of* connexions $\langle \langle p_1, p_2 \rangle, s_o, s_i \rangle$ *such that* p_1 *and* s_0 *belong to the same block, and* p_2 *and* s_i *belong to the same block.*
- $\mathcal{R} \subset \mathcal{B} \times \mathcal{B}$ *is such that*
  - *its transitive closure* $\mathcal{R}^*$ *is a noetherian order.*
  - *its inverse relation* $\mathcal{R}^{-1}$ *is a function.*

  When $\langle B_1, B_2 \rangle \in \mathcal{R}$*, we say that* B_1 *"contains"/"is a superblock of"* B_2 *and that* B_2 *"is contained by"/"is a subblock of"* B_1*.*

Remark: the set of subblocks of a block B defines a finite tree with B as root.

Definition 6 (Syntactically Correct S-Model).
Let $\mathcal{M} = \langle \mathcal{B}, d, \mathcal{L}, \mathcal{C}, \mathcal{R} \rangle$ *be a S-Model.* $\mathcal{M}$ *is syntactically correct if and only if:*

- $\forall \langle \langle p_1, p_2 \rangle, s_o, s_i \rangle \in \mathcal{C}$*,* $profile(s_o) = profile(s_i)$
- $\forall\, B \in \mathcal{B}$*,* $d(B)$ *is syntactically correct and* $\forall\, s_x \in sign(smd_B)$*,*
  - $\exists! \langle \langle p_1, p_2 \rangle, s_o, s_i \rangle \in \mathcal{C}$*,* $s_x = s_o \vee s_x = s_i$*.*
  - $\exists\, B' \in \mathcal{B}$*,* $(B', B) \in \mathcal{R}^* \wedge (s_x \in S_{oB'} \vee s_x \in S_{iB'})$*.*

3.3 The Hardware-Software Partitioning View (HSW)

HSW-View models are mathematically close to S-View models, but present some key differences. Indeed, they rely on a couple of specific SysML block diagrams (one modeling software aspects, the other modeling hardware aspects) and on two *allocations* that explicit the links between the both block diagrams. The whole HSW-Model is defined in definition 12. Definitions 10 and 11 define the both specific SysML block diagrams used in this view. The SysML blocks used in the software diagram are defined in definition 9, and the activity diagrams used for modeling their behavior in definition 8. These five definitions rely on basic concepts that are provided in definition 7.

Definition 7 (HSW-View Basic Sets and associated Abstract Syntax[9]).

- $\mathsf{DSign} \subseteq \mathsf{Sign}$ *is a set of* data *signals. For any* d *in* DSign, $profile(d) = (\mathbb{N})$.
- Keys *is a set of cryptographic* keys.
- *There are six kinds of* actions*:*
  - Affectations *and* Random affectations *such as defined in Def. 2.*
  - Send *actions:*
    - $send_e(a_1, \ldots, a_n)$*, where* $e \in \mathsf{OutSign}$ *and* $a_1, \ldots, a_n$ *are attributes respecting* $profile(e)$.
    - $encrsend_d(k, e)$ *where* $d \in \mathsf{OutSign} \cap \mathsf{DSign}$, $k \in \mathsf{Keys}$ *and* e *is a (natural) integer expression.*
  - Receive *actions:*
    - $receive_e(a_1, \ldots, a_n)$*, where* $e \in \mathsf{InSign}$ *and* $a_1, \ldots, a_n$ *are attributes respecting* $profile(e)$.
    - $encrreceive_d(k, e)$ *where* $d \in \mathsf{InSign} \cap \mathsf{DSign}$, $k \in \mathsf{Keys}$ *and* e *is a (natural) integer expression.*
  - Delay*:* $delay(e)$*, where* e *is a (natural) integer expression.*

Definition 8 (Activity Diagram). *An* activity diagram *is a directed (control flow) graph* $ad = \langle s_0, S, T \rangle$ *where:*

- S *is a set of states,* $s_0 \in S$ *is the initial state.*
- T *is a set of transitions* $t = \langle s_{start}, condition, action, s_{end} \rangle$ *where:*
  - $(s_{start}, s_{end}) \in S^2$ *are respectively the source and the target states of* t
  - $condition$ *is a boolean expression that must be true to enable* t
  - $action$ *is an action executed when* t *is fired.*

$attr(ad)$ *(resp* $sign(ad)$, $keys(ad)$*) denotes the set of attributes (resp. signals, keys) used in* ad.

An activity diagram is syntactically correct if all states are reachable from s_0 *(by some syntactic path on transitions).*

[9] Note that in this view, data are abstracted: we do not model data values and the profile of a data signal is an integer representing the amount of transfered data. Moreover, computations (exec, wait,...) are abstracted by their complexity (a duration) in one unique "delay" operation.

Definition 9 (Task Description).
A task description *is a 5-uple* $D = \langle A_D, S_{iD}, S_{oD}, ad_D \rangle$ *where* $A_D \subset \mathsf{Attr}$, $S_{iD} \subset \mathsf{InSign}$, $S_{oD} \subset \mathsf{OutSign}$, ad_D *is an activity diagram, and all these sets are finite.*

It is syntactically correct if ad_D *is syntactically correct,* $attr(ad_D) \subseteq A_D$ *and* $sign(ad_D) \subseteq S_{iD} \cup S_{oD}$.

Definition 10 (Application Model).
An application model *is a 4-uple* $\langle \mathcal{T}, d, \mathcal{C}, \mathcal{K} \rangle$ *where* $\mathcal{K} \subseteq \mathsf{Keys}$ *and:*

- $\mathcal{T}$ *is a finite set of* tasks.
- d *is a function which associate a task description to each task of* $\mathcal{T}$. *For* $X \in \{A, S_i, S_o, ad\}$ *and* $T \in \mathcal{T}$, X_T *abbreviates* $X_{d(T)}$, *and* $\mathcal{S}_o = \bigsqcup_{T \in \mathcal{T}} S_{oT}$ *and* $\mathcal{S}_i = \bigsqcup_{T \in \mathcal{T}} S_{iT}$.
- $\mathcal{C} \subset \mathcal{S}_o \times \mathcal{S}_i$ *is a set of* connexions.

It is syntactically correct if and only if :

- $\forall \langle s_o, s_i \rangle \in \mathcal{C}$, $profile(s_o) = profile(s_i)$
- $\forall\, T \in \mathcal{T}$, $d(T)$ *is syntactically correct and* $keys(ad_T) \subseteq \mathcal{K}$.

Architecture models rely on specific SysML blocks called *hardware nodes*. There are different kinds of hardware nodes, depending on the compoment they model: *Bus*, *CPU*, *FPGA*, *HWA* (hardware accelerator), *DMA Controller*, *Memory*, and *Bridge*.

Definition 11 (Architecture Model).
An architecture model *is a 2-uple* $\langle \mathcal{H}, \mathcal{L} \rangle$ *where:*

- $\mathcal{H} = Buses \sqcup CPU \sqcup FPGA \sqcup HWA \sqcup DMA \sqcup Memories \sqcup Bridges$ *is a finite set of* hardware nodes.
- $\mathcal{L} \subset (\mathcal{H} \backslash Buses) \times Buses$ *is a set of* links *between hardware nodes.*

Definition 12 (HSW-Model).
A HSW-Model *is a 4-uple* $\langle App, Arch, Alloc_t, Alloc_k \rangle$ *where* $App = \langle \mathcal{T}, d, \mathcal{C}, \mathcal{K} \rangle$ *is an application model,* $Arch = \langle \mathcal{H}, \mathcal{L} \rangle$ *is an architecture model such as defined in definition 11, and:*

- $Alloc_t : \mathcal{T} \to CPU \sqcup FPGA \sqcup HWA$ *is a total function called* task allocation.
- $Alloc_k \subseteq \mathcal{K} \times Memories$ *is the* key allocation.

