Which Attacks Lead to Hazards?

Combining Safety and Security Analysis for Cyber-Physical Systems

Luca Maria Castiglione and Emil C. Lupu

Abstract—Cyber-Physical Systems (CPS) are exposed to a plethora of attacks and their attack surface is only increasing. However, whilst many attack paths are possible, only some can threaten the system's safety and potentially lead to loss of life. Identifying them is of essence. We propose a methodology and develop a tool-chain to systematically analyse and enumerate the attacks leading to safety violations. This is achieved by lazily combining threat modelling and safety analysis with formal verification and with attack graph analysis. We also identify the minimum sets of privileges that must be protected to preserve safety. We demonstrate the effectiveness of our methodology to discover threat scenarios by applying it to a Communication Based Train Control System. Our design choices emphasise compatibility with existing safety and security frameworks, whilst remaining agnostic to specific tools or attack graphs representations.

Index Terms—Security, Safety, Formal Verification, Attack Graphs, Cyber Physical Systems.

1 Introduction

In February 2021, an individual gained unauthorised access to the computer network managing Florida's water supply and tried to abuse the treatment software to inject dangerous levels of lye in the water [1]. Along with other similar incidents such as [2] [3] [4], this case serves as a reminder that attacks against Cyber Physical Systems (CPSs) can impact the safety of the controlled processes and result in monetary losses, significant physical damages to infrastructure and, in some cases, loss of life. As operators of Industrial Control Systems deploy new families of network enabled Operational Technology (OT) devices to augment connectivity and minimise costs, the attack surface also increases, allowing for a constantly growing variety of attacks. Whilst more attacks are becoming possible, only some of them will impact the safety of the system. For example, losing the availability of a non-critical components can reduce the quality of the provided service but not threaten safety. Similarly, a breach of confidentiality can result in a data loss which, generally, is not a safety critical event. There is no doubt that identifying the attacks that lead to safety violations is a priority. However, the complexity of modern CPSs renders this task extremely difficult. Sophisticated attacks often rely on the effect of apparently legitimate commands that, interleaved with system behaviour, trigger cascading effects within the CPS with the aim of rendering it vulnerable to further attacks and eventually cause losses. For example, in [5] the adversary relies on a vulnerability affecting the alternate mode of functioning of an aircraft to hijack it. By changing the environment around the aircraft (communication jamming), they were able to trigger a legitimate transition of operating mode, where a spoofing attack becomes successful. To prevent these scenarios, tools and methodologies need to be developed that support complex integrated analyses of safety and security in the context of the behaviour of the system.

In this work, we address the following questions: *(RQ1) How do we identify the attacks leading to violation of the system level safety properties?*, *(RQ2) Which attack paths allow these attacks*, and *(RQ3) Which vulnerabilities should be remediated first to mitigate these paths and preserve system safety?* To answer these questions we have developed Tiresias [1], a methodology (Figure 1) that combines System Theoretic Process Analysis Security (STPA-Sec) [6] and STRIDE [7] with formal verification and attack graph analysis in novel ways. The process starts with preparatory step (❶) where

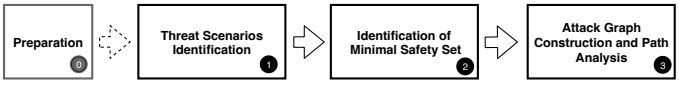

Fig. 1. Overview of the proposed methodology.

we use System Theoretic Accident Models and Processes (STAMP) to model safety aspects of the system, and perform part of STPA. Unlike traditional frameworks such as HAZOP [8], FMEA/FMECA [9] [10], and FTA [11], STAMP models the safety as a control problem and sees accidents as the result of flaws in the interactions between system components. We start with the identification of threat scenarios (❶), where certain sequences of malicious actions can result in catastrophic consequences. Our approach grounded in STPA-Sec and formal verification enables us to automatically identify threat scenarios, answering the first research question. (❷) Then, we define and solve a Boolean satisfiability (SAT) problem to find the minimum set of privileges that should be protected to prevent the attacker to trigger threat scenarios. (❸) We leverage the integration with the architecture of the CPS to construct, starting from the set of privileges, a system wide attack graph [12]. The resulting graph encompasses feasible attack paths that are relevant

- *Authors are with Department Computing, Imperial College London, UK. E-mail: {l.castiglione, e.c.lupu} @imperial.ac.uk*
- *The support of the EPSRC Centre for Doctoral Training in High Performance Embedded and Distributed Systems (HiPEDS, Grant Reference EP/L016796/1) is gratefully acknowledged.*

1. In Greek mythology, Tiresias was a blind prophet renowned for clairvoyance.

to system safety. Finally, we analyse the attack paths found and elaborate on mitigation strategies. The output of our methodology is a set of safety critical attack paths in the CPS. More broadly, the solutions found reveal the weight of individual vulnerabilities and weaknesses with respect to the preservation of system level safety properties. In this work, we bring the following contributions:

- We build a clear correspondence between the safety model, security threats and system vulnerabilities.
- We define a methodology to systematically derive sequences of threats that, in combination with the behaviour of the system, can lead to safety violations (threat scenarios).
- We develop and present a tool-chain that employs formal verification to identify and enumerate threat scenarios - also considering the behaviour of system components.
- Threat enumeration is lazily[2] combined with the generation of *attack graphs*. The analysis of latter enables us to verify the feasibility of complex attacks and to evaluate the importance of individual vulnerabilities with respect to consequences.
- We identify the minimum sets of privileges that must be protected to preserve safety.
- We demonstrate the effectiveness of our methodology applied to a realistic use cases, a *Communication Based Train Control System*.

Novelty This works introduces the following novel aspects. First, we combine STPA-Sec with threat modelling and formal verification. The analysis of the combined models allows to enumerate complex sequences of attack steps leading to system-level losses. The second novel aspect consists in the integration of the results obtained from the analysis of functional models with attack graphs. In doing so, we can verify the feasibility of safety critical attacks, but also evaluate the importance individual vulnerabilities have in enabling these attacks and, thus, help prioritise remediation. Finally, to the best of our knowledge, our work is the first to analyse the impact of all generic threats identified in a threat analysis on emergent safety properties of the CPS.

Impact The impact of this work is threefold. Firstly, it enables to enumerate the *threat scenarios*, complex sequences of threats, which when exploited and combined with system behaviour lead to hazards. Secondly, formal verification guarantees the soundness of these scenarios. We provide a clear map between elements of the safety model, and between safety and security models so that they can be independently replaced and/or improved. Thirdly, by decoupling the formal verification from the attack graph analysis, we can evaluate the impact of vulnerabilities on safety without repeating the formal verification.

The rest of the paper is organised as follows. Section 2 presents the state of the art at the intersection of safety and security. We introduce STPA-Sec and present our use case in Section 3. In Section 4 we show an overview of Tiresias . In Section 5 we build a set of relations between the behavioural and the safety model of the CPS, and we introduce the threat model in Section 6. In Section 7 we formalise the process of discovery of threat scenarios, while second and third steps of Tiresias are formalised in Section 8. The methodology is applied to the use cases in Section 9. Finally, in Sections 10 and 11 we discuss the obtained results and present our conclusions.

2. Each stage of our methodology was implemented as a separate program. The analyst is expected to run start each separately.

2 Related Work

Whilst, to the best of our knowledge, our approach is novel, several studies, in literature, have proposed steps in the same direction. SAHARA combines STRIDE threat modelling with the safety analysis methodology HARA, to perform integrated safety and security analysis [13]. VERDICT is an annex for Architecture Analysis & Design Language (AADL) that facilitates integrated safety and security analysis on AADL models [14]; it consists of two major components: Model Based Architecture Analysis and Synthesis (MBAAS) and Cyber Resiliency Verification (CRV). MBAAS uses AGREE [15] to infer the propagation of the effects of threats in the CPS model. To do that, MBAAS requires the system designer to specify a set of propagation rules, hard-coded inside each component of the AADL model. In contrast, our approach relies directly on the behavioural model of system components to analyse the impact of the attack and does not require human input to specify threats' propagation. On the other hand, CRV employs model checking to verify the reachability of threat scenarios directly from the behavioural model of system components, given a limited set of threats. MBAAS uses a subset of CAPEC [16] as source for threats whereas CRV employs a non-standard fixed set of threats hard-coded in the model interface whereas, our methodology relies on STRIDE to systematically construct the attacker model. VERDICT returns high-level threats that can potentially impact safety, whereas we analyse the attack graph of the system architecture and also determine the attack paths leading to those threats. Longari et al. [17] use attack trees to perform a goal oriented analysis of automotive systems. They need to manually select the objectives for the analysis and cannot automatically discover complex attacks that rely on cascading effects as in our approach. [18] proposes a threat modelling methodology for CPS based on STRIDE. [19] present a methodology to identify attacks leading to the *shut down* of system operations, while [20] introduces a recursive approach for building attack graphs for IoT systems based on cyber and physical interactions. Barrère et al. [21] use an approach based on MaxSAT on AND/OR graphs to identify the "Most Likely Mission Critical Component Set of a CPS". A Boolean Driven Markov Processes (BDMP) approach is proposed to analyse interactions between safety and security and subsequently used to assess the impact of vulnerabilities on the safety risks in a control system for a pipeline in [22]. [23] proposes a quantitative analysis of the effects of securing communications between controller and physical process. More recently, [24] presents an interesting study on the need for standards for security and safety

co-analysis. Two surveys summarise methodologies to perform integrated and security analysis [25] and discuss their similarities and differences [26]. [27] relies on manually specified attack-fault-maintenance trees (AFMT) to determine trade-offs between system attributes including safety, security, and maintenance. In contrast, we use STRIDE as threat modelling methodology and the sequences of attack steps interleaved with component behaviour (for cascading effects) are automatically derived through symbolic model checking. International standards for safety critical industry are also mandating the need of integrated security and safety analysis across a variety of sectors from nuclear [28], [29] to aviation [30].

System Theoretic Process Analysis and Security. STPA-Sec [6], [31] extends STPA [32] to the security domain. Both STPA and STPA-Sec are widely accepted methods for integrated safety analysis and are applied to identify safety critical threat scenarios across a number of different sectors [33], including industrial control systems [34], micro-grids [35] and aerospace [36]. Friedberg et al. propose STPA-SafeSec [37] which builds a correspondence between the STAMP model and the architecture of the CPS. STPA-SafeSec manually investigates the effects on safety of high-level threats. [38] propose a methodology to analyse the impact of security flaws on a STAMP model at a functional level. [39] extend STPA with STRIDE. These approaches are executed by hand and require a significant amount of effort as well as deep expert knowledge. Furthermore, their output is a set of high level scenarios and recommendations. With Tiresias , we automatically discover complex sequences of high level threats that impact safety, without losing generality. Towards the end of the process we exploit the map with the deployed architecture to identify specific attack paths. In a previous work, we have performed the integrated safety and security analysis of a smart grid testbed using a manual approach to combine STPA-Sec with attack graph analysis [40]. Finally, Nourian et al. show the effectiveness of STAMP for safety and security analysis to analyse the Stuxnet case [41].

System Theoretic Process Analysis and Model Checking. Although partially automated in [42], STPA remains a manual process driven by expert knowledge. Several studies [43], [44] and [45] map a STAMP model of the CPS to the behavioural model of its implementation. Such mapping enables to formulate hazardous control actions as specification in Linear Temporal Logic (LTL) [46]. These specifications are formally verified to ensure that safety requirements are met by the current implementation. Abdulkhaleq et al. [44] use this strategy to generate tests for safety critical software. Similarly, system safety requirements (SSRs) are generated in [47]. Finally, Zhao et al. in [48] construct specifications for an avionic system and verify them with the UPPAAL model checker [49].

