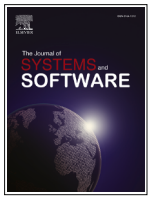

# Bridging safety and security in complex systems: A model-based approach with SAFT-GT toolchain

The Journal of Systems and Software 238 (2026) 112865

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss



# Bridging safety and security in complex systems: A model-based approach with SAFT-GT toolchain

Irdin Pekaric [a,*], Raffaela Groner [b], Alexander Raschke [c], Thomas Witte [c], Jubril Gbolahan Adigun [d], Michael Felderer [d,e,f], Matthias Tichy [c]

[a] Universität Liechtenstein, Department of Information Systems and Computer Science, Fürst-Franz-Josef-Strasse, 9490, Vaduz, Liechtenstein
[b] Chalmers University of Technology and University of Gothenburg, Department of Computer Science and Engineering, Hörselgången, 41296, Gothenburg, Sweden
[c] Ulm University, Institute of Software Engineering and Programming Languages, James-Franck-Ring 9, 89081, Ulm, Germany
[d] University of Innsbruck, Department of Computer Science, Technikerstraße 21a, A-6020, Innsbruck, Austria
[e] German Aerospace Center (DLR), Institute of Software Technology, Muenchener Strasse 20, 82234, Wessling, Germany
[f] University of Cologne, Department of Mathematics and Computer Science, Albertus-Magnus-Platz, 50923, Cologne, Germany

ARTICLE INFO

Editor: Dr Antonio Filieri

*Keywords:*
Attack-fault tree
Safety and security analysis
Self-adaptive system
Model formalism
Expert survey

ABSTRACT

In the rapidly evolving landscape of software engineering, the demand for robust and secure systems has become increasingly critical. This is especially true for self-adaptive systems due to their complexity and the dynamic environments in which they operate. To address this issue, we designed and developed the SAFT-GT toolchain that tackles the multifaceted challenges associated with ensuring both safety and security. This paper provides a comprehensive description of the toolchain's architecture and functionalities, including the Attack-Fault Trees generation and model combination approaches. We emphasize the toolchain's ability to integrate seamlessly with existing systems, allowing for enhanced safety and security analyses without requiring extensive modifications and domain knowledge. Our proposed approach can address evolving security threats, including both known vulnerabilities and emerging attack vectors that could compromise the system. As a use case for the toolchain, we integrate it into the feedback loop of self-adaptive systems. Finally, to validate the practical applicability of the toolchain, we conducted an extensive user study involving domain experts, whose insights and feedback underscore the toolchain's relevance and usability in real-world scenarios. Our findings demonstrate the toolchain's effectiveness in real-world applications while highlighting areas for future improvements. The toolchain and associated resources are available in an open-source repository to promote reproducibility and encourage further research in this field.

## 1. Introduction

The growing complexities of modern systems have necessitated the employment of different approaches in their analysis (Pekaric et al., 2023). These approaches often attempt to reduce human effort in the delivery of various services. This is achieved through the incorporation of autonomy and allowing systems to be aware of their physical environment (Ceccarelli and Montecchi, 2023). This fundamental ability enables software systems to sense and reason about the system behaviors as well as to react by exhibiting self-adaptation — giving rise to (SASs). SASs can modify and optimize their performance based on feedback and the changing conditions, as well as the system goals (Weyns et al., 2023).

Despite all the benefits that SASs provide, they also bring various challenges that often involve security and safety concerns. According to the practitioners in the SASs domain, these two aspects represent some of the biggest problems related to these types of systems. This was demonstrated in the recent industrial survey conducted by Weyns et al. (2023). The issues often occur due to various types of vulnerabilities, which can cause parts of a system to fail, causing the overall state of a system to become unsafe. Many of these vulnerabilities arise from diverse hardware and software configurations, which necessitates an ongoing analysis and vigilant monitoring throughout their operational lifecycle. Given the interconnected nature of these challenges, it is of utmost importance to consider security and safety jointly, as these two aspects significantly affect each other and can determine the overall reliability of the system.