It is syntactically correct iff App is syntactically correct and, informally,

- *for each signal connection there is a path from the execution node of the task of the output signal to a memory and a path from this memory to the execution node of the task of the input signal.*
- *for each key used by a task, there is a path from the execution node of the task to a memory where the key is allocated.*

(roughly speaking, a path is a sequence of linked busses and bridges between an execution node and a memory)

4 W-Sec: Theory and Practice

W-Sec relies on formal methods, since its aim is to **quantitatively** assess the impact and efficiency of security countermeasures. Verification techniques, including model-checking and simulation, are then needed for verifying safety and security properties as well as performance thresholds. Therefore formal methods take part in the four W-Sec stages: an initial modeling stage, a second modeling (mutation) stage, a verification stage and then a last modeling (enrichment) stage. This section explains these four stages both from theoretical a point of view and an applied perspective. These stages derive from the stages of our initial NTA-based impact assessment method, and bring to them several substantial improvements as discussed in Sect. 6.

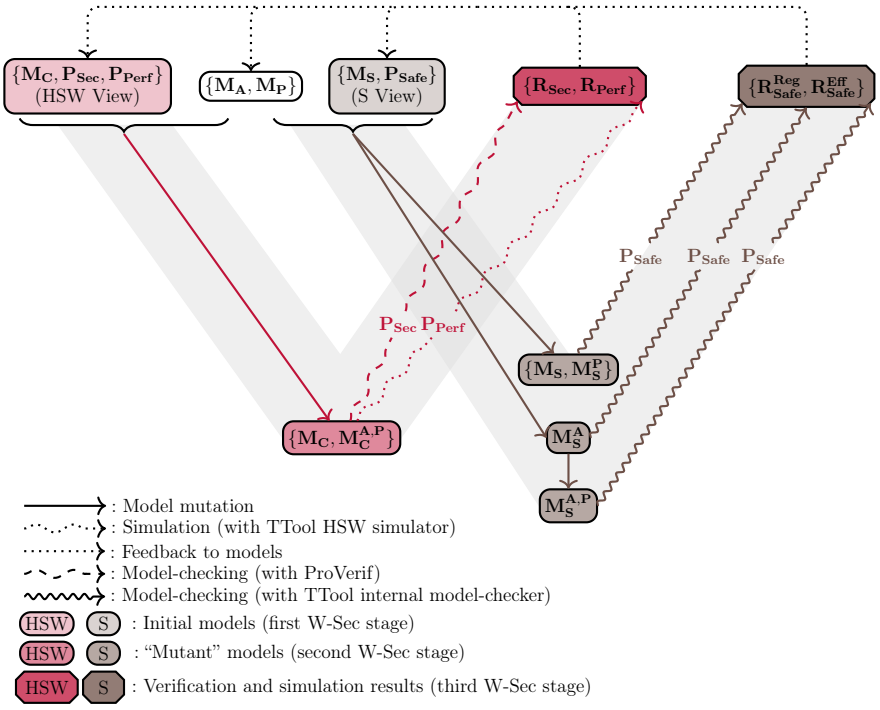


Fig. 2. The W-Sec Method (adapted from [25])

4.1 Modeling the System using S and HSW Views

The preliminary W-Sec stage consists in building a comprehensive modeling of the system. For this, three classes of models are designed:

- The **system models**, called S-Models (see Sect. 3), are captured in S-View. We denote with $\mathbf{M_S}$ the set of these system models: each element of $\mathbf{M_S}$ models the high-level discrete and physical behavior of the whole system[10]. Note that $\mathbf{M_S}$ gathers several models. Indeed, a system may operate according to different *functional scenarios*: for instance, the IT'm Factory's packaging chain can be controlled from the local control panel (first scenario), from the remote SCADA (second scenario), or from the both (third scenario). As we usually want to compute the countermeasures impact on the system according to several of these functional scenarios, we need to design one system model per scenario. Differences between these system models are usually light, e.g., the variables of some blocks are initialized with different values.
- The **components models**, called HSW-Models (see Sect. 3), are captured in HSW-View. We denote with $\mathbf{M_C}$ the set of these components models. $\mathbf{M_C}$ contains one model per modeled component[11]. As explained in [25], these models are focused on the component but can also include some external tasks/hardware patterns in order to model the component's external communications and to be able to consider these external elements during the verification stage.
- The **attack models** and the **countermeasures models**. We denote with $\mathbf{M_A}$ and $\mathbf{M_P}$ the sets of these models. $\mathbf{M_A}$ is a set of attack models (e.g., attack trees): each element of this set describes an attack that can occur on the system. $\mathbf{M_P}$ is a set of countermeasures models: each element of this set describes a countermeasure we want to assess.

Unlike our NTA-based impact assessment method [26] that relies on a single model approach at this stage, W-Sec is based on a two-view modeling approach for two main reasons.

Firstly, as we explained in [25], the single model approach 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, an important effort is necessary to express heterogeneous concepts with the same language and within the same views. Also, the resulting model typically lacks in precision with respect to one of the modeled aspects, for instance data security, and details are not given at the same abstraction level. The two-view approach tackles both issues since (i) the HSW-View provides dedicated SysML patterns for low-level hardware and data security modeling and (ii) the S-View provides a SysML profile tailored for complex systems behavioral modeling, including all the modeling capabilities of standard behavioral modeling formalisms like networks of finite automata. This approach thus enables for a better modeling accuracy with respect to both high-level (system) and low-level (components) aspects.

[10] Systems models may also include some blocks and signals modeling the system's environment.

[11] A *component* is an equipment of the system. For instance, the components of IT'm Factory's packaging chain include the PLC, the two switches, the local control panel, etc.

Secondly, gathering all aspects in a single model leads to a pointlessly complex model that may be difficult to formally verify or to simulate. Indeed, the objective consists in performing safety, security and performance analyses thanks to model-checking and simulation. To this end, some modeling aspects are essential when performing one of these three assessments while being unnecessary when performing the two others (e.g., the output values of an algorithm can be needed when analyzing its safety, while they can be useless when assessing its performance). Once again, the two-view approach addresses this issue since our modeling approach cleverly separates safety, security and performance concerns. On the one hand, **security** and **performance** assessments are indeed performed in the HSW-View: hardware modeling is obviously needed to assess a system's performance, and verifying 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) includes hardware-related attacks (e.g, putting a probe on a bus, stealing a cryptographic key from a memory, ...). On the other hand, **safety** assessments are carried out in the S-View, since their aim is to provide an analysis of the countermeasures regarding the overall system's functions and behavior. If safety countermeasures need to modify the HSW views, e.g., by introducing a redundancy on processor, then obviously the HSW view must be updated and performance and security reassessed. This drawback due to the separation of concerns is further discussed in Sect. 4.3.

In conclusion, the models of $\mathbf{M_C}$ shall only include the information that is necessary to precisely depict the software and hardware architecture of the modeled components, as well as the low-level security and performance, including algorithmic complexity, aspects. In addition, the models of $\mathbf{M_S}$ shall only integrate the information that is necessary to depict the high-level aspects of the system (e.g., system's functions, dynamics and system-wide network topology) and, if needed, some component-level information in order to depict the high-level behavior of the components algorithms (i.e., the evolution of their output parameters depending on their inputs).