Adversarial Model Checking. The use of formal verification to discover vulnerabilities in complex systems and protocols dates back, at least to, 1998 when Mitchell at al. used model checking in [50] to find vulnerable attack paths in SSL. More recently, Hussain et al. in [51] combined model checking and cryptographic protocol verification to identify attack paths in LTE. CVAnalyzer [52] relies on both traditional and probabilistic model checking to find vulnerabilities and assess risk in inter-vehicle communication protocols. They also consider the presence of an attacker in the loop. [53] proposes to derive the Finite State Machine (FSM) of a firewall through fuzzy requests and subsequently use it to find attack paths through model checking. Researchers at AWS applied model checking in TLA+ to find complex sequences of actions leading to unexpected behaviour [54]. However, they do not consider an adversarial environment. Alshalalfah et al. [55] UPPAAL to verify safety properties of an insulin pump against a custom designed attacker capable of executing replay attacks. Poorhadi et al. developed a framework based on Event-B to analyse the impact of cyber attacks on the safety of a railway signalling system [56].

3 PRELIMINARIES

Safety is generally defined as *freedom from unacceptable risk* [57], where the risk *is a combination of likelihood of harm and severity of that harm* [58] [59]. In this work, we rely on a simpler definition of safety, namely the absence of accidents and their consequent losses. The objective of our methodology is to analyse how security threats such as the spoofing of a component, the tampering, or the unauthorised access to a piece of information, etc. can contribute to the causation of accidents. The definitions of *losses*, *accidents* and *hazards* that we use in this paper are those presented in [32].

Tiresias relies upon System Theoretic Process Analysis Security (STPA-Sec) to analyse the safety of the CPS STPA-Sec is built on System Theoretic Process Analysis (STPA) and they are both organised in the same $four$ steps [60]. The fundamental difference between STPA and STPA-Sec lies in the identification of possible causes for accidents, in the last step. In fact, while STPA looks at *faults* and *miscommunication* as possible root causes for accidents, STPA-Sec finds them in *threats* [6]. An overview of STPA-Sec is shown in Figure 2. The steps involved are represented in white rectangles

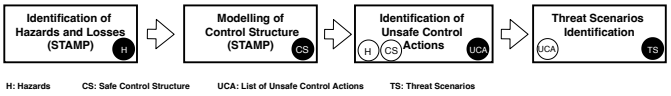

Fig. 2. STPA-Sec Overview

whereas the input processed and the output produced at each step are shown in white and black circles, respectively. The acronyms used in the diagram are expanded in the key at the bottom of the Figure. Steps (1) and (2) of STPA/STPA-Sec provide guidance towards the definition of the STAMP model for the system [32]. In fact, (1) consists of the definition of system level hazards, losses, and safety requirements, followed by the modelling of the control structure (2). The control structure or Safe Control Structure (SCS) is a tuple (C, D, K) where C is the set of components, D the set of information flows (*control actions* and *feedback*), and K the set of *functional channels* [61] . For each controller in the SCS, STAMP requires to specify its *process model* and *control algorithm*.The former retains a representation of the controlled physical process while the control algorithm defines input/output relations. Step 3 defines the *Unsafe Control Actions (UCA)*, those control actions (and feedback) that can cause hazards when applied in a specific *context*. The *context* of a UCA describes the combination of values of variables in the process model of the issuing controller for which the

application (or missed application) of the control action is unsafe [42], [43]. More formally, a UCA can be defined as a tuple $(ca, ctx, r, type)$ where $ca \in D$ is the control action, ctx the *context*, r is the value for which the hazard subsists and $type$ the type of the UCA [44]. A control action can be *unsafe* if it is: *provided* when not required or *not provided* when required. In addition, a UCA in the case *provided* can be: simply *provided* but also *provided too early, too late or in the wrong order*, or *provided with wrong duration* [32]. The *Context Table* of a controller has been introduced in [42] as a tool to systematically enumerate UCAs. Finally, step (4) consists in the identification, through the whole Safe Control Structure, of possible scenarios of applications of UCAs. The fundamental difference between STPA and STPA-Sec lies in the domain of the causes identified at this stage (faults for STPA, security threats for STPA-Sec). Traditionally, the $four$ steps of STPA are all performed by hand, mainly relying on expert knowledge. Although there have been proposals to automate at least part of the analysis using techniques based on formal verification and model-checking, they do not consider security threats [42], [43]. In Section 3.1, we briefly introduce our use case and conventionally apply the first $three$ steps of traditional STPA/STPA-Sec as they are proposed in [60].

3.1 Railway management infrastructure

A Communication Based Train Control (CBTC) system is a traffic management system typical of wide railway infrastructures such as the European Railway Traffic Management System (ERTMS) [62]. In our scenario N trains share access to the same railway infrastructure which is divided into M *zones* that trains access in a mutually exclusive fashion. The zone controller (ZC) coordinates access through the zones by allowing or declining requests from trains, depending on their current location and signalling context. When a train enters a new zone, it sends a *clear* message to the ZC, to indicate that it has left the previous zone. Trains are fitted with an onboard Train Controller (TC) that constantly receives their current position from an on-board location unit (LU) and speed from the Speed Controller (SC). TC sends 'access' and 'clear' requests to ZC and evaluates the replies. In our scenario, the location unit combines data from a GNSS module with radio beacons to improve the accuracy of the estimated position [63], in a configuration similar to [64]. A fallback strategy allows the LU to solely rely on GNSS if the radio signal is unavailable. The TC also receives a reference speed limit from the ZC for the current zone and sets the target speed for SC. Train motion is controlled by the speed controller, which computes the acceleration profile and commands brake and acceleration unit (AU) over a local bus. Positive values of the acceleration are interpreted as acceleration and negative ones as a brake [61]. A safety mechanism halts the trains if the communication between the ZC and one of the trains fails. We start with the definition of accidents and find the related hazards. For brevity, we only consider *collision* (A_1) and *derailment* (A_2) as accidents, as these lead to the heaviest losses and could endanger human lives [65]. For completeness, we also consider *traffic disruption* (A_3) as an example of a less important accident. In fact, although (A_3) is a non life-threatening accident, an adversary can still aim to provoke it, to cause financial loss. We identify a first set of three hazards: *Train enters a not cleared zone*, *Over-speed* and *Train Halts*. We refine these hazard into the ten in Table 1.

TABLE 1
Table of Hazards for the CBTC system.

#	Hazard	Accidents
H_1	Controller allows the train to enter a *not clear* zone	A_1, A_3
H_2	Controller does not allow the train to enter a *clear* zone	A_3
H_3	State of zone controller is not consistent with the trains' position	A_1, A_2, A_3
H_4	Train halts	A_3
H_5	Zone Controller marks as *not clear* a clear zone	A_3
H_6	Zone Controller marks as *clear* an engaged zone	A_1
H_7	Train Controller commands to enter a *not clear* zone	A_1, A_3
H_8	Train Controller does not command to enter a *clear* zone	A_3
H_9	Zone Controller commands a higher speed limit than expected for a specific zone	A_2
H_{10}	Train Controller commands sets speed above limits dictated by the zone controller	A_2

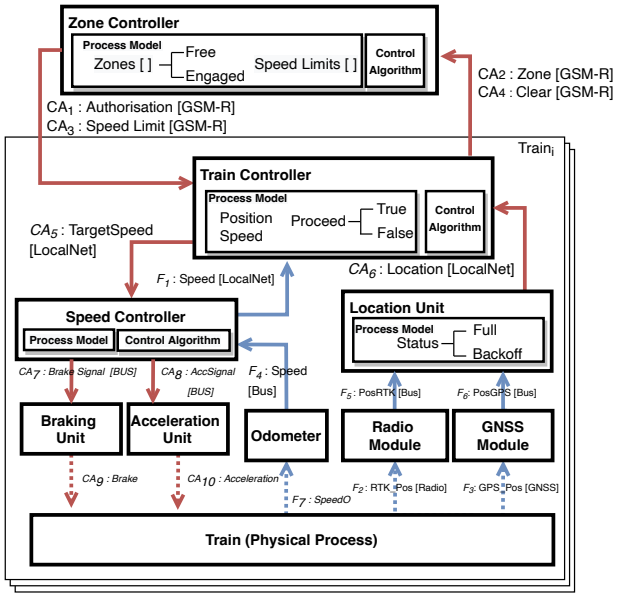

Fig. 3. Safe Control Structure of CBTC system.

The safe control structure (SCS) of the system is shown in Figure 3. Control actions (red) and feedback signals (blue) are exchanged over digital (dashed) and physical (dotted) channels. The system comprises *one* zone controller, and *three* controllers for each train: a *train controller*, a *location unit* and an *acceleration unit*. ZC is responsible for issuing control actions CA_1 (Authorisation) and CA_3 (Speed Limit) in a timely manner over a GSM-R channel. The process model of the ZC has a view of the zones, and specifies which zones are free and which are engaged along with their speed limits. The status of a zone is a binary value indicating whether it is *free* or *engaged*, while the speed limit is an integer value. CA_1 carries the value *allow* or *deny* while CA_3 is a positive integer. Both control actions depend on the internal status of the controller (*Zones* and *Speed Limits* arrays, respectively) and the position of the

recipient. The Train Controller holds, in its process model, the current position (zone) and speed as well a binary value, which indicates whether the train has been cleared to proceed or not. TC issues CA_2 and CA_4 to respectively request access to and clear a specific zone. CA_2 and CA_4 carry the id of the issuer train ($tid \in N$) along with the id of the zone ($i \in M$) and also depend on the internal state of the controller for the current position. TC also sets the target speed CA_5 on the speed controller and receives the current speed from the speed controller (F_1). The Location Unit is responsible for sending the current location CA_6 to TC. The value of CA_6 depends on the operating mode (*full, fallback*), which is saved in LU's process model as a binary variable. The LU receives its current position from the sensors Radio (F_5) and GNSS (F_6). The LU relies solely on GNSS (F_6) when it is operating in *fallback* mode. The Speed Controller receives the target speed from the TC and, given the current speed (F_4) computes an acceleration profile. The latter is used to command the brake (CA_7) and acceleration (CA_8) units. The odometer, radio sensor and GNSS module respectively receive feedback signals (F_7), (F_2) and (F_3).