However, joint consideration of safety and security aspects in the context of SASs and their uncertainty is a complex and challenging endeavor (Prikler and Wotawa, 2022). One promising approach to handle

* Corresponding author.
*E-mail addresses:* irdin.pekaric@uni.li (I. Pekaric), raffaela@chalmers.se (R. Groner), alexander.raschke@uni-ulm.de (A. Raschke), thomas.witte@uni-ulm.de (T. Witte), jubril.adigun@student.uibk.ac.at (J. Gbolahan Adigun), michael.felderer@dlr.de (M. Felderer), matthias.tichy@uni-ulm.de (M. Tichy).

https://doi.org/10.1016/j.jss.2026.112865
Received 25 February 2025; Received in revised form 2 December 2025; Accepted 23 March 2026
Available online 27 March 2026

this complexity is to leverage a joint safety and security analysis using (AFTs). AFTs are a powerful analytical tool that combines (AT) and (FTs) in a single modeling formalism (André et al., 2019; Kumar and Stoelinga, 2017), allowing for a systematic exploration of the interplay between safety and security issues (Groner et al., 2023). They provide a graphical representation of the various ways in which a system can fail (faults) or be compromised (through attacks (Pekaric et al., 2021a,b)), and how these faults and attacks can propagate through the system. This is achieved by integrating safety and security considerations at runtime when the system is in its operational context, allowing for a more holistic and robust analysis of potential safety risks and system vulnerabilities (Witte et al., 2022).

AFTs can be verified using existing model-checking techniques, e.g., critical path analysis, or calculation of failure rates and probabilities. Generating large AFTs for realistic systems by hand, however, is error-prone and infeasible. In comparison, it is much easier for safety experts to create only FT models for a target system separately as well as security experts generating AT models, which are at a much lower abstraction level (Giese and Tichy, 2006) compared to creating AFTs.

One disadvantage of AFTs, however, is that they cannot model the system state but only possible hazards. Thus, we restrict ourselves to architectural reconfigurations that change the data flow and/or the hardware and software components of a managed system. Since our (SAFT-GT) (Groner et al., 2023) generates AFTs for cyber-physical systems based on the data flow and the used hardware and software components, the AFTs we analyze take the current system state into account implicitly.

By incorporating AFTs into the SASs loop, the system can leverage real-time data to dynamically update its models (due to potential security and safety issues), which corresponds to their defining characteristics. Furthermore, the generation of AFTs allows for rapid adaptation to new threats and vulnerabilities, enhancing the system's resilience. This integration not only facilitates proactive risk management but also supports informed decision-making in the planning phase, ultimately leading to more robust and secure self-adaptive systems that can autonomously respond to emerging challenges in their operational environment.

### 1.1. Overview of the approach

Fig. 1 shows the integration of the presented approach in a self-adaptive system controlled by a MAPE-K (Kephart and Chess, 2003) loop: First, a managed system forms the basis. Our example system implements a control system for a quadcopter with ROS2 components (robot operating system (Macenski et al., 2022)) to plan trajectories and avoid obstacles, as well as high-level scripting of robot behavior, e. g., waypoints to follow. Second, a MAPE-K subsystem that monitors the state of the ROS node graph and messages, can reconfigure the application by activating or deactivating components of the managed system, e. g., to switch between a mission and safety behavior, such as flying back to origin. Finally, our approach performs the safety and security analysis in the MAPE-K loop feeding the knowledge base with risk values for hazards. This information is used by the planning component to influence or trigger future adaptation decisions. More details are given in Fig. 5 in Section 4.

Our approach focuses on the security problems that lead to a failure of the system. Therefore, other security aspects, such as confidentiality or data integrity, are only considered if they can also lead to a failure. For example, the manipulation of sensor data might lead to a drone crashing into an obstacle due to manipulated LiDAR sensor data.

### 1.2. Contributions

In order to close the feedback loop, we extend our previously published SAFT-GT (Groner et al., 2023) pipeline by a transformation of the generated AFTs into (DFTs) and use them as input for the Storm model checker.[1] We evaluate our extension and the overall approach by conducting a workshop with domain experts and an experimental evaluation of the approach. Additionally, we perform time measurements to evaluate the performance of the extended SAFT-GT pipeline, which provides insights on whether the pipeline is applicable in a real system. Specifically, we provide the following contributions:

- We provide a detailed description of our previously implemented SAFT-GT approach (Groner et al., 2023).
- We extend our SAFT-GT approach by applying a (PMC) to verify the output of the AFT generator.
- We link and evaluate SAFT-GT through the feedback loop of self-adaptive systems. To this end, we used a SASs built upon the robot operating system (ROS) (Macenski et al., 2022).
- We conduct a workshop that includes two surveys and expert discussions. Based on the insights we obtained, we discuss the strengths and limitations of the developed approach and outline future research directions.
- We perform experiments to evaluate the performance of the proposed approach.

We provide the complete toolchain, including the models of the presented approach and examples for download, fostering further research and collaboration in the field: https://github.com/sp-uulm/saft-gt.

### 1.3. Paper structure

The remainder of the paper is structured as follows: Section 2 provides background information on self-adaptive systems, safety and security modeling and security concepts (including metrics). Section 3 demonstrates a running example for which the proposed approach can be applied. Section 4 presents each separate model that we utilize in our approach, while Section 5.1 explains how these models are combined, verified and integrated into the feedback loop of self-adaptive systems. Section 6 offers an evaluation of the developed approach through expert surveys, discussions as well as performance evaluation. Section 7 elaborates on the limitations of the approach as well as experts' feedback. Section 8 discusses the related work on AFT generation. Finally, Section 9 concludes the work.