Based on the system's safety, security and performance requirements, safety and security properties ($\mathbf{P_{Safe}}$ and $\mathbf{P_{Sec}}$) are also designed at this preliminary stage, and performance thresholds ($\mathbf{P_{Perf}}$) are also set. Due to our separation, security properties are included in the HSW-View while the S-View features safety properties. TTool offers specific pragmas to describe properties within views.

Example 1 (S-Model and HSW-Model).

An excerpt of the packaging chain S-Model is given in Fig. 3 and 4:

- Fig. 3 shows the part of the block diagram modeling the filling machine. Note that only a subset of the blocks attributes and signals is displayed in this figure.
- Fig. 4 shows the state-machine diagram associated with the block `FillingMachine_HMI`.

In addition, Fig. 5 provides an excerpt of the architecture diagram modeling the hardware part of the Siemens S7-1200 PLC used for the filling machine

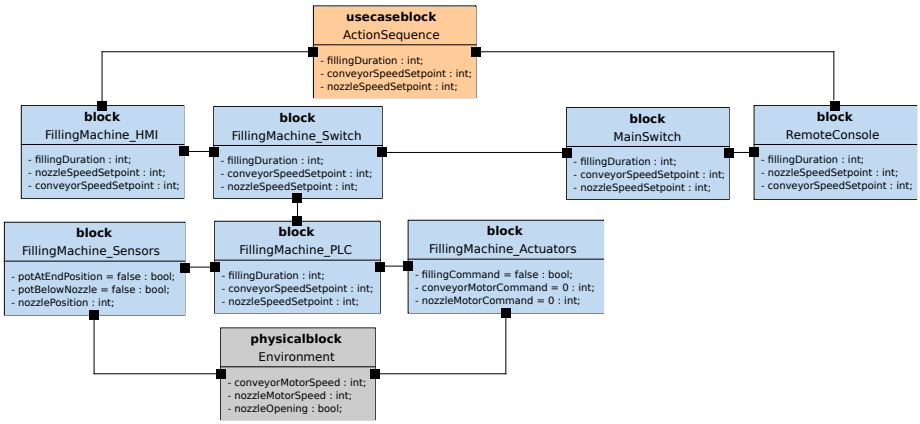


Fig. 3. S-Model: part of the block diagram modeling the filling machine

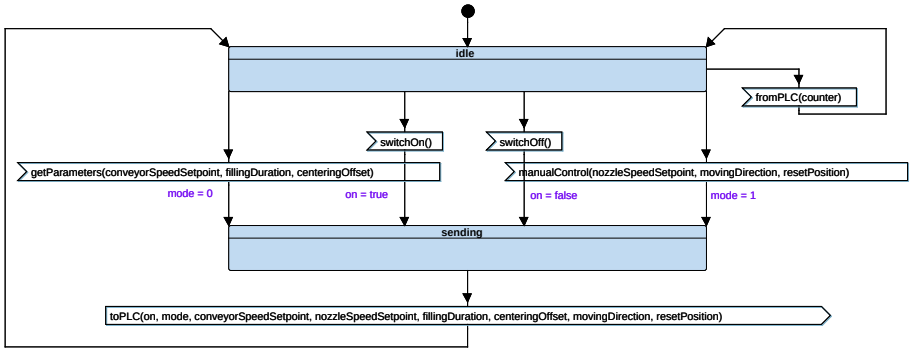


Fig. 4. S-Model: state-machine diagram modeling the behavior of the filling machine's control panel

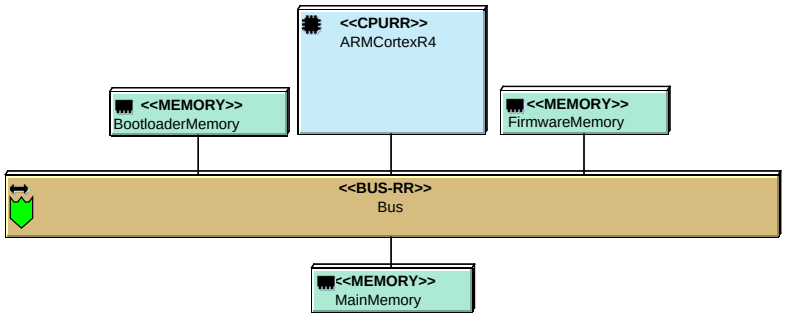


Fig. 5. HSW-Model: part of the architecture diagram of the filling machine PLC (based on the analysis presented in [1])

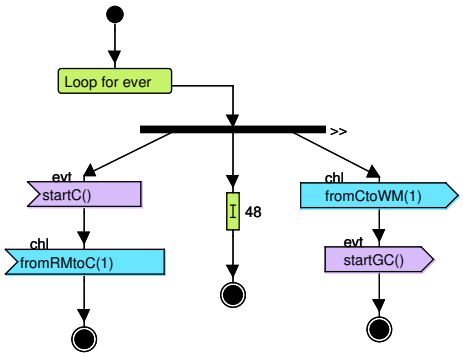


Fig. 6. HSW-Model: activity diagram of a typical task

control, and Fig. 6 shows an activity diagram of a task executed by the PLC. The horizontal bar models a sequence of activities (from left to right). Purple sending/receiving refer to events, while blue sending/receiving refer to data exchanged. There, only a quantity of data is exchanged, not values: values are not needed for performance evaluation, but only the amount of values which is exchanged via buses and memories. This diagram illustrates the HSW-View modeling abstractions: apart from the signals reception and sending, the whole algorithm behavior is depicted through a complexity operator that performs 48 operations on integers. This figure has been chosen after an analysis of the source-code of the modeled task, by adding the number of add, multiply, move, etc. instructions along the most complex execution path.

At the end of this stage, the S-View then features the set of system models $\mathbf{M_S}$ and the set of safety properties $\mathbf{P_{Safe}}$, while the HSW-View features the set of components models $\mathbf{M_C}$, the set of security properties $\mathbf{P_{Sec}}$ and the set of performance thresholds $\mathbf{P_{Perf}}$ (see Fig. 2).

In order to ease the consistency between S and HSW views, the following modeling rules should be followed when it is possible:

- In the S-View, a component shall be modeled with one single block (that might, if needed, contain subblocks).
- In the HSW-View, for each HSW-Model, the attribute sets of the tasks shall be included in the attribute set of the S-Model block modeling the same component.
- In the HSW-View, for each HSW-Model, each output (resp. input) data signal modeling an external communication of the component shall match a unique output (resp. input) signal belonging to the output (resp. input) signals set of the S-Model block modeling the same component. More formally, given a component C, we denote with $\mathcal{M}_C \in \mathbf{M_C}$ its HSW-Model and with B_C the block modeling this component in models of $\mathbf{M_S}$. S_{oB_C} (resp. S_{iB_C}) is the output (resp. input) signals set of B_C. We denote with $S_{o\mathcal{M}_C}$ (resp. $S_{i\mathcal{M}_C}$) the output (resp. input) signals set of the application model of $\mathcal{M}_C$,

and with $S^{d,ext}_{o\mathcal{M}_C} \subseteq S_{o\mathcal{M}_C}$ (resp. $S^{d,ext}_{i\mathcal{M}_C} \subseteq S_{i\mathcal{M}_C}$) the set of output (resp. input) *data* signals that model an external communication of the component. Then, it shall be ensured that $\forall\, \mathcal{M}_C \in \mathbf{M_C}, \exists f :\ S^{d,ext}_{o\mathcal{M}_C} \hookrightarrow S_{oB_C} \wedge \exists\, g :\ S^{d,ext}_{i\mathcal{M}_C} \hookrightarrow S_{iB_C}$[12].

