Towards Automating Safety and Security Co-Analysis with Patterns

Yuri Gil Dantas
fortiss GmbH
Munich, Germany
dantas@fortiss.org

Antoaneta Kondeva
fortiss GmbH
Munich, Germany
kondeva@fortiss.org

Vivek Nigam
fortiss GmbH
Munich, Germany
nigam@fortiss.org

***Abstract*—This article presents the first results towards automating safety and security co-analysis with patterns.**

***Index Terms*—safety, security, co-analysis, automation**

I. Introduction

Our vision is to provide methods for automating safety and security co-analysis with patterns. These methods shall incorporate safety and security reasoning principles and take into account the trade-offs between safety and security. The remainder of this section motivates why such automated methods are needed.

System interconnectivity has been a motivating factor behind the evolution of, *e.g.*, autonomous cars. This interconnectivity, however, leads to new challenges for safety and security. That is, an intruder might cause catastrophic events by remotely targeting safety-critical systems. For example, an intruder might exploit a connection vulnerability in an autonomous car to remotely disable safety features, such as airbags, to put passengers in danger [1].

A better integration between safety and security is then appealing. Standards and guidelines for avionics [2] and automotive [3] industries have already taken steps towards this integration. They specify interaction points between the analyses performed by safety and security engineers. That is, when information gathered by safety engineers shall be made available to security engineers and vice versa [1]. The goal is a co-analysis between safety and security engineers to address, respectively, malfunctioning behavior and intentionally caused harm on safety-critical systems.

Such a co-analyses can, however, lead to at least three interrelations: There can be (1) conflicts, (2) synergies or (3) no conflicts between safety and security analyses. The challenge is to understand what are the trade-offs between safety and security analyses, and how to proceed when conflicts or synergies are found.

Our ultimate goal is to provide automated methods for safety and security co-analysis that accounts for trade-offs. Before achieving this goal, we first investigate how much of the safety analysis and security analysis w.r.t pattern selection can be automated.

Safety engineers commonly use hazard analysis and risk assessment (HARA) to identify the main hazards that might potentially cause harm. To control the identified hazards, safety engineers may use safety architectural patterns [5] (*e.g.*, watchdogs or safety monitors). Security engineers focus on threat detection and mitigation under the presence of an intruder, using, *e.g.*, threat assessment and remediation analysis (TARA). Security engineers may use security patters (*e.g.*, firewall or encryption) to mitigate the identified threats.

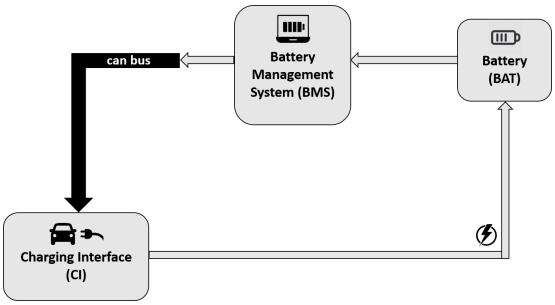

Fig. 1: Battery Management System (BMS) functional architecture

Currently, however, safety and security analyses are mostly performed manually, in particular, the analysis performed by safety engineers. That is, the reasoning of which pattern to use at which part of the target system to control which hazard is documented mostly in textual form or by means of models, such as GSN-models [7], with limited support for automation. As a result, it is not possible to automatically check whether all hazards have been properly controlled by, *e.g.*, safety patterns.

This article presents in a nutshell the first results towards achieving our vision. We provide safety and security reasoning principles with patterns during the definition of system architecture for embedded systems. We specify these principles using logic and logic programming as they are suitable frameworks for the specification of reasoning principles as knowledge bases and using them for automated reasoning [4]. We validate our current results with an example of safety-critical embedded system taken from the automotive domain.

II. Running Example

We consider a simplified Battery Management System (BMS) responsible for controlling a rechargeable electric car battery [8]. The BMS is a critical system as harm, *e.g.*, battery

Identify applicable funding agency here. If none, delete this.

explosions, may occur if it does not compute the charging state of the battery correctly.