## 2. Background

In this section, we provide the necessary background on self-adaptive systems, safety and security models used in this work and relevant security concepts and metrics.

### 2.1. Self-adaptive system

SASs are systems that adjust their behavior to optimally fulfill their requirements even if their environment or goals change. Usually, a SASs consists of two subsystems: 1) the managed subsystem, which should fulfill a predefined goal, and 2) the managing subsystem that controls the adjustment of the behavior of the managed subsystem (Weyns et al., 2013). For example, a drone that should independently plan and execute missions can be considered as a part of the SASs. The drone forms the managed subsystem, and the corresponding managing subsystem plans new missions and adjusts the drone's behavior to complete a mission.

A well-known architecture used for the managing subsystem is MAPE-K (**M**onitor, **A**nalyze, **P**lan, **E**xecute - **K**nowledge) (Kephart and Chess, 2003). Fig. 1 in Section 1.1 provides an overview of the relationships between managed subsystem, managing subsystem, and MAPE-K. The Monitor component observes the state and behavior of the managed subsystem as well as its environment. The Analyze component decides

[1] https://www.stormchecker.org/

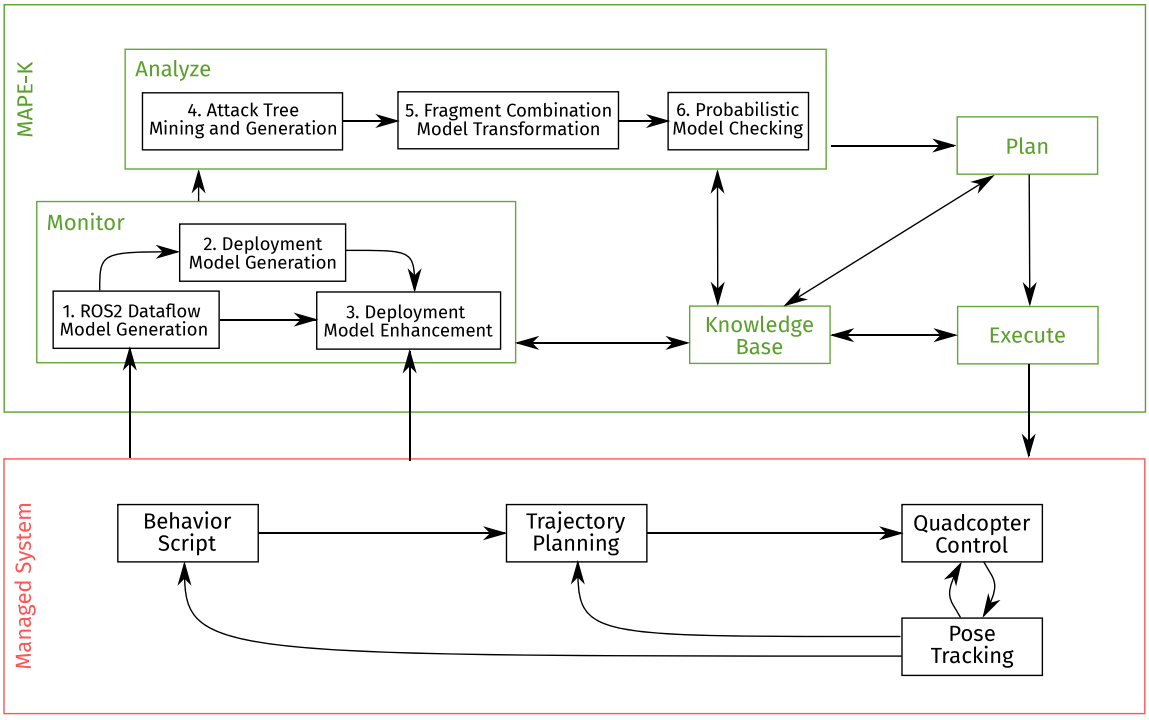



**Fig. 1.** Integration of our approach in the MAPE-K loop.

whether the managed subsystem should adapt its behavior. For example, the analyze component might detect that the drone is running low on energy and decide to trigger an adaptation to avoid damage due to a crash. The Plan component plans the adjustment of the managed subsystem. For example, the Plan component determines a plan that forces the drone to abandon its current mission and initiate a safe landing. Finally, the Execute component implements the plan of the Plan component. The Knowledge component serves as the centralized provision of information. The other components use the Knowledge component to store information and to communicate indirectly with each other (Arcaini et al., 2015).