We apply the first step of STPA to the CBTC system (Figure 3) and derive the context table (Table 2). For brevity, we only analyse the unsafe application of CA_1, CA_2, CA_3, CA_4 and CA_5. These actions are all in the discrete domain; therefore, UCAs of the type *applied for too long* or *applied for too short* are not included in the Table 2. UCAs of the types *provided for too short* and *provided for too long* are not included as they only apply to continuous control actions. The following UCAs facilitate the zone management and the i-th train. We also indicate the hazard they lead to. A list of UCAs derived from table 2 for CA_1, CA_2, CA_4 and CA_5 is shown below

UCA_1: CA_1 *not provided* when $zone[i]$ is *engaged* $\longrightarrow$ (H_4)
UCA_2: CA_1 *provided* with *allow* when $zone[i]$ is *engaged* $\longrightarrow$ ($H1$)
UCA_3: CA_1 *provded late* when $zone[i]$ is *engaged* $\longrightarrow$ (H_4)
UCA_4: CA_1 *not provided* when $zone[i]$ is *free* $\longrightarrow$ (H_4).
UCA_5: CA_1 *provided* with *deny* when $zone[i]$ is *free* $\longrightarrow$ (H_2).
UCA_6: CA_1 *provided* too late when $zone[i]$ is *free* $\longrightarrow$ (H_4).
UCA_7: CA_2 *not provided* $\longrightarrow$ (H_4).
UCA_8: CA_2 *provided* with wrong parameter $\longrightarrow$ (H_3, H_5, H_7).
UCA_9: CA_2 *provided* too late $\longrightarrow$ (H_4).
UCA_{10}: CA_3 *not provided* $\longrightarrow$ (H_4).
UCA_{11}: CA_3 *provided* with parameter r, with $r > v$ and v is the allowed speed limit $\longrightarrow$ (H_9).
UCA_{12}: CA_3 *provided* early ($r > v$) $\longrightarrow$ (H_9).
UCA_{13}: CA_4 *provided late* $\longrightarrow$ (H_4).
UCA_{14}: CA_4 *not provided* $\longrightarrow$ (H_3, H_5).
UCA_{15}: CA_4 *provided* with wrong position $\longrightarrow$ ($H3, H_5, H_6$).
UCA_{16}: CA_4 *provided* too late $\longrightarrow$ (H_6).
UCA_{17}: CA_5 *provided* when controlled replied *deny* (H_7)
UCA_{18}: CA_5 *not provided* when *Proceed* is true (H_8)

4 Methodology Overview

A detailed overview of Tiresias is shown in Figure 4. Preparatory stages (P.1) to (P.6) precede its application and are highlighted with a lighter background in the picture. The input consumed at each stage and the produced output are indicated in white and black circles, respectively. During the preparation, we perform the conventional application of the first three steps of STPA/STPA-Sec. In particular, ((P.1)) we identify Losses, Accidents and Hazards, ((P.2)) we model the Safe Control Structure, and ((P.3)) we determine the Unsafe Control Actions (Section 3). Model checking enables us to

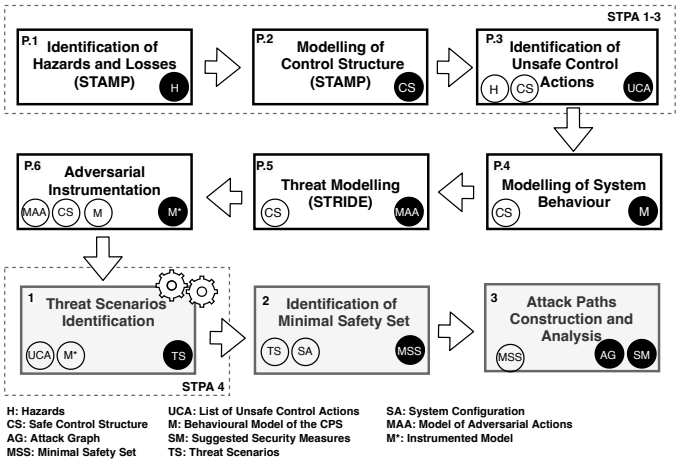

Fig. 4. Overview of the proposed methodology.

automatise the fourth step of STPA-Sec and automatically identify threat scenarios in the first stage of Tiresias. For it to be applicable we need to model ((P.4)) the safe behavioural model (SBM) of the CPS and build a map to its safe control structure (SCS) [66]. The output of this step is a behavioural model of the CPS that maps to its SCS. In this step, we assume that the high-level behaviour of system components is already known from system design and provided as labelled transition system (LTS). We propose and apply a set of rules to map the a subset of the states of the LTS to the SCS. Such map allows us to formally verify UCAs on the behavioural model of the system. The implementation of the process is outlined in Section 5. Then,in ((P.5)) we apply STRIDE to identify potential threats to elements of the SCS. STRIDE allows us to identify the *adversarial actions (AAs)*, where each action represents the use of a STRIDE threat against an element of the SCS. We aggregate the AAs into a model of adversarial actions (MAA). As final stage of the preparation, we construct ((P.6)) the parallel composition between the SBM and the MAA (Section 6). As STPA and STPA-Sec are accepted for the creation of assurance cases for safety critical systems we assume that stages (P.1), (P.2), (P.3), (P.5) and (P.6), are carried out already in the design life-cycle of Cyber Physical Systems [30]. At the same time, we assume that the high-level behaviour of system components is known at design time; therefore it is possible to design the SBM ((P.4)). Although the preparation phase is essential to apply our methodology, we leave the development of tools that support it for further work.

TABLE 2
Context Table of Zone Controller (ZC) and Train Controller (TC). r is the value for which CA is unsafe, v is the speed limit.

Controller	CA	$Zone[i]$	Limit	Position	Previous	Proceed	Not Provided	Provided	PE	PL
ZC	CA_1	Engaged	-	-	-	-	H_4	$H_1\ [r = allow]$	-	H_4
ZC	CA_1	Free	-	-	-	-	H_4	$H_2\ [r = deny]$	-	H_4
ZC	CA_2	-	-	$i \in M$	-	-	H_4	$H_3, H_5, H_7\ [r \neq i]$	-	H_4
ZC	CA_3	-	v	-	-	-	H_4	$H_9\ [r > v]$	H_9	H_4
ZC	CA_4	-	-	-	$i \in M$	-	H_3, H_5	$H_3, H_5, H_6\ [r \neq i]$	-	H_6
TC	CA_5	-	-	-	-	False	-	$H_7\ [r > 0]$	-	-
TC	CA_5	-	-	-	-	True	H_8	-	-	-

In the first stage of our methodology, (❶) we combine STPA-Sec with STRIDE and model-checking to automatically generate threat scenarios, sequences of malicious actions that, together with the behaviour of the system, can result in catastrophic consequences. The identification of threat scenarios corresponds with the fourth step of STPA-Sec where this operation is performed manually. In contrast, we rely on model checking to formally verify the reachability of UCAs in presence of a omniscient attacker (Section 7). Then, (❷) we define a Boolean function that evaluates true for those combination of privileges that enable the pursuing of catastrophic attacks[3]. We use an SMT solver to enumerate the minimum set of privileges that should be defended to preserve the safety. When more than one set of privileges is discovered, defending any of these sets guarantees overall safety. Thus (❸), we leverage the integration with the architecture of the CPS to produce an attack graph that encompasses feasible attack paths leading to the privileges in the found set (Section 8). To do so, we assume that the network topology of the CPS is known and that a vulnerability assessment has been already carried out. It is obviously not possible to address unknown vulnerabilities (zero-days), although further analysis on this point is possible, which we leave for further work. The map between STRIDE threats and the privileges required to effect the threats needs to be carried out manually, as part of the preparation stage.Finally, we perform a reachability analysis on these paths and identify the weaknesses and vulnerabilities that should be addressed first to prevent accidents. Our approach is able to uncover a wide range of mitigations strategies, within both the OT and the Enterprise network. This is a considerable advance on other approaches that only consider measures applicable on the OT network (Section 10).

Tool-chain. Each stage of our methodology was implemented as a separate program. In Section 7 we outline our implementation of the Threat Enumeration Process (❶). The identification of the Minimal Safety Set (❷) has been implemented as a Python application using the Microsoft Z3 libraries (Section 8) [67]. Finally, ((❸) we use MulVal to generate a system-wide attack graph and developed our own Java routines to perform the analysis presented in Section 3.1.

5 Safety Verification

In conventional STPA-Sec, the identification of threat scenarios is typically performed manually relying on expert knowledge. This is an exceptionally complex operation that requires a deep understanding of the functioning of a multitude CPS components, and how their output can vary, depending on their current state and variations in their input. The manual nature of this process constitutes a bottleneck in the application of STPA-Sec. To overcome this problem in STPA, it is possible to model the behaviour of the CPS in the safe behavioural model (SBM) and use model checking to identify hazardous scenarios [43], [66]. In this Section, we propose our implementation of the safe behavioural model (SBM) to automatise the identification of threat scenarios in STPA-Sec. This fundamentally differs from the implementation proposed in [66] in the case of STPA, as we allow for the integration of the SBM with the behavioural model of the adversary. Futhermore, we employ an approach based on Time-based Computational Tree Logic (TCTL), instead of linear logic, to verify the reachability of unsafe states. While the relative benefits of branching logic and linear logic have been debated in the literature [68], our choice of using TCTL is linked to the tool used for verification (Section 7).

Safe Behavioural Model (Structure). We define the safe behavioural model (SBM) of the CPS as a network of timed automata (NTA). Semantically, a NTA is a labelled transition system $\mathcal{N} = (T, Ch, B)$ where T is the set of timed automata (TAs), B a set of data buffers and Ch a set of synchronisation channels. Automata in T synchronise with each other over synchronisation channels in Ch and exchange data through data-buffers in B. The semantics of the NTA and of the Timed Automata used here are the same as those in [69] and [70]. A Timed Automaton (TA) is a tuple $(\Sigma, L, l_0, \mathcal{V}, \mathcal{C}, E, I)$ with Σ the input alphabet, L the set of locations and $\mathcal{V}$ the set of local variables. l_0 marks the initial location while $\mathcal{C}$, E and I are the sets of clock variables, edges and invariants respectively. An edge $(l, l', \sigma, \lambda, \delta, o, s)$ marks the transition $(l \rightarrow l')$, from location l to location l'. The transition is fired on the word σ, λ represents the updates engaged when the transition is fired and δ represents the guard conditions. Finally, o is the output and s points to the synchronisation channel. The latter is set only if the transition $(l \rightarrow l')$ is synchronised with one or more different transitions.

The structure of the SBM largely depends on the behaviour of the CPS components, which we assume is known at design time. At the same time, a subset SBM needs to be linked to the safe control structure (SCS) to enable the formal verification of UCAs. Let (C, F, K) be the SCS of a CPS with N_p physical processes, N_c controllers, N_k functional channels and N_f control actions and feedback signals. Let $\mathcal{N} = (T, Ch, B)$ be the related SBM. $\mathcal{N}$ has

3. An attack is catastrophic if it has the potential to provoke a loss.

$N_{TA} = N_p + N_c + N_M + N_k$ Timed Automata synchronised over $N_{Ch} = 2 * N_f$ synchronisation channels with $|B| = N_f$ buffer variables. N_M is the number of observer automata. These are used to monitor the unsafe applications of control actions of the type *not applied*, *applied with wrong duration* and *applied with wrong timing*. To this extent, the SBM contains as many observer automata as there are control actions whose unsafe applications we need to monitor. Modelling functional channels through dedicated automata enables to split the send and receive phases of the communication between two automata in two subsequent transitions. By doing so we can verify safety properties related to attacks on the integrity and the availability of the information flow, such as altered, missing or delayed communications. In Section 3, we defined an UCA as a tuple $(ca, ctx, r, type)$. To verify unsafe control actions, we begin by building a map of control actions and contexts between the SCS and the SBM. Let $T_1, T_2, ..., T_{N_{TA}}$ be the timed automata in T, if T_i encodes the behaviour of the component C_i, we build a one-to-one mapping between the process model variables of C_i and the subset $\mathcal{V}_i' \subseteq \mathcal{V}_i$ of T_i. Such correspondence between process model variables and automata local variables allows us to unambiguously translate the *context* of a UCA from one model to the other. Additionally, for each control action and feedback, we define a *provided* location in the automaton that encodes the behaviour of the issuer component. Checking that the automaton transits through the *provided* location enables us to formally verify a UCA. Furthermore, we introduce *transactions* to verify UCAs defined on the timed, ordered execution of multiple events. Let $a!$ and $b?$ be two synchronisation events on two different channels, we define a transaction as any pair $< a!, b? >$. We employ a special class of observer automata to measure the time elapsed between the two events that define the transaction and flag any violation of time-critical safety constraints. $\mathcal{N}$ counts N_M observer automata, with N_M also being the number of real-time sensitive *transactions* in the system. For each physical process in the safe control structure, the SBM features one automaton that describes its evolution. This is important as safety attributes can also be defined on the states of the physical process (e.g. the process never enters the hazardous state $\bar{s}$).