4.2 Modeling the Countermeasures Deployment and the Attacks: HSW-Models and S-Models Mutations

Like in [26], the second W-Sec stage consists in altering the initial sets of models $\mathbf{M_S}, \mathbf{M_C}$ in order to enrich them with the attacks and countermeasures description provided by $\mathbf{M_A}$ and $\mathbf{M_P}$. Alterations of formal models are often called *mutations* [21,2,24]. In our context, a mutation of a S-Model or of a HSW-Model is any alteration that preserves its syntactic correctness (see Def. 13 - 15).

S-Models Mutations

Definition 13 (S-Model Mutation). *Let $\mathfrak{S}$ be the set of all syntactically correct S-Models. We call* S-Model mutation *any (partial) function $\mu : \mathfrak{S} \nrightarrow \mathfrak{S}$.*

Such a function can be seen as a composition of atomic modifications on the model, or *mutation operators* similarly to what is explained in [2] for timed automata. Concerning S-Models, these operators include:

- At S-Model level:
  1. Addition/deletion of a block
  2. Addition/deletion of a link between two ports
  3. Addition/deletion of a connexion
  4. Addition/deletion of a block containment relation
- At block level:
  5. Addition/deletion of an attribute
  6. Addition/deletion of an input (resp. output) signal
  7. Addition/deletion of a state in the block's state machine diagram
  8. Addition/deletion of a transition in the block's state machine diagram

Formally defining each of these operators in this chapter would be pointlessly long; but we provide below, as an example, the definition of the block addition operator that is used in example 2.

Definition 14 (Block Addition). *Let $\mathfrak{B}$ be the set of all blocks and $\mathfrak{S}$ be the set of all syntactically correct S-Models. We define a* block addition *as the function*

$$\begin{aligned} add : \mathfrak{S} \times \mathfrak{B} &\to \mathfrak{S} \\ (\mathcal{M}, B) &\mapsto \mathcal{M}' \end{aligned}$$

such that $\mathcal{M} = \langle \mathcal{B}, d, \mathcal{L}, \mathcal{C}, \mathcal{R} \rangle$ and $\mathcal{M}' = \langle \mathcal{B} \cup \{B\}, d, \mathcal{L}, \mathcal{C}, \mathcal{R} \rangle$.

[12] $f : E \hookrightarrow F$ means that f is an injective application from E to F.

Using these S-Model mutations, three sets of S-Models are derived from the set $\mathbf{M_S}$ and from the sets of counteremeasures and attacks models $\mathbf{M_P}$ and $\mathbf{M_A}$.

1. $\mathbf{M_S^P}$. This set contains the initial S-Models enriched with the countermeasures descriptions. Since it is necessary to assess the impact of the countermeasures with respect to each functional scenario, each S-Model of $\mathbf{M_S}$ is mutated into $card(\mathbf{M_P})$ mutant S-Models (one per countermeasure). In other terms, if we denote with μ_{Patch} the mutation that integrates the countermeasure model *Patch* with a S-Model, $\mathbf{M_S^P} = \{\mu_{Patch}(\mathcal{M}) \mid \mathcal{M} \in \mathbf{M_S}, Patch \in \mathbf{M_P}\}$.
2. $\mathbf{M_S^A}$. This set contains the relevant initial S-Models enriched with the attacks descriptions: for each element of $\mathbf{M_A}$, each S-Models of $\mathbf{M_S}$ that model functional scenarios in which the modeled attack can occur is mutated into a S-Model enriched with the attack model. In other terms, if we denote with (i) μ_{Att} the mutation that integrates the attack model *Att* with a S-Model and with (ii) $\mathbf{M_S}_{Att} \subseteq \mathbf{M_S}$ the set of S-Models that model functional scenarios in which the attack described by *Att* can occur, $\mathbf{M_S^A} = \bigcup_{Att \in \mathbf{M_A}} \{\mu_{Att}(\mathcal{M}) \mid \mathcal{M} \in \mathbf{M_S}_{Att}\}$.
3. $\mathbf{M_S^{A,P}}$. This set contains the S-Models of $\mathbf{M_S^A}$ enriched with the countermeasures models: for each element of $\mathbf{M_S^A}$, $card(\mathbf{M_P})$ mutant S-Models (one per countermeasure) are produced. In other terms, if we denote with μ_{Patch} the mutation that integrates the countermeasure model *Patch* with a S-Model, $\mathbf{M_S^{A,P}} = \{\mu_{Patch}(\mathcal{M}) \mid \mathcal{M} \in \mathbf{M_S^A}, Patch \in \mathbf{M_P}\}$.

Example 2 (Mutations of S-Models).

Fig. 7 shows the block diagram initially depicted in Fig. 3 after a mutation modeling the integration of the attack scenario introduced in Sect. 2. This mutation is the composition of: (i) a block addition, (ii) two link additions, (iii) two new connexions, (iv) two state additions and (v) seven transition additions in the `RemoteConsole` block. In addition, Fig. 8 shows an excerpt of the `FillingMachine_PLC` block's state-machine diagram before and after a mutation modeling the offset setpoint check countermeasure. This mutation consists in (i) a transition deletion, (ii) two state additions and (iii) four transition additions.

HSW-Models Mutations

Definition 15 (HSW-Model Mutation). *Let* $\mathfrak{C}$ *be the set of all syntactically correct HSW-Models. We call* HSW-Model mutation *any (partial) function* $\mu : \mathfrak{C} \nrightarrow \mathfrak{C}$.

As for S-Models mutations, HSW-Models mutations are sequences of atomic operators:

- At HSW-Model level:
  1. Modification of the *task allocation* function
  2. Addition/deletion of an element in the *key allocation* set

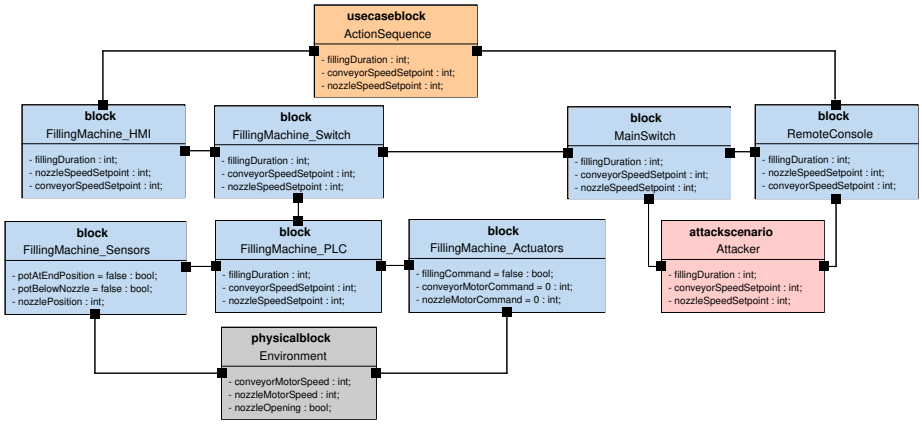
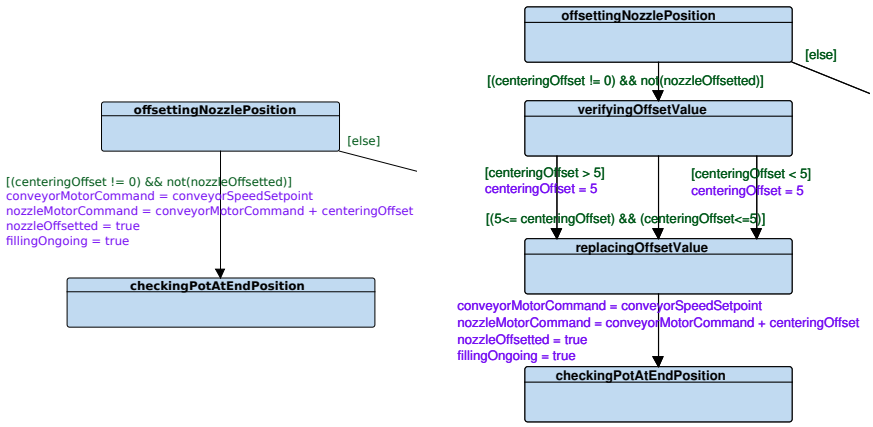