Figure 1 depicts the main functions composing the BMS. The charging interface (CI) represents the interface at the charging car station. This interface is triggered while recharging the battery (BAT) of the car. BMS receives relevant information (*e.g.*, voltage and temperature values) from BAT so that it can compute the charging state of BAT. Depending on the state of BAT, BMS sends signals of activation or deactivation of the external changer to CI. These signals are sent though a CAN bus.
To address the safety of the BMS, safety analyses are carried out to determine main hazards. The main hazard is:

H0$_{bms}$: The BAT is overcharged leading to its explosion.

We identify one erroneous hazard **H1**$_{bms}$ that may lead to **H0**$_{bms}$. The word erroneous is used by safety engineers to describe hazards: *erroneous* is used when a function is working but not correctly.

- **H1**$_{bms}$**– Erroneous CI:** The CI sends charging signals when BAT is fully charged.

The following two hazards may lead to **H1**$_{bms}$.

- **H1.1**$_{bms}$**– Erroneous BMS:** The BMS sends wrong signals to CI;
- **H1.2**$_{bms}$**– Erroneous CAN:** The CAN bus sends wrong signals to CI;

III. Reasoning Principles

We are developing a domain-specific language, called **SafSecPat**, for enabling automated safety and security reasoning with patterns. We only present the main features of **SafSecPat**.

In **SafSecPat**, one can specify the main elements of a functional architecture. These main elements include components, sub-components, channels, and information flows. A *component* is a function in the system, *e.g.*, BMS from the running example, and a *sub-component* is a sub-function of the function. A *channel* is a logical channel connecting an output of one function to an input of the other function (*e.g.*, the channel from BAT to BMS). Notice that it denotes a unidirectional connection. An *information flow* denotes a flow path from one function to another function, *e.g.*, an information flow from BAT to CI is specified as $[\mathsf{BAT}, \mathsf{BMS}, \mathsf{CI}]$.

A. Safety Reasoning

Our goal is to provide automated methods for (a) checking whether hazards can be controlled by a safety pattern and (b) placing suitable safety patterns in a system architecture. We describe in a nutshell how we achieved our goal.

We introduce four new elements from **SafSecPat**:

- $\mathsf{hz}(\mathsf{id}_H, \mathsf{id}_c, \mathsf{htp}, \mathsf{sv})$ is a hazard associated with the function id_c (*e.g.*, BMS) of the hazard type htp (*e.g.*, erroneous), and severity sv (*e.g.*, catastrophic).
- $\mathsf{subHz}(\mathsf{id}_1,\mathsf{id}_2)$ denotes a sub-hazard in which id_1 is a hazard causing hazard id_2.
- $\mathsf{ctl}(\mathsf{id}_H, \mathsf{id}_c, \mathsf{htp}, \mathsf{sv})$ denotes that the hazard id_H of type htp, severity sv and associated with the function id_c can be controlled.
- $\mathsf{nctl}(\mathsf{id}_H, \mathsf{id}_c, \mathsf{htp}, \mathsf{sv})$ denotes that the hazard id_H of type htp, severity sv and associated with the function id_c can be not controlled.

We also distinguish hazards into two types: *basic* hazards and *derived* hazards. A hazard is classified as *basic* if it does not have any sub-hazards (hazard that may lead to the main hazard), and *derived* otherwise. A derived hazard is not controlled if any one of its sub-hazards is not controlled.

Currently, we have specified five safety patterns: Heterogeneous Duplex Redundancy (HDR), Triple Modular Redundancy (TMR), N-Version Programming (NProg), Safety Monitors (SafMon), and Watchdog (WD)[1].

In **SafSecPat**, we specify reasoning principles to use safety patterns to control identified hazards. In the following, we show such reasoning principles for the safMon:

```
ctl(ID,CP,err,SV) :-
  hz(ID,CP,err,SV),
  safMon(ID2,CP,_,_,_,_,_,_),
  not inpNotCovSF(ID2),
  not outNotCovSF(ID2).
```

It specifies that a hazard associated to a function `CP` of type erroneous can be controlled if a safety monitor is associated to `CP` provided `not inpNotCovSF(ID2)` and `not outNotCovSF(ID2)`: there are no input logical channels, *i.e.*, channels incoming to `CP` specified by `ch(CH,_,CP)`, not taken as input to the safety monitor, nor output channels *i.e.*, channels outgoing from `CP` specified by `ch(CH,CP,_)`.