### 2.2. Safety and security modeling

There is a variety of modeling approaches used in the context of safety and security analysis. Our work leverages the security-related models (ATs), the safety-related models FTs, and AFTs, which are used in a joined safety and security modeling approach. In the following, we provide background information on these three established models in more detail. We use a running example that describes how a flooding attack can lead to a drone crashing into a person to illustrate these models.

#### 2.2.1. Attack Trees

(ATs) are models used to represent adversary actions regarding how a certain system or component can be targeted (Mauw and Oostdijk, 2006; Schneier, 1999). They consist of a root node, which represents the goal of an attack, attack steps or intermediate targets, and logical gates. Depending on the goal of the analysis, different values are assigned to attack sets within the three. These can include values such as the expected costs or the probability of performing the respective attack set successfully. Logical gates, e.g., AND and OR, are used to model the relationships between individual attack steps necessary to reach the goal of the modeled attack (Mauw and Oostdijk, 2006; Muzammil et al., 2024; Lallie et al., 2020).

Fig. 2 shows a simplified example of an AT for a packet flooding attack. It demonstrates that an attacker needs to "identify the receiving components" in a system and "generate traffic" in order to perform a flooding attack. The probabilities specified for each attack step (depicted as ellipses) indicate the probability of successfully performing the respective attack step.

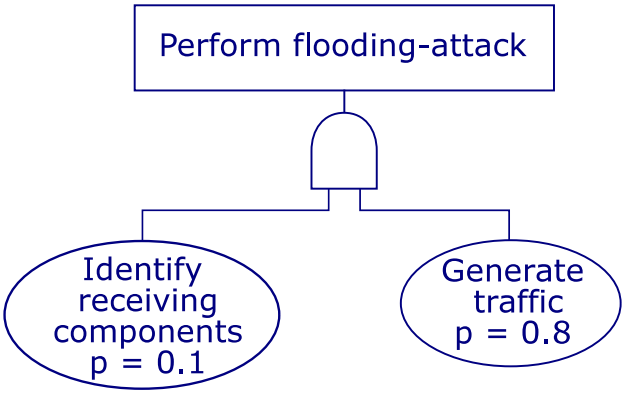



**Fig. 2.** Example of an Attack Tree (AT).

#### 2.2.2. Fault Trees

FTs represent a common formalism from safety analysis that is used to model possible hazards and their causes (Vesely et al., 1981; Pai and Dugan, 2002). In addition to standard FTs, there are also more specialized variants, such as DFTs (Dugan et al., 1992), which consider temporal sequences of events or Repairable Fault Trees can also model complex repairable systems (Raiteri et al., 2004). However, FTs are in general acyclic graphs that consist of two categories of nodes, namely events and logical gates. Regarding events, a distinction is drawn between the top event, basic events, and intermediate events. The top event represents the hazard to be analyzed. Basic events model events that occur randomly and intermediate events model events caused by other events. Logical gates are used to model the propagation of failures through the system. Standard FTs can only model simple logical relationships such

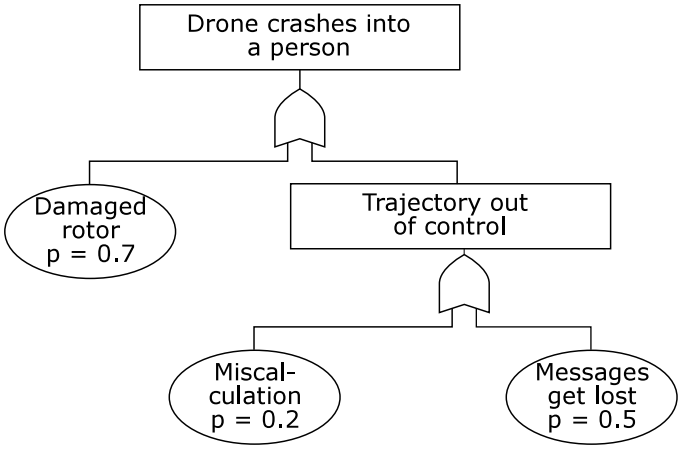



**Fig. 3.** Example of a Fault Tree (FT).

as AND and OR (Ruijters and Stoelinga, 2015). DFTs, however, offer dynamic and temporal gates that enable more complex combinations of events. For example, a Simultaneous-AND (SAND) gate triggers when all incoming events occur at the same time. Another example is a Priority-AND (PAND) gate, which triggers when all incoming events occur in their predefined order (Edifor et al., 2012).

Fig. 3 shows a simplified example of a FT. This FT models that the failure that "Drone crashes into a person" can occur due to a damaged rotor with a probability of 0.7 or due to the drone being out of control, which can be caused by a miscalculation with a probability of 0.2 or by losing messages with a probability of 0.5.