Verification. We express STPA UCAs in Timed Computational Tree Logic (TCTL) [69] and verify them against the SBM. The presence of clocks in TCTL allows us to explicitly handle and verify time constrained hazards. Safety specifications are expressed in TCTL as reachability statements of the form: $E <> \phi_{UCA}$, i.e., *there is at least one computation where ϕ_{UCA} is true*. UCAs of the type *provided when not required* or *provided with wrong parameters* are expressed as:

$$\phi_{UCA} := ctx \wedge T_i.l_{CA}\,[arg]$$

where ctx denotes the context, $T_i.l_{CA}$ denotes the location of the controller automaton that issues the control action and arg the (optional) parameters. The verification of UCAs of the type *not provided* (NP), *provided with wrong timing* and *provided with the wrong duration* is carried out through *observers*, e.g., the observer automaton of a control action enters a *wait* state when a particular context is true. From there we can tell if the UCA is not provided within an amount of time $\bar{t}$ or whether it lasts for too long or too short.

$$\phi_{UCA} := (CAMonitor).NP \wedge (CAMonitor).clock \leq \bar{t}$$

where $T_i.clock$ denotes the observer's clock. Similarly, UCAs of the type *stopped too early (too late)* are formulated as:

$$\phi_{UCA} := (CAMonitor).Stop \wedge (CAMonitor).clock \leq (\geq)\bar{t}$$

Finally, we combine observers with logical clocks [71] to identify whether a control action is provided too early or too late. Given a reachability statement, the model checker verifies whether the hazardous location is reachable under the current context and returns a *witness trace* π of the sequence of events leading to the hazard.

5.1 Modelling the behaviour of the CBTC

We have designed, implemented and tested the Safe Behavioural Model of the CBTC subsystem responsible for the zone management. This includes aspects related to the exchange of the train position between the zone controller and multiple trains as well as coordination, permissions and clearances (Section 3.1). Details on the implementation are outlined in the Additional Materials (Appendix A). For brevity, we only focus on zone access control and omit the aspects related to the control of the trains' speeds. The Safe Behavioural Model is represented by a NTA that includes $5N + 1$ timed automata, with N the number of the trains. In fact, $N_p = 1$ status of the mutual exclusive access zone, $N_c = 2N + 1$ with two controllers for each train (TC and LU) and ZC, and $N_k = 1$ GSM-R channel for communications between ZC and trains. For each train we have $N_M = 2$ observers in place to monitor unsafe control actions of the type *Not Provided* and *Provided too Late* on CA_1, CA_2 and CA_4. *Three* pairs of synchronisation channels and *three* buffer variables allow to orchestrate the information exchange.

Using the UCAs in the context table (Table 2) of the CBTC (shown in Figure 3), and the safe behavioural model of its implementation we define the following TCTL specifications:

UCA_1, UCA_4 - $\phi_1 := E <> CA1Monitor.NotProvided$ *and* $clk < \bar{t}$

Explanation: The ZC must always reply to CA_2. A violation occurs if it is not provided within $\bar{t}$ from CA_2.

UCA_2 - $\phi_2 := E <> zones[i] == j$ and $t\_id \neq j$ $loc[j] == loc[zones[i-1]]$ and $ZC.ReplyProvided$ and $reply == allow$

Explanation: Find an example of trace where, if i-th zone is engaged by train j the ZC replies *allow* to an access request for the same zone incoming from a train with an id that is different from the one in the zone i.

(UCA_3, UCA_6) - $\phi_3 := E <> CA1Monitor.Timeout$

Explanation: The automaton that monitors the time delay between CA_2 and CA_1 reaches timeout.

UCA_5 - $\phi_4 := E <> zone_i == 0$ and $t\_id = k$ and $ZC.Busy$

Explanation: Finds examples where a controller infers that a zone is "busy" when the zone is not occupied by any train.

UCA_7 - $\phi_5 := E <> CA2Monitor.NotProvided$ *and* $clk < \bar{t}$

Explanation: Control action CA2 is not provided within $\bar{t}$ from a change of location.

UCA_8 - $\phi_6 := E <> (trains.Approach$ *and* $area\_code! = location)$

Explanation: The train communicates a location that is different from its actual current location to the zone controller.

$UCA_{15} - \phi_7 := E <> (trains.Clear$ *and* $area\_code! = previous\_location)$

Explanation: The train asks the controller to clear an area different from that it has just left.

$UCA_{17} - \phi_8 := E <> (zone_i == j$ *and* $trains(k).Enter)$

Explanation: Train Controller commands to proceed through a zone already occupied by another train.

These safety properties should be satisfied by a correct SBM of the CPS in the absence of an adversary. Indeed, we have verified that our model checker does not find any traces that violate the above TCTL specifications on the initial model. In the next sections we will investigate what happens in adversarial conditions.

6 Threat Model

We employ STRIDE to identify potential threats to elements of the SCS. We find that STRIDE provides a more structured approach to the analysis of potential threats compared to traditional STPA-Sec. Moreover, STRIDE is widely established, and has already been used with STAMP [18]. STRIDE allows us to identify the *adversarial actions (AAs)*, where each action represents the use of a STRIDE threat against an element of the SCS. *By aggregating the adversarial actions into an aggregated model (MAA) and formally verifying the specifications derived from STPA against the composition of the MAA with the CPS behavioural model, we identify the threat scenarios as sequences of AAs that, combined with system behaviour, lead to hazards (Section 6.1).* The SCS is a functional model and does not include deployment details. Thus, we need to ground the analysis in the specific deployment context to map the output from the STRIDE analysis to the specific exploitation of vulnerabilities on system components. To address this gap, we build the *Threat-Accident* model, which relates the adversarial actions (AA) to the exploitation of specific vulnerabilities on system components by identifying the privileges required to perform an AA (Section 6.2). Finally, we relate these privileges to an attack graph representation (Section 6.3) to reason about the attack paths [12], [72].

6.1 Model of adversarial actions

We employ STRIDE on the STAMP model (Step P.5, Figure 4), to build the model of Adversarial Actions (MAA). The MAA is a synthetic aggregated model of the possible behaviour of an adversary. This synthetic model is an over approximation of an adversary's behaviour, i.e., it includes threat sequences that may not be feasible, but serves our purpose well. In particular, we use STRIDE-per-interaction [73] on control actions, feedback and process models to build the MAA in a semi-automated, systematic, staged process shown in Figure 5. From the systematic application of STRIDE we derive an *attack library* (AL) i.e., a list of adversarial actions an attacker can apply against the SCS. Formally, the attack library is an alphabet of adversarial actions $AL^* = \{a_1, a_2, \ldots, a_K\}$ of size K.

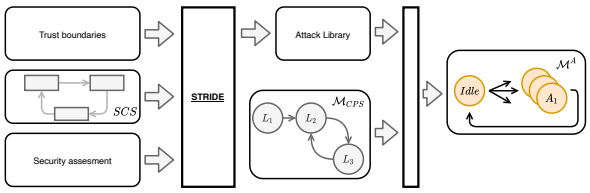

Fig. 5. Derivation of the model of adversarial actions.

Definition 1 (Adversarial action). *An adversarial action a is the specialisation of a STRIDE threat to an element of the SCS. It is defined as a tuple* $(type, obj, values, privileges)$.

Where $type$ is type of threat (e.g. spoofing, tampering, etc.), obj is the target object in the safe control structure, $values$ are the values of the action's parameters (e.g., values that are injected or spoofed), and $privileges$ is the set of privileges required to perform the action in the system. $values$ can be null (e.g. Denial of Service), a value or a range of values and is determined during the safety analysis. Because we are applying model checking, we are considering discrete (or discretised values) with at least one adversarial action defined for each value range. In general, determining values for data spoofing in conjunction with a model checking approach remains a topic for further research. To our knowledge no related studies have attempted this. Privileges are obtained through the exploitation of vulnerabilities (see Section 6.3). The map between STRIDE threats and required privileges needs to be carried out manually, as part of the preparation stage (P.5). The output of P.5 is the the model of adversarial actions (MAA). The process is shown in (Figure 5). The MAA is a Timed Automaton where the location set consists of the actions in the Attack Library together with a set of buffer locations for the communication (synchronisation) between the MAA and the behavioural model (e.g., to represent injection or tampering with a message). The MAA is constructed as follows:

- Each adversarial action in the AL is a state in the MAA.
- The MAA synchronises with the behavioural model when a control action (or feedback) is exchanged between two components. From there, the MAA non-deterministically chooses whether to perform the attack. For this reason, the SBM is also an input in the process of generation of MAA Figure 5)
- The attacker chooses the next action non-deterministically and after each action, the attacker returns to the initial location.
- We assume that all adversarial actions are independent from each other. However, we allow dependencies between them to be set manually (e.g., when the outcome of an action depends on the success of the preceding

action). For example, a *tamper* action can only be reached following an *information disclosure*.

The resulting automaton models the adversary performing any of the adversarial actions identified in the STRIDE model at any point or following the dependencies included in the model. This represents a superset of the possible attacks. An example of an MAA for the CBTC use case is shown in Figure 9. In Step P.6 (Figure 4), we then integrate the MAA with the safe behavioural model following a process of adversarial instrumentation (Figure 6). The output of the instrumentation is $\hat{\mathcal{M}} = \mathcal{M}_{CPS}|\mathcal{M}_A$ where $\mathcal{M}_{CPS}$ is the safe behavioural model of the CPS and $\mathcal{M}_A$ the MAA. It is possible to generate more than one MAA and analyse them separately e.g., one for each subset of threats that analysts intend to consider. This allows for faster analysis on a reduced set of threats.

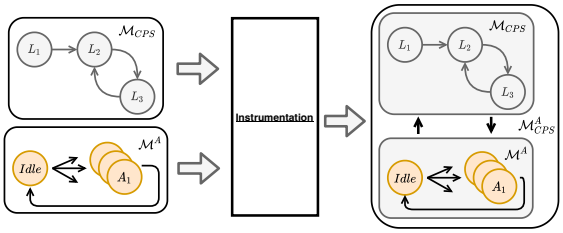

Fig. 6. Process of adversarial instrumentation

6.2 Threat-Accident Model

We aim to identify threat scenarios, i.e., those attacks where the attacker can trigger *Losses* to the CPS by causing *Accidents*. To achieve this, we build the *Threat-Accident* model to relate attack steps such as the exploitation of a vulnerability to hazards and their respective accidents. Figure 7 shows the relations we establish amongst the elements of the STAMP model, threats and low-level attack steps. This diagram, extends the concepts presented in [74] with elements from the STAMP domain. In STAMP *Losses* are defined as the consequences of *Accidents* which constitute the high level target of *Threat Actors*; *Threat Actors* use *Threats* to cause *Accidents*. From a functional perspective, *Threats* affect the *Control Actions*, *Feedback Signals* and the *Process Model*. The unsafe application of *Control Action*s leads the system into a hazardous state that can cause an *Accident*. From an architectural perspective, *Threats* are facilitated by the exploitation of vulnerabilities on system components and weaknesses (e.g. credentials, network access, etc.), which lead to the attacker obtaining the *Privileges* to *spoof/tamper/read/delay/-drop* the information flow of the CPS and thus cause the unsafe application of *Control Actions*. This application can be through direct actions e.g. through compromising *Control Actions*, *Feedback* or *Process Model*s or through cascading effects.