We use ASP/DLV semantics [9] to automate the recommendation of safety patterns in a given architecture. The following rule specifies the placement or not of a safMon, denoted by `nsafMon`, associated with the function `CP` that is furthermore associated with a basic or not controlled hazard `ID`:

```
safMon(nuSafMon,CP,allInp,allOut,
   nuSC,numin,numout,numcp) v
nsafMon(nuSafMon,CP,allInp,allOut,
   nuSC,numin,numout,numcp)
:- cp(CP),hz(ID,CP,err,SV),
   basicOrNCTL(ID,CP,err,SV),
   explore(N,safMon).
```

We assume here that the constants starting with `nu` are fresh, *i.e.*, do not appear in the given architecture, thus used only for recommended safety patterns. Since it is enough to know to which function a safety monitor is associated to, we do not need to enumerate all the inputs and outputs of `CP`, but rather simply denote `CP`'s inputs and outputs using, respectively, the fresh constants `allInp` and `allOut`.

In summary, we specify safety reasoning principles to determine when a hazard can be controlled or not, including reasoning principles used to decide when a safety pattern can

[1]We refer the reader to [5] for detailed description of these patterns.

be used to control a hazard. Moreover, our machinery identifies which safety patterns can be used and where exactly they should be deployed to control hazards that have not yet been controlled. With our machinery, one might receive a number of options where to place safety patterns to control identified hazards. This enables a safety engineer to understand which options of safety patterns he can use to control hazards and decide which one is more appropriated given factors, such as costs, hardware availability.

B. Security Reasoning

Our goal is to provide automated methods for (a) checking whether a threat can be mitigated by a security pattern and (b) placing suitable security patterns in a system architecture. We present the first results towards achieving (a).

Our *threat model* is an intruder who is trying to carry out an attack by accessing an untrusted or vulnerable interface (*i.e.*, a channel or a component).

In **SafSecPat**, we specify the elements public and bdCh. The element public is either a physical communication channel (*e.g.*, CAN) or a component (*e.g.*, CI) that may be accessible by external users (possibly an intruder). A bdCh denotes a boundary where the system changes its level of security. In our language, security patterns shall only be placed in the security boundaries. The specification of boundaries constraints the number of design options a security engineer can choose from. In the same way, the use of boundaries shall reduce the search space of our machinery for recommending security patterns. These methods for automatically placing security patterns are yet to be specified in **SafSecPat**.

To achieve goal (a), we specify the following elements:

- $\mathsf{potThreat}(\mathsf{id}_{PT}, \mathsf{id}_c, \mathsf{ttp})$ is a potential threat associated with the function id_c of the threat type ttp (*e.g.*, availability or integrity).
- $\mathsf{reachI}(\mathsf{id}, L)$ denotes that id can be reached by an intruder through a path L (sequence of components in a communication channel).
- $\mathsf{threat}(\mathsf{id}_T, \mathsf{id}_c, \mathsf{ttp})$ is an actual threat associated with the function id_c of the threat type ttp.
- $\mathsf{mitigated}(\mathsf{id}_T, L)$ denotes that the threat id_T that is associated with a path L can be mitigated.

A potential threat associated to a function id_c is true if there is a hazard associated with id_c. Moreover, a potential threat becomes an actual threat if id_c is reachable by an intruder.

We have so far only specified firewall as security pattern. We specify firewall as a security pattern able to mitigate threats of the type integrity. In **SafSecPat**, this is specified as follows:

```
mitigated(IDT, L) :-
 firewall(ID,Comm,CP,_,_,_,FWCP),
 threat([IDT,L,int]),
 #subList([Comm,CP],L).

mitigated(IDT, L) :-
 firewall(ID,Comm,CP,_,_,_,FWCP),
 threat([IDT,L,int]),
 #subList([CP,Comm],L).
```

It specifies that the firewall can mitigate threats that are associated with the path L.

In summary, we specify security reasoning principles to automatically determine when a threat can be mitigated or not by a security pattern. Methods to automatically place security patterns in a given architecture are left to future work.