#### 2.2.3. Attack-Fault Trees

Kumar and Stoelinga (Kumar and Stoelinga, 2017) present in their work Attack-Fault Trees (AFT), which combines FTs with (ATs) into a unified modeling mechanism to leverage a joint analysis of safety and security. To this end, FTs are enhanced with additional events that represent faults caused by malicious actions. These events are also called attack events. Attack events are analogous to an attack goal that serves as the root of an AT and, therefore, mark the juncture between FT and AT in an AFT (Ruijters and Stoelinga, 2015; Nai Fovino et al., 2009).

Fig. 4 shows an AFT resulting from combining the FT in Fig. 3 and the AT in Fig. 2. It should be noted that our example FT has been extended by one intermediate event ("Messages get lost") to model the context in which an attack leads to failure.

This example AFT describes possible events that can cause a drone to crash into a person. On the one hand, a damaged rotor can cause this failure. On the other hand, the loss of control over the trajectory of the drone can lead to the failure mentioned above. A miscalculation or the loss of messages can lead to the trajectory being out of control. Since the loss of messages can be caused by performing a flooding attack, the FT part of the AFT is extended with the AT in Fig. 2.

### 2.3. Security concepts and metrics

In this section, we outline the general security concepts and metrics that were utilized in the proposed modeling approach. CVE[2] data is used to distinguish between different *vulnerabilities*. These vulnerabilities have already been identified by security experts and recorded in order to provide a means of protection against existing threats. Each CVE has a unique identifier, description, the links to technical reports, advisories and patches.

[2] Common Vulnerabilities and Exposures, https://cve.mitre.org/

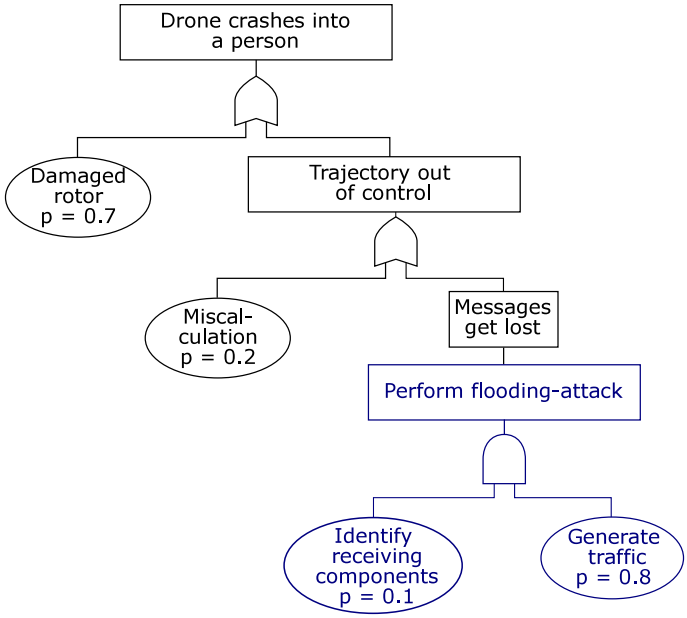



**Fig. 4.** Example of an Attack-Fault Tree (AFT).

CWE[3] entries represent specific higher-level groups to which CVEs are associated. This is done in order to provide a hierarchy for vulnerability data and these are formulated as *weaknesses*. There are two main hierarchies, which divide CVEs into software and hardware weakness types. Most of the CVEs contain a CVSS[4] vector. It provides various types of qualitative scores related to the severity of the vulnerability. Besides providing more detailed information for assessing severity, it also includes information about the impact of a CVE, the so-called CIA triad (Samonas and Coss, 2014; Sauerwein et al., 2019).

CPEs[5] portray various systems, software, and platforms, which are represented using syntax for Uniform Resource Identifiers (URI) including a specific version of a software or library. This allows security engineers and researchers to know exactly which software is affected by a certain CVE. In cybersecurity, attacks can also occur by exploiting multiple vulnerabilities, which form attack chains. Some of these chains are similar because they address CVEs that belong to the same CWE or they have corresponding mechanics. To represent these similarities, CAPEC[6] entries were created, which allow an easy understanding of common attacker actions. The aforementioned databases present a solid foundation for the proposed approach since they include vulnerabilities, weaknesses, platforms, and attack patterns.