The representation shown in Figure 7 plays a key role in linking specific vulnerabilities to malicious actions, *Accidents* and *Losses*. Security engineers can define adversary goals in terms of the system level *Losses* and *Accidents*. Our approach provides them with the tools to identify the *Privileges* required to trigger a specific *Accident* and to find the respective attack paths. Security engineers and operators can then address the remediation of these paths.

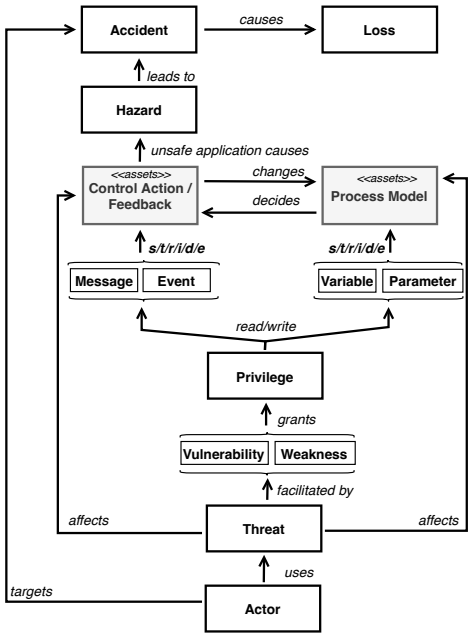

Fig. 7. Associations among threats to safety properties and vulnerabilities in a CPS deployment

6.3 Model of attacker progression

Formally verifying the safety properties against the instrumented behavioural model $\hat{\mathcal{M}}$, together with the references provided in the *threat-accident* model, produces a list of privileges that attacker needs to acquire to cause an accident. An attack graph based approach then allows us to explore the paths to the acquisition of those privileges from the moment a foothold is established in the network [75]. We use the logical attack graphs to model relations between vulnerabilities and privileges [76] [12]. In particular, we use MulVal to generate attack paths leading to threat scenarios. *MulVal* is a *logic-based network security analyser* [77] and uses uses *facts* and *rules* to derive the attacker's progression throughout the computer network. We provide, as input, a set of logic predicates (facts) which define architecture of the system (e.g., hosts, network topology, programs, privileges, etc.), the vulnerabilities affecting system components as well as the privileges that are objectives of the attack. To derive the progression of the attack, we use the set of rules already available in MulVal. These define the relations among the facts and are expressed through Horn clauses [78].

7 Discovery of Threat Scenarios

The discovery of threat scenarios is the first stage of Tiresias (Figure 4). It is a top-down staged process and uses, as input, the UCAs obtained from the execution of the third step of STPA. We have shown (Section 5) that, for each UCA, it is possible to define the safety properties as reachability properties in TCTL. We have then shown (Section 6) that we can use model checking to verify these safety properties against the the parallel composition of the SBM and the synthesised attacker model MAA. When a violation of the safety properties in the instrumented model is reachable, the model checker outputs a trace leading to that violation (witness trace). This trace contains the ordered

and timed sequence of adversarial actions that the threat actor needs to perform to trigger a hazard. A sequence of adversarial actions leading to a safety violation is a threat scenario. A traditional verification process would return a single trace that violates the safety properties, whereas we are interested in *all* the traces, i.e., all the ways in which the property can be violated. Therefore, we re-iterate the model checking process multiple times, each time excluding the traces already found. When no more traces (and threat scenarios) are found, the safety property cannot be violated by the attacker, assuming that the model is correct and complete.

7.1 Formalisation

Formally, let H be the hazard caused by the application of $UCAs$

$$H \leftarrow \{UCA\}$$

Let $\mathcal{M}_{CPS}$ be the safe behavioural model of the CPS and $\Phi_H = \{\phi_1, \phi_2, .., \phi_N\}$ the set of safety properties for $UCA_1, ..., UCA_N$, where $UCA_1, ..., UCA_N$ are the UCAs leading to H. Also, let $\mathcal{M}_A$ be the model of adversarial actions built with respect to a specific threat model, and $\hat{\mathcal{M}} = \mathcal{M}_{CPS} | \mathcal{M}_A$ be the parallel composition of the $\mathcal{M}_{CPS}$ and $\mathcal{M}_A$. We verify that $\hat{\mathcal{M}} \models \phi_i$, $\forall i \in \{1, .., N\}$ and call π_i the symbolic trace that satisfies ϕ_i. Therefore, a threat scenario (attack) A_i consists of the locations of $\mathcal{M}_A$ visited in π_i, and is defined as $A_i \coloneqq \{e_{i1}, e_{i2}, ..., e_{iK_i}\}$, where e_{ik} are the locations of $\mathcal{M}_A$ contained in π_i. This can be easily generalised to multiple hazards and UCAs per hazard. For example, given hazards $H_1, H2, ..., H_N$,

$$H_1 \leftarrow \{UCA_{11}, ..., UCA_{1M_1}\}$$
$$...$$
$$H_N \leftarrow \{UCA_{N1}, ..., UCA_{NM_N}\}$$

we define $K = \Pi_{i \in N} M_i$ sets of safety specifications (one for each hazard)

$$\Phi_1 = \{\phi_{11}, ..., \phi_{1M_1}\}$$
$$...$$
$$\Phi_N = \{\phi_{N1}, ..., \phi_{NM_N}\}$$

where ϕ_{ij} is the safety specification that verifies the reachability of UCA_{ij}. Thus for $\hat{\mathcal{M}} \models \phi_{ij}$ ($\phi_{ij} \in \Phi_i$), π_{ij} is the respective trace, where there exists an attack A_{ij} leading to UCA_{ij}, and therefore to H_i. An attacker aiming to cause H_i must therefore be able to carry out all the adversarial actions in A_{ij} for any i and j.

7.2 Attack enumeration

As mentioned earlier, each verification of each $\phi \in \Phi$ returns a single witness trace that satisfies ϕ. To obtain all the traces that violate ϕ we need to re-iterate the verification excluding the previous found traces until no more traces that satisfy $\hat{\mathcal{M}}$ are found. We have therefore developed an algorithm for this.

Notation. Let N be the number of timed automata in $\hat{\mathcal{M}}$, we introduce the notation $\phi(\hat{L}_1, \hat{L}_2, \ldots, \hat{L}_{\bar{N}})$ with $\bar{N} < N$ and $\hat{L}_i \subseteq L_i$ a subset of locations of the automaton T_i. Let $\hat{L} = \bigcup_{i=1}^{\bar{N}} \hat{L}_i$ and let s_i be a state of $\hat{\mathcal{M}}$. Formally, $\phi(\hat{L}_1, \hat{L}_2, ..., \hat{L}_N) \models \hat{\mathcal{M}}$ if there exists a trace π such that $(s_1, s_2, ..., s_M) \notin \pi$ and $l_j \notin s_i$, with $l_j \in \hat{L}\ \forall\ i$ and $\forall\ j$. In other words, $\phi(\hat{L}_1, \hat{L}_2, ..., \hat{L}_N) \models \hat{\mathcal{M}}$ is true if there exists trace π that satisfies ϕ for which the automaton T_i does not pass through any location $l_{ij} \in \hat{L}_i\ \forall j$ and $\forall\ i \in \bar{N}$. Looking for infinite traces would take an infinite time, so we introduce a time bound $\bar{t}$ for the search. This gives us $\phi(l_1, l_2, ..., l_P)_{\leq \bar{t}}$ with $P, \bar{t} \in \mathbb{N}$ where we search for a trace that does not go through $(l_1, l_2, ..., l_P)$.

Description. The algorithm to enumerate safety critical attacks against the specification ϕ of a CPS is fully detailed in Appendix B. In short, given the hazard H triggered by the hazardous control action UCA and being $\Phi_{UCA} = \{\phi\}$ the set of specifications that verifies the presence of H in $\hat{\mathcal{M}}$, the algorithm starts with the verification of the specification ϕ. If $\hat{\mathcal{M}} \models \phi$ with witness trace π, we call *threat scenario* $A(\pi)$ the sequence of locations of the MAA contained in π. After we find a *threat scenario*, a new specification $\phi(A(\pi))_{\leq \bar{t}}$ is generated to verify the existence of other attacks different from the one has been found. The process is repeated until the new generated specification does not satisfy $\hat{\mathcal{M}}$.

Completeness of Results When iterated over all ϕ_{ij}, the algorithm allows us to discover all the threat scenarios, i.e., sequences of adversarial control actions, leading to the violation of a specification ϕ. The enumeration process can lead the discovery of threat scenarios that contains redundant adversarial actions. To this extent, discovered scenarios need to be reduced before further processing. Let $A_p = (e_{p1}, \ldots, e_{pN})$ and $A_q = (e_{q1}, \ldots, e_{qM})$, A_p is redundant if and only if $A_p \subset A_q$.

Implementation. We have implemented the attack enumeration process (Step 1, Figure 4) in a JAVA tool featuring two main components: a query generator and a model checker service (wrUPPAL). Given a safety specification ϕ and a time limit for the verification $\bar{t}$, the pair $(\phi, \bar{t})$ is sent to the wrUPPAAL service which has been previously initialised with the instrumented model $\hat{\mathcal{M}}_{CPS}$. wrUPPAAL is a wrapper of UPPAAL [79] JAVA API with a cache for the queries and a watchdog. The latter is key to limit the verification in time. The results of the verification are sent back to a query generator that extracts the sequence of adversarial actions $A_1 = e_{11}, e_{12}, \ldots, e_{1K}$ from the witness trace and computes the next query $\phi(A_1)$ until no more traces can be found, i.e., no more attacks are possible given the current implementation and threat model. The query generator and the model checker service communicate through Java Message Server (JMS) over Apache ActiveMQ.

8 MSS & Path Analysis

In Section 7, we have shown that, given the hazards H, we can enumerate the threat scenarios $A_1, A_2, ..., A_N$ leading to H. Given a set of adversarial actions $AL = \{a_1, a_2, ..., a_K\}$ a threat scenario is an ordered sequence of length M_i of adversarial actions $A_i = (e_{i1}, \ldots, e_{iM_i})$, with $e_{ij} \in AL^*$. To cause harm, an adversary must execute the entire sequence

of adversarial actions in at least one scenario A_i. We manually map each adversarial action to one or more a set(s) of privileges required to perform it. To perform an action, the adversary must obtain all the privileges in at least one privilege sets. The relation between the hazard H and the privileges required to cause it is summarised in Figure 8.

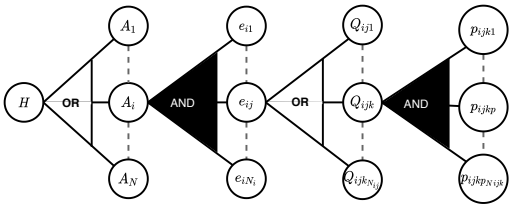

Fig. 8. Hazards in functions of privileges.