Fig. 7. M_S^A: mutation modeling an ARP spoofing and man in the middle attack



(a) smd before mutation (b) smd after mutation

Fig. 8. M_S^P: mutation of a subpart of the PLC state-machine diagram, modeling the offset setpoint check

- At architecture model level:
  3. Addition/deletion of a hardware node
  4. Addition/deletion of a link
- At application model level:
  5. Addition/deletion of a task
  6. Addition/deletion of a connexion
- At task level:
  7. Addition/deletion of an attribute
  8. Addition/deletion of an input (resp. output) signal
  9. Addition/deletion of a state in the tasks's activity diagram
  10. Addition/deletion of a transition in the tasks's activity diagram

Using these HSW-Model mutations, the set of HSW-Models $\mathbf{M_C^{A,P}}$ is derived from the set $\mathbf{M_C}$ and from the sets of counteremeasures and attacks models $\mathbf{M_P}$ and $\mathbf{M_A}$. This set contains the relevant HSW-Models[13] enriched with the countermeasures and attack descriptions. Due to the modeling approach in the HSW-View, countermeasures are often modeled with additional tasks and/or with additional complexity and encryption/decryption operators[14]. In HSW-View, attacks are modeled in two ways:

1. a Dolev-Yao attacker model is embedded in ProVerif [7], the security model-checker used in TTool. Therefore, before performing security verification, the tool automatically composes HSW-Models with a worst-case attack scenario at data level.
2. if this attacker model is inadequate to model the desired attack scenario (e.g., if the attack modifies the execution flow of a task), mutations can be used to model it thanks to new tasks, signals and actions in activity diagrams that influence the execution flow of the application model.

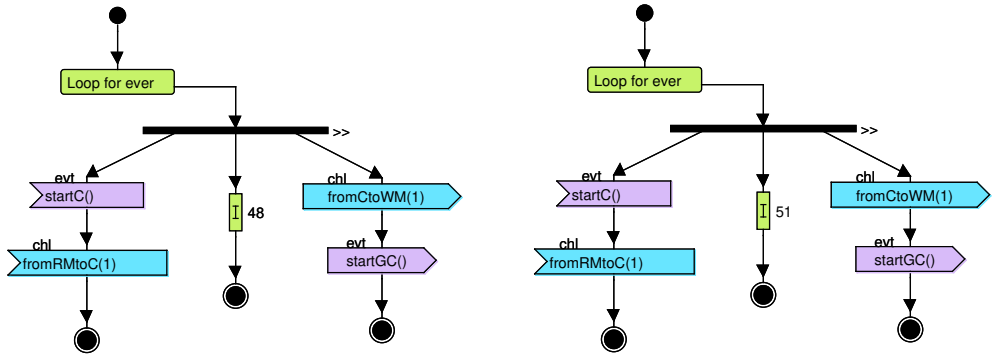
In other terms, if we denote with (i) μ_{Att} the mutation that integrates the attack model *Att* with a HSW-Model, with (ii) μ_{Patch} the mutation that integrates the countermeasure model *Patch* with a HSW-Model and with (iii) $\mathbf{M_C}_{Att,Patch} \subseteq \mathbf{M_C}$ the set of HSW-Models that model components on which the countermeasure modeled by *Patch* is deployed and which is targeted by the attack scenario modeled by *Att*, $\mathbf{M_C^{A,P}} = \bigcup_{Att \in \mathbf{M_A}, Patch \in \mathbf{M_P}} \{\mu_{Att} \circ \mu_{Patch}(\mathcal{M}) \mid \mathcal{M} \in \mathbf{M_C}_{Att,Patch}\}$.

Example 3 (Mutations of HSW-Models).

Fig. 9 shows the mutation, in HSW-View, modeling the deployment of the offset setpoint check countermeasure. Since this countermeasure consists in performing two comparisons and at most one move instruction, the complexity operator modeling 48 integer operations is replaced with a complexity operator modeling three more operations on integers.

[13] i.e., the models of components on which the countermeasures described by $\mathbf{M_P}$ are deployed and that are targeted by the attack scenarios described by $\mathbf{M_A}$.

[14] These operators are actions over transitions in the tasks activity diagrams (see Def. 7).



(a) ad before mutation (b) ad after mutation

Fig. 9. $\mathbf{M_C^{A,P}}$: mutation of an activity diagram of a task executed by the PLC, modeling the offset setpoint check

4.3 Computing the Impacts and Enriching the Models

Once the mutations have been applied, the third stage of W-Sec consists in computing the positive and negative impacts of the countermeasures thanks to formal verification and simulation.

Computing the Components-Level Performance and Security Impacts

First, simulations are performed with TTool's internal simulator on the models of $\mathbf{M_C}$ and on the models of $\mathbf{M_C^{A,P}}$. The aim of these simulations is to measure, for each component targeted by a given countermeasure, the total execution time of an iteration of the application modeled in the component's HSW-Model. The set $\mathbf{R_{Perf}}$ contains the results of these simulations.

Example 4 (Performance impacts).

(i) Composition of the set $\mathbf{M_C}$. We have modeled two components in the HSW-View: the filling machine's S7-1200 PLC, and the SCADA (i.e., the remote console). The whole control algorithm has been modeled in the PLC application model, whereas only an abstract setpoint message formatting and sending task has been modeled in the SCADA application model.

(ii) Assessed countermeasure. We provide here the performance assessment results of the symmetric encryption countermeasure. Note that the full results (i.e. regarding the four countermeasures) will be provided in Sect. 6.

(iii) Chosen performance thresholds. In our case-study, we consider that the PLC performance threshold is given by the filling machine's sensors commutation rate. As their maximum commutation rate is equal to 500 Hz, the full operating cycle of the PLC (i.e., the elapsed time between the reception of a

setpoint message and the sending of the commands/supervision message) shall remain **below 2 ms**. In addition, since the SCADA console is not safety-critical we have not set up a performance threshold for this component; notwithstanding, we still can compare the execution times after/before the deployment of the countermeasures and therefore quantify their overhead cost.

(iv) Results ($\mathbf{R_{Perf}}$): Table 1 provides the execution times [15], in nanosec-

	No countermeasure	*Symmetric Encryption*
PLC	608.5 ns	690 ns (+13,3%)
SCADA	42.5 ns	170.8 ns (+301,9%)

Table 1. Some performance assessment results

onds, of an iteration of the PLC control application and of the SCADA message formatting and sending task. We can notice that the overall computational cost for the PLC is far below our performance threshold in both cases, and that the overhead cost for the SCADA message handling task (128.3 ns) is reasonable. Therefore, this countermeasure should be suitable from a performance perspective.