C. Safety and Security Co-Analysis

Our ultimate goal is to provide methods to automate safety and security co-analysis taking into account their trade-offs. Such methods are been developed, hence are not shown here. Instead, we briefly discuss interrelations between safety and security that one needs to consider during a co-analysis.

There can be synergies between safety and security: A system component can be safety solution and security measure at the same time. For example, the safety analysis may determine the need for verifying the integrity of a network message. The standard solution is to use a checksum to verify integrity. However, if the message also needs to be encrypted for security reasons, then one can do without the checksum, as decryption may already include an integrity check.

There can be conflicts between safety and security: A measure that is aimed at guaranteeing safety might conflict with a security measure and vice versa. For example, if a firewall is placed between BMS and CI (example presented in Section II) to avoid that an intruder can access the CAN bus via CI (only component accessible by external users). Then this decision might result in a new hazard as the firewall might incorrectly block signals from BMS.

IV. Case Study

This section illustrates the results of our machinery for the BMS example described in Section II. To illustrate our results, we assume a co-analysis between safety and security engineers. We also assume that such safety and security engineers use our machinery. Our results are depicted as dark gray boxes, and the channels related (inputs or outputs) to safety patterns are depicted as dashed arrows.

a) Results from Safety Analysis: We identified an erroneous ($\mathbf{H1}_{bms}$) hazard on CI, as described in Section II. This erroneous hazard ($\mathbf{H1}_{bms}$) is broken down into two sub-hazards, namely erroneous BMS ($\mathbf{H1.1}_{bms}$), and erroneous CAN ($\mathbf{H1.2}_{bms}$). Typically, hazards on CAN buses can be controlled by replacement only. Hence, we assume that $\mathbf{H1.2}_{bms}$ has already been controlled.

Our machinery yielded *two solutions* (*i.e.*, architectures) to control $\mathbf{H1}_{bms}$, and $\mathbf{H1.1}_{bms}$. We only show one of those solutions. The architecture of the chosen solution is depicted on left-hand side of Figure 2. Our machinery recommended to use a safMon to control the identified hazard ($\mathbf{H1.1}_{bms}$), and consequentely $\mathbf{H1}_{bms}$. This safMon monitor shall monitor the behavior of BMS by checking its outputs.

b) Results from Security Analysis: CI is considered the only function accessible by external users (*e.g.*, drivers). Then, CI is specified as an actual threat in **SafSecPat**. Allowing an

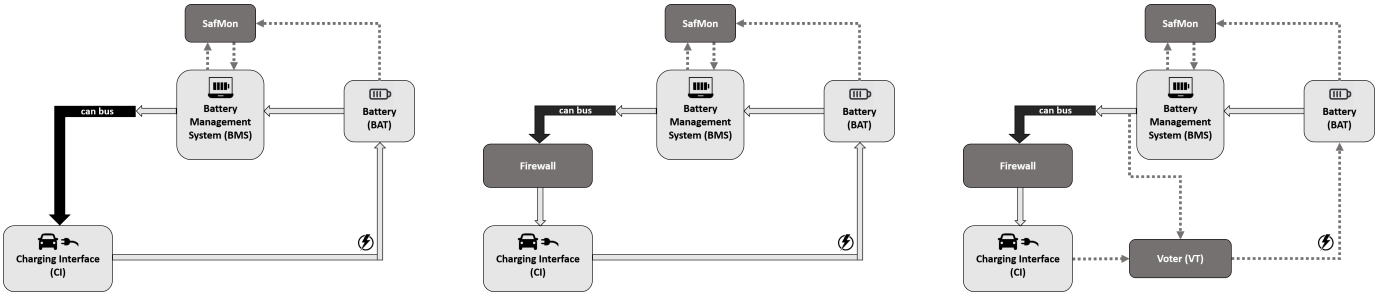

Fig. 2: BMS functional architecture after safety analysis, security analysis and safety analysis, respectively.

intruder to access CI might lead to an attack, since an intruder may access the CAN bus through CI.