Formally, let $\{p_1, \ldots, p_M\}$ be the set of privileges that map to the adversarial actions in AL and $\{P_1, \ldots, P_M\}$ be a set of Boolean variables, such that P_i is true if the attacker has acquired the privilege p_i. We can write the hazard H as a function of the $P_1, \ldots, P_M$.

$$\mathcal{H}(P1, \ldots, P_M)\{0,1\}^M \rightarrow \{0,1\}$$

where $\mathcal{H}$ is a Boolean function that evaluates *true* those combinations of privileges that enable the attacker to cause the hazard. We denote with $\mathcal{S}$ the minimal safety set(s), i.e., the minimum set of privileges that falsifies $\mathcal{H}$. We compute $\mathcal{S}$ during the second step of Tiresias. If the defender can remediate the vulnerabilities leading to privileges $p_i \in \mathcal{S}$, P_i is always false and the attacker cannot cause H. Finally, in the third step of Tiresias, we build the attack graph of the CPS $\mathcal{A} = G(V, E)$, which contains all the attack paths starting from the system perimeter and leading to $\{p_1, \ldots, p_M\}$. Given two vertices $v \in V$ and $w \in \mathcal{S} \subseteq V$, we consider a vulnerability leading to v as *critical* if there exists a path from v to w. Finally, given the attack graph $\mathcal{A} = G(V, E)$ and the minimal safety set $\mathcal{S}$ we propose an enumeration algorithm (Appendix F) that enables us to find common vulnerabilities $v_i \in V_c \subseteq V$ in the logical attack graph. However, the complexity of the algorithm grows with the product of the number of edges in the attack graph and can only be applied by limiting the depth of the search (Appendix F). We leave the development of more tractable algorithms for further work.

Implementation. We have developed a program that uses Microsoft's Z3 solver to compute $\mathcal{S}$. The program expects in input a Boolean function that we defined from the threat scenarios identified during the previous step of Tiresias. After computing the MSS, we employ MulVal to generate the attack paths leading, from the system perimeter, to the privileges in $\mathcal{S}$.

9 Railway Traffic Control System

We apply our methodology to the CBTC use case introduced in Section 3.1. We start by applying STRIDE to the CBTC system (Figure 3) and determine the Adversarial Actions (AA) in Table 3. We consider Spoofing, Tampering, ID and DoS threats against control action CA_1. For each tampering and spoofing threat, we instantiate an adversarial action

TABLE 3
Resulting Attack Library

#	Object	Type	Value	Privileges
a_1	ZC : Zone[i]	Tamper	$[0-1]$	(p_{10}, p_{11}, p_{12})
a_2	LU : GNSS	Tamper	$[1-5]$	p_{13}
a_3	CA_1 : Auth	Spoof	$[0-1]$	P_2
a_4	CA_1 : Auth	DoS	-	$(p_{14}), (P_8)$
a_5	CA_1 : Auth	Disclosure	$[0-1]$	$(p_{10}), (p_{11}), (p_{12}), (p_2)$
a_6	CA_1 : Auth	Tamper	-	$(p_{10}, p_{11}, p_{12}), (p_2)$
a_7	CA_2 : Zone	Spoof	$[1-5]$	p_7
a_8	CA_2 : Zone	DoS	-	$(p_{14}), (p_8)$
a_9	CA_2 : Zone	Tamper	$[1-5]$	$(p_3, p_6, p_5), (p_7)$
a_{10}	CA_2 : Zone	Disclosure	-	$(p_3), (p_6), (p_5), (p_7)$
a_{11}	CA_4 : Clear	Spoof	$[1-5]$	(p_7)
a_{12}	CA_4 : Clear	Tamper	$[1-5]$	$(p_3, p_6, p_5), (p_7)$
a_{13}	F_3 : GNSS	Spoof	$[1-5]$	(p_9)
a_{14}	F_2 : RTK	Dos	-	$(p_4), (p_1)$
a_{15}	LU RTKOn	Tamper	$true$	p_{13}

for each significant[4] value to inject, in this case *allow* and *deny*. Similar threats affect CA_2 and CA_4. We assume that RTK messages are signed and cannot be tampered with or spoofed; but they can be disrupted (a_{14}). However, the communication with the GNSS can be spoofed. We introduce a dependency between information disclosure and tampering i.e., we assume that the communication protocol requires an attacker to read a message before modifying it. Three further tampering actions can be carried out against the process model variables held in zone controller (a_1), and the location unit (a_2, a_{15}). Each adversarial action in Table 3 is mapped to one or more sets of privileges. Privileges in $\{p_1, \ldots, p_{14}\}$ are obtained through the exploitation of critical vulnerabilities within the system architecture. These are vulnerabilities affecting components responsible for the exchange of control actions and feedback. The model of adversarial actions (MAA) is shown in Figure 9. Initial and buffer locations are shown in blue, while locations corresponding to adversarial actions are shown with a white background. Transitions towards the execution of adversarial actions are shown in red, whilst those in blue return to the initial location [5]. We proceed with the adversarial instrumentation and build the parallel composition $\hat{\mathcal{M}} = \mathcal{M}_{CPS} | \mathcal{M}_A$ of the SBM of the CBTC system and the MAA in Figure 9.

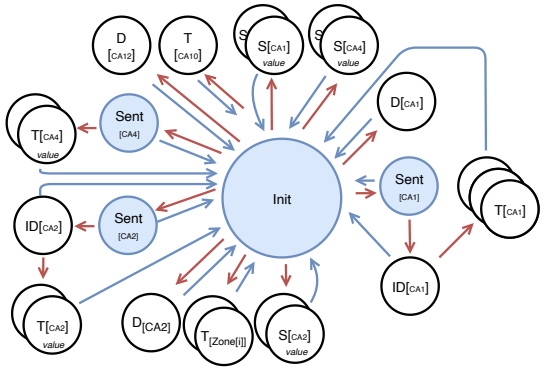

Fig. 9. MAA for the CBTC use case.

We use our tool-chain to verify safety specifications

4. STPA provides values r for which a UCA is hazardous.
5. The colour coding only aims to aid the visualisation. There is no difference in terms of the model-checking semantics

$\phi_1, .., \phi_8$ defined in Section 5. The outcome of the verification with a time bound set to $\bar{t} = 500s$ is shown in Table 4. For each specification ϕ we report: the number of threat scenarios found (TS), the number of threat scenarios after elimination of redundant attacks (RTS), the respective hazards and threat scenarios and the time taken to verify each query ϕ_i and those derived from it i.e., the time elapsed (TE) (Table 4).

TABLE 4
Attack traces violating safety specifications. Size of trace set (TS). Size of reduced trace set (RTS). Verification time (TE).

Hazard	Spec	TS	RTS	Threat Scenarios	TE
H_4	ϕ_1	1	1	$\{a_4\}$	$500.14s$
H_1	ϕ_2	7	7	$\{a_{10}, a_9\}$, $\{a_{14}, a_{13}\}$, $\{a_{15}, a_{13}\}$, $\{a_7\}$, $\{a_1\}$, $\{a_2\}$, $\{a_{11}\}$	$500.93s$
H_2, H_4	ϕ_3	1	1	$\{a_4\}$	$500.17s$
H_2	ϕ_4	2	2	$\{a_4\}$	$500.3s$
H_4	ϕ_5	1	1	$\{a_8\}$	300.15
H_3, H_5	ϕ_6	8	4	$\{a_{10}, a_9\}, \{a_4\}, \{a_{15}\}$	$962.3s$
H_5	ϕ_7	6	6	$\{a_{10}, a_9\}$, $\{a_5, a_6\}$, $\{a_{14}, a_{13}\}$, $\{a_{15}, a_{13}\}$, $\{a_{11}\}, \{a_2\}$	$537.3s$
H_7	ϕ_8	10	6	$\{a_5, a_6\}$, $\{a_{10}, a_9\}$, $\{a_{14}, a_{13}\}$, $\{a_{15}, a_{13}\}$ $\{a_2\}, \{a_3\}$	$864.11s$

Hazards H_1, H_3, H_5 and H_7 lead to a *collision* event. We enumerate attacks leading to these hazard through the verification of ϕ_2, ϕ_6, ϕ_7, ϕ_8 and find, respectively, 7, 4, 6 and 6 attacks. Thus, there are 11 distinct ways an attacker can cause a *collision* between two trains (some of the 23 traces enumerated in Table 4 appear more than once). We use Z3 [67] to find a minimal safety set $\mathcal{S}$ for which $\mathcal{H} = (H_1 \vee H_3 \vee H_5 \vee H_7)$ is false (i.e. a collision cannot be caused). As the set of privileges is contained, the solver can completely enumerate all the 9 minimal safety sets $\mathcal{S}_1, \ldots, \mathcal{S}_9$ in $< 1s$ (Appendix D). In complex examples, enumerating all sets may be computationally challenging. However, preventing one minimal safety set is sufficient to make the attack impossible. We use one of the minimal safety sets that satisfy $\neg\mathcal{H}$ to identify the minimum number of vulnerabilities the defender needs to remediate to prevent the hazard.

We use the architecture for the CBTC system shown in Figure 10, which features three layers: enterprise, ground control (RBC) and mobile. The enterprise layer consists of $three$ interconnected sub-networks including both workstations and servers. The ground network (RBC) ensures the infrastructure control operations and comprises three modular redundant (TMR) [80] implementations of the zone controller that orchestrates access to the shared infrastructure and a GSM receiver/transmitter module that also acts as arbitrator of the TMR. Each train has its own local network that hosts a TMR implementation of the train controller, a GSM module, the GNSS and radio modules for positioning and the HMI. We omit components not connected to the TPC/IP layer such as speed controller, odometer and actuation units as they communicate with the train controller over a dedicated bus (i.e. CAN), which we assume is out of reach for the adversary. We have instantiated this architecture with 54 known vulnerabilities including 36 vulnerabilities on components of the mobile and ground layers that lead

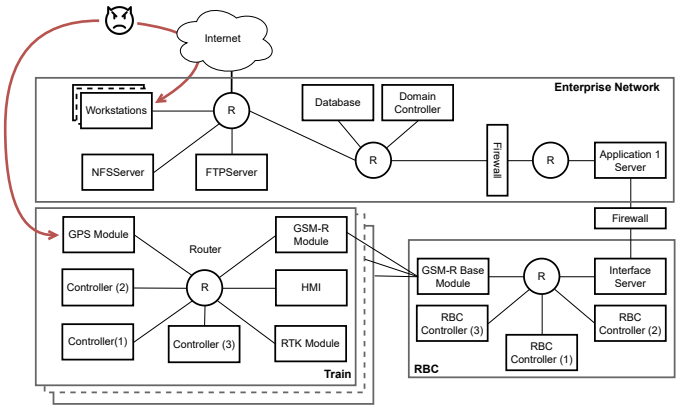

Fig. 10. Deployment of the CPS

to an attacker obtaining access to the privileges $\{p_1, ..., p_{14}\}$ (the full list of privileges is shown in Appendix E). We are interested in understanding which of the 54 vulnerabilities can lead to a safety critical attack.

We use MulVAL [81] to build the attack graph for the system assuming that an attacker can establish a TCP/IP connection with some of the work-stations in the enterprise network (e.g., as a result of phishing), a physical backdoor [82] and can carry out a GNSS spoofing attack. To carry out an attack that leads to a safety violation on the hazards, the attacker must obtain privileges $p_1, \ldots, p_{14}$, so they are used as targets for the attack graph generation. The corresponding attack graph produced by MulVAL has 333 nodes: 170 *and* nodes, 45 *or* nodes and 118 facts.