Lastly, we also utilize the "Exploit Prediction Scoring System" (EPSS[7]). It describes "the probability that a vulnerability will be exploited in the wild within the first 12 months after public disclosure"(Jacobs et al., 2021). Since the existence of a system vulnerability does not mean that the vulnerability is instantly exploited, the score is used to estimate the probability of a successful attack over time (generated as a part of the attack model, wherein the scores are assigned automatically). The EPSS score is applied because we do not consider specific attacker profiles or other more detailed aspects, such as the duration and the success of an attack. Thus, an exponential distribution can be assumed for an AFT (Kumar and Stoelinga, 2017), which is also the default approach for FT analysis (Ruijters and Stoelinga, 2015). The reason we do

[3] Common Weakness Enumeration, https://cwe.mitre.org/
[4] Common Vulnerability Scoring System, https://www.first.org/cvss/
[5] Common Platform Enumerations, https://nvd.nist.gov/products/cpe
[6] Common Attack Pattern Enumeration and Classification, https://capec.mitre.org/
[7] Exploit Prediction Scoring System, https://www.first.org/epss

not consider attacker profiles is that we aim to make our approach as automated as possible, whereas attacker profiles need to be manually crafted by security experts. In addition, this can also result in a false sense of security due to profile inaccuracies as well as raise potential scalability issues (Lenin et al., 2014). However, due to the flexibility of the toolset, manual specification and integration of attacker profiles in the generated attack models are still possible.

## 3. Running example

We introduce a small example that we will use in the following sections to illustrate the models used. The scenario is an automatically controlled drone that could be used, for instance, for reconnaissance flights in agriculture to detect hidden animals in a field before the crops are harvested. The drone receives the trajectory from a control computer and transmits its position back so that the control computer can adjust the trajectory multiple times. We have simulated this scenario in our drone laboratory with the following necessary adjustments: The position data of the drone is determined via an external camera system (Optitrack). This data is then transferred to the actual control computer, which calculates the trajectory of the drone according to the area to be covered. The drone is then gradually sent instructions via the Wi-Fi network as to which points to fly over one after the other. Our system is based on ROS and consists of several nodes that communicate and interact with each other and perform various tasks such as transmitting position data, calculating the trajectory or communicating with the drone.

In the following, we will only consider the risk of a drone flying into a person. The manually developed FT therefore includes intermediate events such as “Trajectory out of control” or “Drone unable to fly”. These events can occur if the drone’s hardware has problems (broken rotors, low battery, etc.), but also if the control messages get lost. The latter can be due to poor connection quality as well as to specific attacks targeting the communication channel or the control computer directly. The individual nodes could, for example, be flooded with messages as a result of a denial-of-service (DoS) attack and thus no longer be able to keep up with the calculation. In the example FT, attack events targeting precisely those nodes that supply data for the node calculating the trajectory are therefore used as basic events.

While this is a simple example, it effectively demonstrates the potential of our approach. All models presented in the remainder of the paper, as well as the evaluation, are based on this example.

## 4. Models

In this section, we describe the use, the textual representation, and the process of obtaining each model we utilize in our AFT generation toolchain. We provide a detailed description of our generation approach that relies on these models in Section 5.1.

Fig. 5 includes all the individual steps and involved models of the presented toolchain. Detailed information about how these models are obtained is given in the subsections indicated in the figure. The principal run is as follows: After a reconfiguration of the ROS2 system, a run of the toolchain is triggered in the MAPE-K loop. This run starts with the (re-)generation of the (adapted) dataflow between the ROS nodes. From this dataflow model, an initial deployment model is derived that is enhanced with dependency information extracted from the running system. In the next step, open-source security databases (see Section 2.3) are mined in order to identify specific vulnerabilities that affect components used in the system. For each vulnerable component, an AT is generated. These (ATs) are combined with manually created and provided FTs using so-called AFT-fragments that are manually created by security and modeling experts. The resulting AFT is translated into an input model for a PMC whose result (MTTF) is fed back to the Knowledge base.

### 4.1. Dataflow model

**Summary:** Dataflow represents the dependency between two components. For example, if component B receives messages from component A, then there is a dependency between component B and component A. As a consequence, if component A has been compromised, it is easy to manipulate component B, so component B should also be considered compromised. In order to capture the consequences of an attack, it is therefore important in our approach to model the data flow between compromised components.

The dataflow model is a simplification of the ROS component meta-model similar to the abstract component meta-model in (Gherardi, 2013). The model is used to generate and integrate dataflow models from any system. It captures the logical components of a system and dataflows between them. This logical view is more abstract from the actual implementation and focuses on the communication between components. In consequence, the meta-model only consists of *components* – entities that provide, transform or process data (such as sensors, actors, or data processing units) – and *channels* – communication paths between components to send or receive messages, call functions, etc.

Since our prototypical implementation builds upon a ROS system, we provide a dataflow model generator for ROS2 systems where ROS nodes are mapped to components while ROS topics, services, and actions are mapped to channels. To overcome the problem that ROS2 systems do not define their interface statically, but connect to topics and services dynamically, our generator consists of a single ROS node that can be triggered to collect architectural and dataflow data using ROS’ introspection capabilities at runtime. The generator can be triggered repeatedly to monitor the system for changes or architectural reconfigurations.