Secondly, security verifications are carried out on the models of $\mathbf{M_C}$ and $\mathbf{M_C^{A,P}}$. These verifications consists in (i) an automated HSW-Models to ProVerif specifications translation [18,17] and (ii) a model-checking of these specifications against the chosen security properties with ProVerif. The set of security results $\mathbf{R_{Sec}}$ is thus built at this stage.

Example 5 (Security verification).

(i) Composition of the set $\mathbf{M_C}$, **(ii) Assessed countermeasure**: same as above.

(iii) Chosen security properties. Since we want to protect the system against a falsification of the network messages sent to the PLC, we have chosen to verify the data origin authenticity (*strong authenticity*) and data integrity (*weak authenticity*) for the following data channels: SCADA → Main switch and Filling machine switch → PLC.

(iv) Results ($\mathbf{R_{Sec}}$): Table 2 provides the formal verification results con-

	No countermeasure	*Symmetric Encryption*
SCADA → Main switch	Weak auth.: ✗	Weak auth.: ✓
	Strong auth.: ✗	Strong auth.: ✗
Local switch → PLC	Weak auth.: ✗	Weak auth.: ✓
	Strong auth.: ✗	Strong auth.: ✗

Table 2. Some security assessment results

[15] Each figure is an average of 10 consecutive measurements.

cerning the two chosen properties. We notice that the nominal (i.e., with no countermeasure deployed) system does not protect the two studied data flows from an attacker. On the other hand, symmetric encryption in our implementation seems to enable the components for detecting any alteration of the data conveyed by the both channels[16].

Computing the System-Level Safety Impacts

The last impact assessment performed at this stage consists in verifying the models of $\mathbf{M_S}$, $\mathbf{M_S^P}$, $\mathbf{M_S^A}$ and $\mathbf{M_S^{A,P}}$ against the chosen safety properties. The models are verified thanks to the direct model-checking algorithm of S-Models embedded in TTool [10]. Like in our NTA-based impact assessment method [26], the aim of these model-checking operations is to quantify:

- the safety regression due to each countermeasure deployment, thanks to the comparison of the model verifications performed on the models belonging to $\mathbf{M_S}$ and $\mathbf{M_S^P}$.
- the efficiency of each countermeasure in terms of safety requirements recovery when the attack is carried out, thanks to the comparison of the model verifications performed on the models belonging to $\mathbf{M_S^A}$ and $\mathbf{M_S^{A,P}}$.

Example 6 (Safety assessment).

(i) Functional scenarios. The two functional scenarios **Sc1** and **Sc2** (see Sect. 2) are assessed.

(ii) Assessed countermeasure. We present here the safety assessment of the offset setpoint check.

(iii) Safety properties. We have chosen to evaluate the following safety properties. **P1**: every pot put on the conveyor belt will eventually be filled by the machine with the volume chosen by the operator; **P2**: the machine never discharges grains outside of a pot (on the conveyor belt); and **P3**: the SCADA remote console is always kept up-to-date with the number of filled pots.

(iv) Results ($\mathbf{P_{Perf}}$): Table 3 (a) shows that the assessed countermeasure

	No countermeasure	*Offset check*
P1	✓	✓
P2	✓	✓
P3	✓	✓

(a) No attack, scenarios **Sc1** and **Sc2**

	No countermeasure	*Offset check*
P1	✗	✓
P2	✗	✓
P3	✓	✓

(b) With attack, scenario **Sc2**

Table 3. Some safety assessment results

does not cause any safety regression, neither in scenario **Sc1** nor in scenario **Sc2**:

[16] Actually, symmetric encryption does not provide integrity. But our verifying environment assumed that if a ciphered message is modified by an attacker, the receiver will notice that the deciphered text is inconsistent.

with respect to our safety requirements, this countermeasure has no negative impact. In addition, Table 3 (b) shows that when the attack is carried out, the countermeasure preserves the properties **P1** and **P2** that the nominal system was not able to preserve: this countermeasure therefore has a positive impact.

Enriching the Models

Once the assessment results are known, further W-Sec iterations may be necessary in order to refine the countermeasures models $\mathbf{M_P}$. For instance, for a given countermeasure, performance results on a component might reveal that some functions cannot be executed by the component: the countermeasure model shall then be refined in order to express this issue in the S-Models thanks to more suitable mutations. Once the models $\mathbf{M_P}$ have been refined, the second and third W-Sec phases are performed again in order to obtain more accurate assessment results.

In addition, if one of the assessed countermeasures *Patch* is selected to be deployed on the system, the $\{\mathbf{M_C}, \mathbf{M_S}\}$ model base must be modified in order to keep them up-to-date: $\mathbf{M_S}$ is replaced with the subset of $\mathbf{M_S^P}$ that depicts the deployment of *Patch* on the system; and the elements of $\mathbf{M_C}$ that model a component targeted by *Patch* are replaced with the result of the mutation integrating *Patch* with themselves.

5 Related Works

Assessing the (negative and positive) impacts of several security countermeasures in order to find the optimal one is not really a new research topic: Brykczynski and Small [8] and Nicol [22] highlighted in 2003 and 2005 the interest of such an assessment before the deployment of a countermeasure. Subsequently, several optimal countermeasures selection methods have been proposed. Nespoli et al. [20] provided an interesting litterature survey focused on the recent (between 2012 and 2016) contributions to this topic. When it comes to evaluate the countermeasures' negative impacts, the surveyed approaches mainly focus on the monetary cost of them. However, as we explained in [25], "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." Some of these approaches even explicitely assess the "collateral damages" of the countermeasures, like the one presented by Gonzalez-Granadillo et al. [14]. Nevertheless, none of them seem to enable for a precise enough countermeasures impact assessment with respect to the behavior of the system, which is critical regarding CPS. For instance, a comprehensive evaluation of a software countermeasure deployed on a PLC controlling a physical process requires, beyond the evaluation of the availability of the PLC, to quantify how the output command values of

the PLC's control algorithms are affected. To the best of our knowledge, however, few contributions published after this survey propose approaches enabling for a precise and objective impact assessment of security countermeasures with respect to the systems behaviors.

Part of a patch-management approach for naval systems, we proposed in [26] a formal verification-based impact assessment method for security countermeasures. This method relies on formal modeling and verification of networks of UPPAAL timed automata [6]. Firstly, the system is modeled with a network of timed automata (NTA). Then, when a vulnerability that affect the system is discovered, the NTA is *mutated* into four sets of new NTAs modeling (i) the vulnerable system, (ii) the vulnerable system enhanced with security countermeasures mitigating the vulnerability, (iii) the realization of successful attacks exploiting this vulnerability on the original vulnerable system, and (iv) the realization of these attacks on the "patched" systems. Afterwards, the original NTA as well as the modified NTAs are formally verified against a set of safety properties. The positive and negative impacts of the evaluated countermeasures are then assessed by comparing the verification results. Formal modeling and verification for the assessment of cybersecurity-related events (countermeasures deployment, attacks) is well-suited to the context of CPS – including ICS –, and further promising contributions have been proposed by Jawad et Jaskolka in [15] regarding the impact assessment of cyberattacks, and by Jawad et al. in [16] where the authors model a botnet infrastructure with UPPAAL NTAs and evaluate the efficiency of a countermeasure thanks to simulation. Nevertheless, as we explained in [24], a drawback of our previous approach [26] is that the NTA-based modeling formalism lacks in expressiveness with respect to data security aspects: for instance, data confidentiality is here modeled in a too simplistic way, with a boolean variable modeling an illegitimate access to the component processing the data. As a consequence, the verification of the properties expressing data security requirements may lead to results that are not accurate and/or representative enough. In addition, designing a fine-grained modeling of heterogeneous aspects (e.g., hardware architecture, scheduling policy of a processor, and software behavior) of a system in a same modeling language can be time-consuming and error-prone, and the resulting model can be complex to understand as it encompasses heterogeneous aspects in a single modeling view. In addition, gathering all the modeled aspects in a single fine-grained model may also lead to a state-space explosion at verification stage due to the potential multiplication of states, variables, etc.