Identifying which vulnerabilities should be remediated requires an enumeration of the attack graphs and is challenging given its depth. To solve the problem, we select one of the minimal safety sets that satisfies $\mathcal{H} = false$, and use JGrapht [83] to evaluate, individually, the paths leading to the privileges in the set. For each path, we count the occurrences of AG nodes that are common among all paths. Our program identifies 119353 potential paths in the graphs in $< 5s$. However, some nodes shown below occur very frequently, in all or most of the paths.

```
netAccess(opgInterfaceServer,_,_)
execCode(opgInterfaceServer,root)
execCode(application_server_1,root)
```

These privileges were not part of the initial set $\{p_1, \ldots, p_{14}\}$ but are necessary to reach them. We select p^*=*execCode(opgInterfaceServer,root)* and apply our reverse enumeration algorithm (Appendix F) to find all the attack paths leading to p^*. This is feasible now since p^* is closer to perimeter than the privileges in the initial set $\{p_1, \ldots, p_{14}\}$. We identify the following vulnerabilities that are shared across the attack paths leading to p^*.

```
(opgInterfaceServer,vulnCVE2022_x,kernel,
    localExploit,privEscalation)
(opgInterfaceServer,vulnCVE2022_y,
    operatorLogin,remoteExploit,privEscalation
    )
(applicationServer1,vulnCVE2021_x,
    internalWebApp,remoteExploit,
    privEscalation)
(workstationHost1,vulnCVE2021_y,rdp_server,
    remoteExploit,privEscalation)
(workstationHost10,vulnCVE2021_z,rdp_server,
    remoteExploit,privEscalation)
```

p^* is common to 119352 of the 119353 paths found in the original attack graph, from the perimeter to $\{p_1, \ldots, p_{14}\}$. We fix the five vulnerabilities above and generate a new attack graph for the entire system. Since p^* was a common node, by removing it we remove almost all the paths leading to the minimal safety set. In fact, the new AG contains only one attack path, which is the one leading to p_9 which, together with any privilege in the set $\{p_1, p_4, p_{15}\}$ can lead to a *collision*. This corresponds to a GNSS spoofing attack against the system, which is not easy to remediate. A possible solution is to remediate vulnerabilities leading to $\{p_1, p_4, p_{15}\}$. These privileges in can prevent the attacker to disrupt the radio beacons (Table 3), making any eventual GNSS spoofing attempt not safety threatening anymore.

10 Discussion

We have presented a novel methodology that integrates safety and security analysis to identify those threats that can lead to accidents and hazards. The key points for the integration between the two analyses are: the generation of a combined MAA from the results of the STRIDE threat analysis, and the creation of a correspondence between the behavioural mdel of the system and its safe control structure. By combining the MAA with a behavioural model of the SCS and using formal analysis to verify the safety properties we identify the sequences of adversarial actions that lead to the violation of the safety properties. The existence of a map between STRIDE threats and privileges in a deployed architecture enables us to identify, via an attack graph analysis, specific vulnerabilities the exploitation of which facilitates threat scenarios. The defender can then prioritise the remediation of those vulnerabilities (e.g., through patching, traffic isolation, etc.). Our analysis can be applied for specific hazards (e.g., those with the potential for more serious consequences) at system design time or, alternatively, at runtime. In the first case case the results of the analysis can be used, within an iterative design process, with different levels of details [36] 1) to produce security requirements for the design team to improve the security of the system architecture (e.g., changing function allocation, network segmentation, etc.). 2) to prove the effectiveness of mitigations to limit the impact on safety of attacks. Similarly, at runtime, it is possible to use the result of our methodology to identify and apply security measures (e.g., host isolation, patching, etc.) in the optic of preserving safety. We have proposed and integrated a concrete set of tools (UPAAL, Mulval, Z3) and developed additional algorithms to support this analysis.

Complexity and Scalability. Threat analysis using STRIDE and safety analysis using STAMP/STPA is common and these methods have been applied to large scale complex systems (e.g., [18], [32], [34], [41], [42] for STPA/STPA-Sec). The main scalability limitations to apply our methodology stem from the formal verification process used to identify and enumerate the safety critical attacks. We have taken several steps to mitigate this: using automata to encode only high level behaviour, decoupling between the functional model and its more complex deployment via the use of privileges, limiting the analysis in time. For example, the CBTC use case 3.1 shows the effectiveness of the decoupling. While its SCS (Figure 3) is relatively simple and straightforward to analyse, its deployed architecture (Figure 10) is significantly more complex. However, the complexity arising from the formal verification process is unavoidable and common to the application of such formal techniques in safety analysis and software engineering in general.

The completeness of the analysis and of the results obtained depends on the completeness of the models and of the formal analysis process. By limiting the analysis in time, we can compromise that completeness. For example, by restricting the analysis to a too short time frame we have missed a possible attack (tampering of clear) because it was buried too deep in the state space. Running the analysis for longer requires computational resources but also more human intervention to manage the cases where the model-checker is stuck. Further optimisations to manage the complexity of the process are possible. For example, by synthesising a more specific model for the MAA using more information from the attack graphs or by using more efficient strategies for the formal verification. We leave the investigation of such optimisations for future work.

An additional limitation of our work lies in the mapping between STRIDE threats and the privileges in the system architecture. This can be a complex process, which we have carried out manually. It could be, at least partially, automated but we have not yet investigated this in depth.

Genericity. The methodology proposed remains generic. It can be applied to different CPS and could be used with different methods for threat analysis and safety analysis as long as the integration points mentioned above remain applicable. Similarly, different tools (e.g., model-checkers, SAT solvers, tools for attack graph generation) could be used for the analysis.

11 Conclusions

Cyber-Physical Systems increasingly pervade all aspects of daily life. As systems are increasingly interconnected and new devices are increasingly being added to them, their attack surface increases exponentially. It becomes increasingly difficult to defend such systems against all possible attacks and it is therefore essential to identify and remediate those attacks that can lead to accidents and hazards, and threaten loss of life. This requires a combined safety and security analysis. However, although several attempts have been made to bridge between the two, no definitive methodology has emerged. We have proposed Tiresias, a methodology that combines STPA analysis with STRIDE and applies formal techniques to identify the threat scenarios and the vulnerabilities that facilitate them. Equally importantly, we have also proposed an integrated tool-chain to perform this analysis and demonstrated its use in a use cases.

Our methodology relies on building a syntehsised model of possible attacks from the security analysis, combining this with the safe control behavioural model of the CPS and formally verifying that safety properties (derived from the safety analysis) are preserved. This takes into account possible attack steps as well as their cascading effects in the system's behavioural model. It also takes into account the ordering/timing of events. The output from this analysis is

used in conjunction with attack-graph analysis to identify which specific vulnerabilities to remediate to remove the attack paths leading to accidents and hazards.

Several mappings need to be established across different models to make this possible. The generation of a combined MAA from the results of the threat analysis, the analysis of the safety control structure and the generation of safety properties are key for integrating threat analysis and safety analysis. Reasoning over the privileges acquired as a result of the exploitation of vulnerabilities is key to integrate with attack-graph analysis and identify the specific vulnerabilities that need to be remediated. Several strategies and metrics can then be employed to prioritise between them and safeguard the minimal safety set. Security measures can themselves have an impact on safety. While our tool-chain does not yet support the application of countermeasures on system design, our method allows to verify that changes do not lead to the violation of other safety properties. We leave this analysis for further work. Like with most other formal verification and model-checking based approaches, there are complexity and scalability challenges. Several optimisations are still possible to considerably mitigate them that we leave for further work. Our methodology could be applied with different approaches for safety analysis, security analysis and formal verification as well as with different tools. The work presented there could therefore form the foundation for further investigations and new solutions in this space.

REFERENCES

[1] BBC. (2021) Hacker tries to poison water supply of florida city. [Online]. Available: https://www.bbc.co.uk/news/world-us-canada-55989843

[2] J. Slowik, "Anatomy of an attack: Detecting and defeating crashoverride," *VB2018, October*, 2018.

[3] ESET, "Industroyer2. Industroyer reloaded." https://www.welivesecurity.com/2022/04/12/industroyer2-industroyer-reloaded/, 2022, [Online; accessed 20-April-2022].

[4] A. Ross. (2022) 'cyberpartisans' hack belarusian railway to disrupt russian buildup. [Online]. Available: https://www.theguardian.com/world/2022/jan/25/cyberpartisans-hack-belarusian-railway-to-disrupt-russian-buildup

[5] S. Shane and D. E. Sanger, "Drone crash in iran reveals secret us surveillance effort," *The New York Times*, vol. 7, 2011.

[6] W. Young and R. Porada, "System-theoretic process analysis for security (stpa-sec): Cyber security and stpa," in *2017 STAMP Conference*, 2017.

[7] A. Shostack, "Experiences threat modeling at microsoft." *MODSEC@ MoDELS*, vol. 2008, 2008.

[8] F. Crawley and B. Tyler, *HAZOP: Guide to best practice*. Elsevier, 2015.

[9] D. H. Stamatis, *Failure mode and effect analysis: FMEA from theory to execution*. Quality Press, 2003.

[10] J. B. Bowles, "The new sae fmeca standard," in *Annual Reliability and Maintainability Symposium. 1998 Proceedings. International Symposium on Product Quality and Integrity*. IEEE, 1998, pp. 48–53.

[11] C. A. Ericson and C. Ll, "Fault tree analysis," in *System Safety Conference, Orlando, Florida*, vol. 1, 1999, pp. 1–9.

[12] S. Jajodia, S. Noel, and B. O'berry, "Topological analysis of network attack vulnerability," in *Managing cyber threats*. Springer, 2005, pp. 247–266.

[13] G. Macher, H. Sporer, R. Berlach, E. Armengaud, and C. Kreiner, "Sahara: a security-aware hazard and risk analysis method," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 621–624.

[14] B. Meng, D. Larraz, K. Siu, A. Moitra, J. Interrante, W. Smith, S. Paul, D. Prince, H. Herencia-Zapana, M. F. Arif *et al.*, "Verdict: a language and framework for engineering cyber resilient and safe system," *Systems*, vol. 9, no. 1, p. 18, 2021.

[15] D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha, "Compositional verification of architectural models," in *NASA Formal Methods Symposium*. Springer, 2012, pp. 126–140.

[16] MITRE. (2023) Common attack pattern enumeration and classification. [Online]. Available: https://capec.mitre.org/

[17] S. Longari, A. Cannizzo, M. Carminati, and S. Zanero, "A secure-by-design framework for automotive on-board network risk analysis," in *2019 IEEE Vehicular Networking Conference (VNC)*. IEEE, 2019, pp. 1–8.

[18] R. Khan, K. McLaughlin, D. Laverty, and S. Sezer, "Stride-based threat modeling for cyber-physical systems," in *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. IEEE, 2017, pp. 1–6.

[19] H. Esquivel-Vargas, J. H. Castellanos, M. Caselli, N. O. Tippenhauer, and A. Peter, "Identifying near-optimal single-shot attacks on icss with limited process knowledge," *arXiv preprint arXiv:2204.09106*, 2022.

[20] I. Stellios, P. Kotzanikolaou, and C. Grigoriadis, "Assessing iot enabled cyber-physical attack paths against critical systems," *Computers & Security*, vol. 107, p. 102316, 2021.

[21] M. Barrère and C. Hankin, "Analysing mission-critical cyber-physical systems with and/or graphs and maxsat," *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 3, pp. 1–29, 2021.