Additionally, the textual representation of the model is designed to be manually extendable: additional components and channels can be defined manually and incorporated with the rest of the model.

**Obtained:** Automatically extracted from a running system and/or manually extended.

**Textual Representation:** Listing 1 shows a small excerpt of the dataflow model of our running example, which defines two components *simple_trajectory_server* and *trajectory_client*, and a channel *trajectory_assign*. Arbitrary properties can be added to components and channels by providing more detailed information about the corresponding entity. A component can be connected to a channel via the keyword `Connect`, indicating the direction of communication wherein `->` signifies outgoing, while `<-` represents incoming messages. This separate description of the communication ends makes it possible to define hyperedges with multiple senders and/or multiple recipients of a message, which is also possible in ROS2.

### 4.2. Deployment model

**Summary:** The deployment model denotes how the components and channels from the dataflow model are deployed on a specific system. This information has to be provided manually. Our toolchain automatically extends this initial information with the libraries on which a component depends. Dependencies that cannot be derived automatically (e.g., because the platform that a component runs on cannot be reached by our analysis tool) can be added manually. We do not rely on the component’s source code to obtain dependency information as snyk.[8] On the contrary, we utilize information about open files of the running processes of a target component. Since we directly identify all libraries and components present on the running system, including their specific versions, we intentionally do not model countermeasures explicitly. Instead, we focus on currently vulnerable components as reported by vulnerability databases. If a component has been patched or a

[8] https://snyk.io

```
Component simple_trajectory_server {
    property ros_name = "simple_trajectory_server";
}

Component trajectory_client {
    property full_ros_name = "/trajectory_client";
}

Channel trajectory_assign {
    property full_ros_name = "/trajectory_assignment";
    property ros_type = "msg/SeTrajectoryAssignment";
    property ros_channel_type = "topic";
}
Connect Component=simple_trajectory_server -> Channel=trajectory_assign;
Connect Component=trajectory_client <- Channel=trajectory_assign;
```