To avoid this potential attack, a Firewall is placed between BMS and CI.[2] The updated architecture with a Firewall is illustrated in the center of Figure 2. The choice of adding a Firewall as security pattern is currently done manually in **SafSecPat**, as methods to automatically place security pattern are being developed. Our security reasoning principles confirms, however, that using a Firewall between BMS and CI can mitigate the identified threat.

c) Results from Safety Analysis: Upon receiving the results from the security engineers, safety engineers can (manually) identify a new hazard that may be triggered by the Firewall. That is, Firewall might incorrectly block signals from BMS. This new hazard (**H1.3**$_{bms}$) is of the type omission and may lead to **H1**$_{bms}$. We use the word *omission* as hazard whenever a function is not provided when expected.

We run our machinery to automatically identify what safety patterns could be used to control **H1.3**$_{bms}$, taking as input the architecture illustrated in the center of Figure 2. Our machinery yielded four solutions to control **H1.3**$_{bms}$. We only show one of those solutions. The architecture of the chosen solution is depicted on right-hand side of Figure 2.

Our machinery recommend to use HDR to improve safety by path redundancy. HDR increases the redundancy of paths in the system in case messages are lost or incorrectly computed. Figure 2 illustrates that BMS and CI sent redundant inputs to Voter so that BAT has a higher chance of getting the expected input. That is, if CI does not send the input to BAT due to, *e.g.*, an omission from Firewall, BAT receives the expected input from BMS through Voter.

V. Related Work

A combined safety and security pattern engineering workflow has been recently proposed [8]. In particular, this article discusses the selection of safety and security patterns for automotive system engineering. The selection of safety and security is performed manually, in contrast to our article. A survey on approaches that aim to combine safety and security concerns for industrial infrastructures is presented here [10]. This survey confirms that finding interdependencies between safety and security is challenging due, *e.g.*, to their diversity. Previously, we have applied techniques to automatically generate ADTs from GSNs annotated with lightweight semantics for an Industry 4.0 application [1].

[2]We refer the reader to [8] for more insights on why adding a Firewall between BMS and CI makes the system more secure.

VI. Conclusion

This article presented in a nutshell the first results towards automating safety and security co-analysis. That is, we presented the current status of our safety and security reasoning principles. Our safety reasoning allows one to automatically place suitable safety patterns in a given architecture in order to control identified hazards. Our security reasoning allows one to automatically check whether an identified threat can be mitigated by a given security pattern.

We are currently investigating a number of future directions to achieve our vision. We are extending our security reasoning. We are implementing methods to automatically place suitable security patterns in a given architecture as well as increasing the number of security patterns specified. We are also investigating the possible trade-offs between safety and security as well as how these trade-offs can affect our machinery.

References

[1] A. Kondeva, V. Nigam, H. Ruess, C. Crlan: On Computer-Aided Techniques for Supporting Safety and Security Co-Engineering. ISSRE Workshops 2019: 346-353

[2] ED 202A: Airworthiness security process specification. https://standards.globalspec.com/std/9862360/eurocae-ed-202

[3] SAE J3061: Cybersecurity guidebook for cyber-physical vehicle systems. https://www.sae.org/standards/content/j3061/.

[4] Baral, C. Knowledge Representation, Reasoning and Declarative Problem Solving. In CUP 2010.

[5] Preschern , C., Kajtazovic, N., and Kreiner, C. Security Analysis of Safety Patterns. In PLoP 2013.

[6] I. S. Jacobs and C. P. Bean. "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[7] GSN Community Standard Version 1, 2011. Available at https://shorturl.at/AMRV4

[8] H. Martin and Z. Ma and Ch. Schmittner and B. Winkler and M. Krammer and D. Schneider and T. Amorim, G. Macher, Ch. Kreiner. Combined Automotive Safety and Security Pattern Engineering Approach. Reliability Engineering & System Safety 2020.

[9] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello: The DLV System for Knowledge Representation and Reasoning. ACM Trans. Comput. Log. 7(3). 2006

[10] S. Kriaa, L. Pietre-Cambacedes, M. Bouissou, and Y. Halgand. A Survey of Approaches Combining Safety and Security for Industrial Control Systems. Reliability Engineering & System Safety, 2015.