[22] S. Kriaa, M. Bouissou, F. Colin, Y. Halgand, and L. Pietre-Cambacedes, "Safety and security interactions modeling using the bdmp formalism: case study of a pipeline," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2014, pp. 326–341.

[23] I. Gashi, A. Povyakalo, and L. Strigini, "Diversity, safety and security in embedded systems: modelling adversary effort and supply chain risks," in *2016 12th European Dependable Computing Conference (EDCC)*. IEEE, 2016, pp. 13–24.

[24] C. Ponsard, J. Grandclaudon, and P. Massonet, "A goal-driven approach for the joint deployment of safety and security standards for operators of essential services," *Journal of Software: Evolution and Process*, p. e2338, 2021.

[25] C. Kolb, S. M. Nicoletti, M. Peppelman, and M. Stoelinga, "Model-based safety and security co-analysis: a survey," *arXiv preprint arXiv:2106.06272*, 2021.

[26] L. Piètre-Cambacédès and M. Bouissou, "Cross-fertilization between safety and security engineering," *Reliability Engineering & System Safety*, vol. 110, pp. 110–126, 2013.

[27] R. Kumar, B. Narra, R. Kela, and S. Singh, "Afmt: Maintaining the safety-security of industrial control systems," *Computers in Industry*, vol. 136, p. 103584, 2022.

[28] L. Pietre-Cambacedes, E. L. Quinn, and L. Hardin, "Cyber security of nuclear instrumentation & control systems: overview of the iec standardization activities," *IFAC Proceedings Volumes*, vol. 46, no. 9, pp. 2156–2160, 2013.

[29] L. Pietre-Cambacedes and E. L. Quinn, "Iec 62859: towards an international standard on the coordination between safety and cybersecurity for nuclear i&c systems," in *9th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human Machine Interface Technologies (NPIC&HMIT 2015)*.

[30] EUROCAE, *ED-203A - Airworthiness Security Methods and Considerations*, 2018.

[31] W. Young and N. Leveson, "Systems thinking for safety and security," in *Proceedings of the 29th Annual Computer Security Applications Conference*, 2013, pp. 1–8.

[32] N. G. Leveson, *Engineering a safer world: Systems thinking applied to safety*. The MIT Press, 2016.

[33] R. Patriarca, M. Chatzimichailidou, N. Karanikas, and G. Di Gravio, "The past and present of system-theoretic accident model and processes (stamp) and its associated techniques: A scoping review," *Safety science*, vol. 146, p. 105566, 2022.

[34] S. Khan, S. Madnick, and A. Moulton, "Cyber-safety analysis of an industrial control system for chillers using stpa-sec," 2018.

[35] P. Beaumont and S. Wolthusen, "Micro-grid control security analysis: Analysis of current and emerging vulnerabilities," *Critical Infrastructure Security and Resilience: Theories, Methods, Tools and Technologies*, pp. 159–184, 2019.

[36] M. Span, L. O. Mailloux, R. F. Mills, and W. Young, "Conceptual systems security requirements analysis: Aerial refueling case study," *IEEE Access*, vol. 6, pp. 46 668–46 682, 2018.

[37] I. Friedberg, K. McLaughlin, P. Smith, D. Laverty, and S. Sezer, "Stpa-safesec: Safety and security analysis for cyber-physical sys-

tems," *Journal of information security and applications*, vol. 34, pp. 183–196, 2017.

[38] S. Khan and S. E. Madnick, "Cybersafety: A system-theoretic approach to identify cyber-vulnerabilities & mitigation requirements in industrial control systems," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[39] N. P. De Souza, C. d. A. C. César, J. de Melo Bezerra, and C. M. Hirata, "Extending stpa with stride to identify cybersecurity loss scenarios," *Journal of Information Security and Applications*, vol. 55, p. 102620, 2020.

[40] L. M. Castiglione, Z. Hau, K. T. Co, L. Muñoz-González, F. Teng, and E. Lupu, "Ha-grid: Security aware hazard analysis for smart grids," in *2022 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2022, pp. 446–452.

[41] A. Nourian and S. Madnick, "A systems theoretic approach to the security threats in cyber physical systems applied to stuxnet," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 2–13, 2015.

[42] J. P. Thomas IV, "Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis," Ph.D. dissertation, Massachusetts Institute of Technology, 2013.

[43] A. Abdulkhaleq and S. Wagner, "Integrated safety analysis using systems-theoretic process analysis and software model checking," in *Computer Safety, Reliability, and Security: 34th International Conference, SAFECOMP 2015, Delft, The Netherlands, September 23-25, 2015, Proceedings 34*. Springer, 2015, pp. 121–134.

[44] A. Abdulkhaleq, S. Wagner, and N. Leveson, "A comprehensive safety engineering approach for software-intensive systems based on stpa," *Procedia Engineering*, vol. 128, pp. 2–11, 2015.

[45] A. L. Dakwat and E. Villani, "System safety assessment based on stpa and model checking," *Safety science*, vol. 109, pp. 130–143, 2018.

[46] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. ieee, 1977, pp. 46–57.

[47] A. Scarinci, A. Quilici, D. Ribeiro, F. Oliveira, D. Patrick, and N. G. Leveson, "Requirement generation for highly integrated aircraft systems through stpa: An application," *Journal of Aerospace Information Systems*, vol. 16, no. 1, pp. 9–21, 2019.

[48] C. Zhao, L. Dong, H. Li, and P. Wang, "Safety assessment of the reconfigurable integrated modular avionics based on stpa," *International Journal of Aerospace Engineering*, vol. 2021, 2021.

[49] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks, "Uppaal 4.0," 2006.

[50] J. C. Mitchell, V. Shmatikov, and U. Stern, "Finite-state analysis of ssl 3.0." in *USENIX Security Symposium*, 1998, pp. 201–216.

[51] S. Hussain, O. Chowdhury, S. Mehnaz, and E. Bertino, "Lteinspector: A systematic approach for adversarial testing of 4g lte," in *Network and Distributed Systems Security (NDSS) Symposium 2018*, 2018.

[52] S. Hu, Q. A. Chen, J. Sun, Y. Feng, Z. M. Mao, and H. X. Liu, "Automated discovery of denial-of-service vulnerabilities in connected vehicle protocols," in *USENIX Security Symposium*, 2021.

[53] S.-J. Moon, Y. Bieri, R. Martins, and V. Sekar, "Automatic discovery of evasion attacks against stateful firewalls," 2021.

[54] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "Use of formal methods at amazon web services," *See http://research. microsoft. com/en-us/um/people/lamport/tla/formal-methods-amazon. pdf*, 2014.

[55] A.-L. Alshalalfah, G. B. Hamad, and O. A. Mohamed, "Towards system level security analysis of artificial pancreas via uppaal-smc," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[56] E. Poorhadi, E. Troubitysna, and G. Dán, "Formal modelling of the impact of cyber attacks on railway safety," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2021, pp. 117–127.

[57] Y. Papadopoulos and J. A. McDermid, "The potential for a generic approach to certification of safety critical systems in the transportation sector," *Reliability engineering & system safety*, vol. 63, no. 1, pp. 47–66, 1999.

[58] D. Standard, "Standard 00-56 on safety management requirements for defence systems," *Ministry of Defence, Directorate of Standardisation, Kentigern House*, vol. 65, 2007.

[59] P. Helle, "Automatic sysml-based safety analysis," in *Proceedings of the 5th International Workshop on Model Based Architecting and Construction of Embedded Systems*, 2012, pp. 19–24.

[60] N. G. Leveson and J. P. Thomas, "Stpa handbook," *Cambridge, MA, USA*, 2018.

[61] L. M. Castiglione and E. C. Lupu, "Hazard driven threat modelling for cyber physical systems," in *Proceedings of the 2020 Joint Workshop on CPS&IoT Security and Privacy*, 2020, pp. 13–24.

[62] European railway traffic management system (ertms). [Online]. Available: https://www.era.europa.eu/activities/european-rail-traffic-management-system-ertms_en

[63] L. Wanninger, "Introduction to network rtk," *IAG Working Group*, vol. 4, no. 1, pp. 2003–2007, 2004.

[64] L. M. Castiglione, P. Falcone, A. Petrillo, S. P. Romano, and S. Santini, "Cooperative intersection crossing over 5g," *IEEE/ACM Transactions on Networking*, 2020.

[65] M. Comptier, D. Déharbe, J. M. Perez, L. Mussat, T. Pierre, and D. Sabatier, "Safety analysis of a cbtc system: a rigorous approach with event-b," in *International Conference on Reliability, Safety and Security of Railway Systems*. Springer, 2017, pp. 148–159.

[66] A. Abdulkhaleq and S. Wagner, "A systematic and semi-automatic safety-based test case generation approach based on systems-theoretic process analysis," *arXiv preprint arXiv:1612.03103*, 2016.

[67] L. d. Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[68] M. Y. Vardi, "Branching vs. linear time: Final showdown," in *International conference on tools and algorithms for the construction and analysis of systems*. Springer, 2001, pp. 1–22.

[69] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking for real-time systems," in *[1990] Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*. IEEE, 1990, pp. 414–425.

[70] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal 4.0," *Department of computer science, Aalborg university*, 2006.

[71] L. Lamport, "Real time is really simple," *Microsoft Research*, pp. 2005–30, 2005.

[72] M. Albanese, S. Jajodia, and S. Noel, "Time-efficient and cost-effective network hardening using attack graphs," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*. IEEE, 2012, pp. 1–12.

[73] A. Shostack, *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[74] M. Muckin and S. C. Fitch, "A threat-driven approach to cyber security," *Lockheed Martin Corporation*, 2014.

[75] J. Lambert, "Defenders think in lists. Attackers think in graphs. As long as this is true, attackers win." https://github.com/JohnLaTwC/Shared/, 2015, [Online; accessed 07-November-2021].

[76] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 217–224.

[77] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006, pp. 336–345.

[78] A. Horn, "On sentences which are true of direct unions of algebras1," *The Journal of Symbolic Logic*, vol. 16, no. 1, pp. 14–21, 1951.

[79] P. Bulychev, A. David, K. G. Larsen, M. Mikučionis, D. B. Poulsen, A. Legay, and Z. Wang, "Uppaal-smc: Statistical model checking for priced timed automata," *arXiv preprint arXiv:1207.1272*, 2012.

[80] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM journal of research and development*, vol. 6, no. 2, pp. 200–209, 1962.

[81] X. Ou, S. Govindavajhala, A. W. Appel *et al.*, "Mulval: A logic-based network security analyzer." in *USENIX security symposium*, vol. 8. Baltimore, MD, 2005, pp. 113–128.

[82] BBC. (2019) Raspberry pi used to steal data from nasa lab. [Online]. Available: https://www.bbc.co.uk/news/technology-48743043

[83] D. Michail, J. Kinable, B. Naveh, and J. V. Sichi, "JGraphT–A Java Library for Graph Data Structures and Algorithms," *ACM Trans. Math. Softw.*, vol. 46, no. 2, May 2020.

Luca Maria Castiglione received the M.Sc. degree in computer science and engineering from the University of Napoli Federico II. He is currently pursuing his PhD with the Resilient Information Systems Security (RISS) Group at Imperial College London. His research interests apply to safety and security of Cyber Physical Systems.

Emil C. Lupu is Professor of Computer Systems in the Department of Computing at Imperial College London where he leads the Resilient Information Systems Security Group. He is a Security Science Fellow with Imperial's Institute for Security Science and Technology. His research activities focus on the resilience of systems to adversarial threats and means to enable their safe operation even when parts of the systems have been compromised.