The aim of W-Sec is precisely to address these lacks. This requires more suitable modeling formalisms and tools that enable for designing and verifying formal models in a fine-grained way while enabling for capturing heterogeneous aspects without complexifying the models. SysML-Sec [4] provides a framework and a method for designing safe and secure embedded systems. For these needs, SysML-Sec relies on two enhanced and formally defined SysML profiles that enables for modeling high-level system architectural and behavioral aspects, as well as fine-grained hardware ones. SysML-Sec is fully supported by the toolkit TTool

that provides a graphical and easy-to-use interface for designing and verifying models with the two SysML-Sec profiles: direct model-checking [9] and simulation can be performed with TTool. In addition, TTool provides two distinct modeling views (one per SysML profile) that help simplifying the system models, and is tailored for producing joint safety, security and performance analyses [3], thus spanning the three critical assessment dimensions of a countermeasure. For these reasons, have chosen TTool and its tailored formal SysML profiles as the underlying formalism and toolkit for designing a method improving our previous one [26]. In a complementary way, W-Sec also complements the SysML-Sec method. Indeed, the attack model considered in SysML-Sec is the Dolev-Yao one [11]. Therfore other kinds of attacks such as "sequences of exploitation of vulnerabilities of several components" [4] are considered out of scope. TW-Sec then widens the considered attack corpus as it considers modular attack scenarios like in [26].

6 Discussion

6.1 Regarding our Previous Contributions

As we explained in [25], W-Sec merges contributions from SysML-Sec and [26] (see Table 4). As a result, W-Sec brings several improvements to both of the methods. With regards to [26]:

- W-Sec "reduces the models complexity since 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 assessment at system-level 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, W-Sec 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)." [25]

In addition, as explained in Sect. 5, W-Sec also complements the SysML-Sec approach thanks to the consideration of modular attack scenarios.

Modeling formalism	From SysML-Sec and [12]
Two-views modeling	From SysML-Sec
Use of HSW-View for low-level component modeling	W-Sec contribution
Separation of safety concerns at system level in S-View vs. security and performance aspects at component-level in HSW-View	W-Sec contribution
Attacks and countermeasures modeling through model mutations	From [26]
S-Models and HSW-Models mutations definitions and operators	W-Sec contribution
Impact assessment approach	From [26] (composition of the sets of mutant system models, comparison of verification results)

Table 4. Comparison with SysML-Sec and [26] (table expanded from [25])

6.2 Regarding IT'm Factory Case-Study Results

W-Sec has been evaluated with the IT'm Factory's packaging chain case-study. This evaluation provides interesting results, complementary to those discussed in [25] where W-Sec is evaluated with a rover swarm case-study. Indeed, even if these two systems have common aspects (cyber-physical systems, safety-critical systems, etc.) they also have significant differences : contrary to the rover swarm, the packaging chain is a real system, it relies on highly specific devices like PLCs, the source code of its control algorithms was provided in a domain-specific language —*ladder logic*—, etc.).

The first interesting conclusion is that the W-Sec modeling approach is well suited for modeling industrial systems, especially PLCs. Designing the PLC HSW-Models was really natural indeed, since ladder logic includes low-level instructions that can easily be converted in complexity operators (see Fig. 6). Moreover, the sequential logic of PLC algorithms is particularly consistent with the semantics of tasks and activity diagrams in HSW-Models. However, the modeling stage is still an engineering task that may require a lot of time depending on the modeled system. Given the relative semantic proximity between ladder logic and our modeling formalism, the (partial) automation of the models generation from PLC source-code is an interesting research direction we intend to investigate.

Also regarding the modeling stage, we have been able to optimize the S-Models to avoid combinatorial explosion at verification stage, while this issue occurred in the rovers case-study [25]. Yet, both the rover swarm and the packaging chain necessitate "environment" blocks that compute the evolution of the physical parameters of the systems, leading to potential highly-complex state space graphs. This shows that W-Sec can be suitable for complex systems of systems, provided the S-Models are wisely designed.

Four countermeasures were evaluated in this case-study. They target different components, implement different mechanisms and belong to two categories: software patches (**C1**, **C2** and **C3**) and hardware temporary workaround (**C4**). For each of these countermeasures, we were able to provide appropriate models with HSW and S-Models mutations that seem well suited for the countermeasures modeling needs. Notwithstanding these encouraging results, applying these mutations still requires human intervention and thus can be time-consuming. We are currently working on a mutation language and compiler that will enable for automating this process: the new formalization provided in Sect. 3 is a good step towards this automation.

	P1	P2	P3
Sc1	✓	✓	✓
Sc2	✓	✓	✓
Sc2 + Attack	✗	✗	✓

(a) without countermeasure

	P1	P2	P3
Sc1	✓	✓	✓
Sc2	✓	✓	✓
Sc2 + Attack	✓	✓	✓

(b) **C1** (offset check)

	P1	P2	P3
Sc1	✓	✓	✓
Sc2	✓	✓	✓
Sc2 + Attack	✗	✓	✓

(c) **C2** (encryption)

	P1	P2	P3
Sc1	✓	✓	✓
Sc2	✓	✓	✓
Sc2 + Attack	✓	✓	✓

(d) **C3** (static ARP table)

	P1	P2	P3
Sc1	✓	✓	✗
Sc2	✗	✓	✗
Sc2 + Attack	✗	✓	✗

(d) **C4** (unplugging)

Table 5. Full safety assessment results

	No countermeasure	**C1**	**C2**	**C3**	**C4**
PLC	608.5 ns	610.7 ns	690 ns	608.5 ns	608.5 ns
SCADA	42.5 ns	42.5 ns	170.8 ns	40.6 ns	42.5 ns

Table 6. Full performance assessment results

	No countermeasure	**C1**	**C2**
SCADA → Main switch	Weak auth.: ✗ Strong auth.: ✗	Weak auth.: ✗ Strong auth.: ✗	Weak auth.: ✓ Strong auth.: ✗
Local switch → PLC	Weak auth.: ✗ Strong auth.: ✗	Weak auth.: ✗ Strong auth.: ✗	Weak auth.: ✓ Strong auth.: ✗

	C3	**C4**
SCADA → Main switch	Weak auth.: ✗ Strong auth.: ✗	Weak auth.: ✗ Strong auth.: ✗
Local switch → PLC	Weak auth.: ✗ Strong auth.: ✗	Weak auth.: ✗ Strong auth.: ✗

Table 7. Full security assessment results

The third W-Sec stage (i.e., impact assessment) provided results that are presented in Tables 5, 6 and 7. As in [25], these results shows the relevance of a joint safety/security/performance assessment of countermeasures. Indeed, a mere

safety analysis would lead to chose the countermeasures **C1** and **C3**: the security analysis yet shows that the system with these countermeasures is still vulnerable to a Dolev-Yao attacker on the two critical data channels studied. Similarly, a single performance analysis would lead to select **C3** and **C4**, although the safety verification results show an important regression for **C4**. Finally the security analysis alone favors **C2**, which leads to a safety regression on one property with respect to **C1** and **C3**.

Relatedly, these results illustrate a current lack of our method. Indeed, comparing the assessment results to find the optimal countermeasure may be a difficult problem since it requires to rank very different properties between them (e.g., is it more important to ensure that no grain can be discharged on the conveyor belt, to minimize the duration of a PLC algorithm iteration or to ensure that one of the communication channels is protected with integrity?). We tried to address a similar problem in [26], but the proposed metrics was based on the definition of a strict total order on the importance of properties and functional scenarios (called *missions*), that may lack in realism [24]. An interesting research direction could be to affect weights to functional scenarios and properties, enabling for a non-strict ordering of properties and scenarios and defining evaluation formulae based on these weights.

Concerning the results again, note that it is necessary to put the SCADA performance results into perspective: unlike the PLC HSW-Model, the SCADA console HSW-Model is not based on a real system description nor on a real algorithm. We have arbitrarily considered that the supervision application runs on a standard host (Core i5 CPU), and we have modeled a fictitious algorithm creating a network message and sending it. Therefore, the SCADA performance results are far less likely than the PLC ones that are based on the real system. They should only be seen as complementary results that illustrate how the method can deal with several HSW-Models.

7 Conclusions and Future Works

Extending the works presented at ModelsWard 2022 [25], this chapter provides a detailed and formalized description of W-Sec, a formal model-based method for assessing the safety, security and performance impacts of security countermeasures. It also defines three consistency rules that help ensuring the consistency of the models used in the method. For those purposes, it also provides new mathematic definitions of the two SysML profiles used in SysML-Sec and W-Sec. W-Sec has now been evaluated with two case-studies, a rover swarm and a connected factory's packaging chain: the chapter presents the results of the latter and discusses the relevance of the method for this new application field. It also discusses the current limitations of W-Sec, and gives several research perspectives.

We are currently investigating two of them. Firstly, the automation of the second (mutation) W-Sec stage: the mathematic definitions given in Sect. 3 enable for formally defining each mutation operator and designing a mutation

language. A compiler for this language is currently being implemented in TTool. Secondly, we are studying the semantic links between the ladder logic language, used for programming Siemens PLCs, and the two SysML profiles we use. This is the first step towards the automation of the first (modeling) W-Sec stage in the context of industrial control systems. A third interesting research perspective we intend to investigate is the design of tailored metrics enabling for an easy comparison of the impact assessment results. We also plan to evaluate W-Sec with further attack and countermeasures scenarios on our connected factory case-study.

References

1. Abbasi, A., Scharnowski, T., Holz, T.: Doors of Durin: The Veiled Gate to Siemens S7 Silicon. BlackHat Europe (2019)
2. Aichernig, B.K., Lorber, F., Ničković, D.: Time for mutants—model-based mutation testing with timed automata. In: International Conference on Tests and Proofs. pp. 20–38. Springer (2013)
3. Apvrille, L., Li, L.W.: Harmonizing safety, security and performance requirements in embedded systems. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1631–1636. IEEE (2019)
4. Apvrille, L., Roudier, Y.: SysML-Sec: A SysML Environment for the Design and Development of Secure Embedded Systems. In: APCOSEC 2013. Yokohama, Japan (Aug 2013), https://hal.telecom-paris.fr/hal-02288385
5. Apvrille, L., de Saqui-Sannes, P., Hotescu, H., Tempia-Calvino, A.: SysML Models Verification Relying on Dependency Graphs. In: MODELSWARD. pp. 174–181 (2022)
6. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on UPPAAL. Formal methods for the design of real-time systems pp. 200–236 (2004)
7. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: 14th IEEE Computer Security Foundations Workshop (CSFW-14). pp. 82–96. IEEE Computer Society, Cape Breton, Nova Scotia, Canada (Jun 2001)
8. Brykczynski, B., Small, R.A.: Reducing internet-based intrusions: Effective security patch management. IEEE software **20**(1), 50–57 (2003)
9. Calvino, A., Apvrille, L.: Direct Model-checking of SysML Models. In: Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,. pp. 216–223. INSTICC, SciTePress (2021). https://doi.org/10.5220/0010256302160223
10. Calvino, A.T., Apvrille, L.: Direct model-checking of SysML models. In: 9th International Conference on Model-Driven Engineering and Software Development. pp. 216–223. SCITEPRESS-Science and Technology Publications (2021)
11. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–208 (1983). https://doi.org/10.1109/TIT.1983.1056650
12. Enrici, A., Apvrille, L., Pacalet, R.: 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 (2017)
13. Enrici, A., Li, L., Apvrille, L., Blouin, D.: A Tutorial on TTool/DIPLODOCUS: an Open-source Toolkit for the Design of Data-flow Embedded Systems. Tech. rep. (2022)

14. Gonzalez-Granadillo, G., Garcia-Alfaro, J., Alvarez, E., El-Barbori, M., Debar, H.: Selecting optimal countermeasures for attacks against critical systems using the attack volume model and the RORI index. Computers & Electrical Engineering **47**, 13–34 (2015)
15. Jawad, A., Jaskolka, J.: Analyzing the impact of cyberattacks on industrial control systems using timed automata. In: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS). pp. 966–977 (2021). https://doi.org/10.1109/QRS54544.2021.00106
16. Jawad, A., Newton, L., Matrawy, A., Jaskolka, J.: A formal analysis of the efficacy of rebooting as a countermeasure against iot botnets. In: 2022 IEEE Conference on Communications, ICC 2022
17. Li, L.: Safe and secure model-driven design for embedded systems. Ph.D. thesis, Université Paris-Saclay (Sep 2018)
18. Lugou, F., Li, L.W., Apvrille, L., Ameur-Boulifa, R.: SysML models and model transformation for security. In: 2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD). pp. 331–338. IEEE (2016)
19. McLaughlin, S., Konstantinou, C., Wang, X., Davi, L., Sadeghi, A.R., Maniatakos, M., Karri, R.: The cybersecurity landscape in industrial control systems. Proceedings of the IEEE **104**(5), 1039–1057 (2016)
20. Nespoli, P., Papamartzivanos, D., Mármol, F.G., Kambourakis, G.: Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks. IEEE Communications Surveys & Tutorials **20**(2), 1361–1396 (2017)
21. von Neumann, J., Burks, A.W., et al.: Theory of self-reproducing automata, vol. 1102024. University of Illinois press Urbana (1966)
22. Nicol, D.: Modeling and simulation in security evaluation. IEEE Security & Privacy **3**(5), 71–74 (2005). https://doi.org/10.1109/MSP.2005.129
23. Pedroza, G., Apvrille, L., Knorreck, D.: Avatar: A sysml environment for the formal verification of safety and security properties. In: 2011 11th Annual International Conference on New Technologies of Distributed Systems. pp. 1–10. IEEE (2011)
24. Sultan, B.: Maîtrise des correctifs de sécurité pour les systèmes navals. Ph.D. thesis, Ecole nationale supérieure Mines-Télécom Atlantique Bretagne Pays de la Loire (2020)
25. Sultan, B., Apvrille, L., Jaillon, P.: Safety, Security and Performance Assessment of Security Countermeasures with SysML-Sec. In: Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development - MODELSWARD. pp. 48–60. INSTICC, SciTePress (2022). https://doi.org/10.5220/0010832300003119
26. Sultan, B., Dagnat, F., Fontaine, C.: A Methodology to Assess Vulnerabilities and Countermeasures Impact on the Missions of a Naval System. In: Computer Security. pp. 63–76. Springer International Publishing, Cham (2018)