**Listing 1.** Excerpt of the dataflow model of our running example.

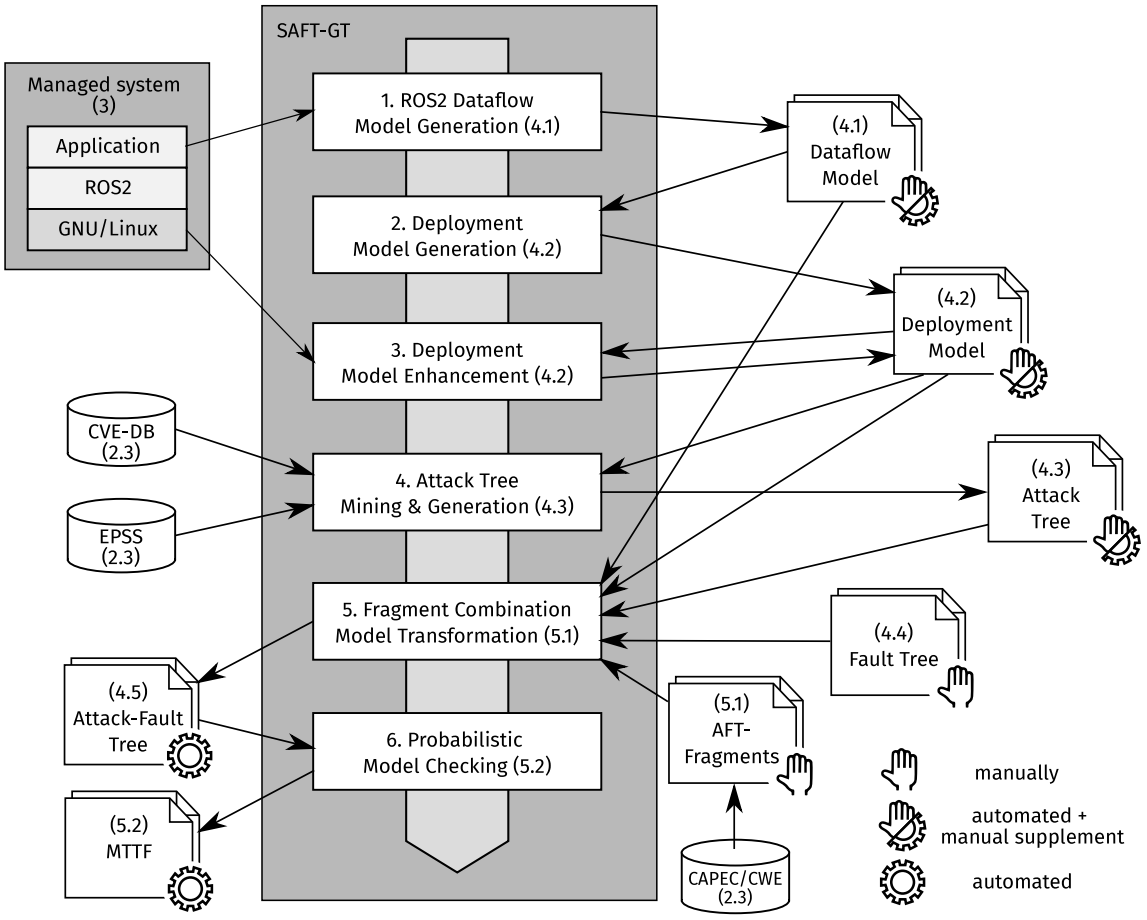



**Fig. 5.** Overview of the SAFT toolchain and produced/used artifacts.

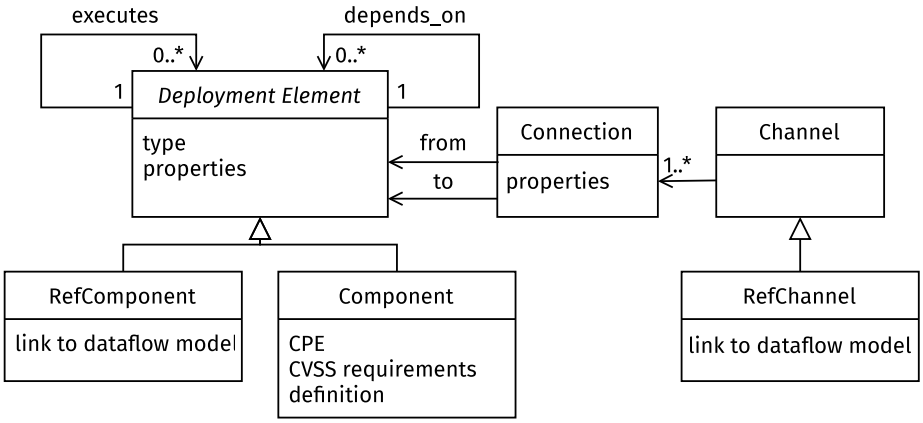



**Fig. 6.** Meta-model of deployment model.

countermeasure has been applied effectively, the corresponding version will no longer be marked as vulnerable and is therefore excluded from our analysis. This approach ensures that only relevant vulnerabilities are considered.

Fig. 6 shows a simplified meta-model of the deployment model. A *Deployment Element* is either a newly defined *Component* or a reference to a dataflow component (*RefComponent*). Similarly, new *Channels* between *Deployment Elements* can be defined — in addition to the already defined *Channels* within the dataflow model (*RefChannel*). Each component has an optional type representing different abstraction levels (e.g., File, Library, Package, Platform, OS, etc.) and arbitrary properties (key/value pairs). A (low-level) component might be related to a CPE entry or have CVSS requirements.

Each deployment element can be *executed* on another element or *depend on* other elements. This dependency information is generated recursively by our analysis tool via the used files and libraries of a component returned by Unix tools such as `lsof`[9] and `ldd`.[10] System-specific package managers as `apt`[11] and `dpkg`[12] abstract this information into package names for which CVEs can be identified. So far, the tool supports Ubuntu and Gentoo as platforms, but its architecture includes several abstraction layers to facilitate the integration of other platforms.

In the following step, the goal is to find the corresponding CPE for each identified package. For this purpose, we use the tool CPEguesser[13] in combination with some heuristic preprocessing such as shortening names, removing additional version information, etc. Lists, CPEs and all packages for which no CPE entry can be found are then passed to the AT Generator to determine possible CVEs for the identified software libraries.

**Obtained:** Automatically extracted from a running system and/or manually.

**Textual Representation:** In our running example, most ROS nodes run on a *Platform* called “rosbox”. The properties of the “rosbox”, including information about how our analysis tool can reach it, are used to gather more detailed information about the components running on it (see the excerpt in Listing 2). In the listing, one can see the already discovered dependency of the component “simple_trajectory_server” for library “libfastrtps” version 2.1.1.

### 4.3. Attack Tree model

**Summary:** The Attack Tree model is utilized to represent potential adversarial actions that can compromise the security of a system. It systematically outlines how an attacker can achieve specific goals by exploiting vulnerabilities within the system. Each node in the tree represents an attack step or a target, while the edges illustrate the relationships and dependencies between these steps. This hierarchical structure allows for a clear visualization of the various paths an attacker might take to reach their objective.

To effectively model adversarial actions, we define attack events within the AT that represent specific attack steps or exploits. Each attack event is characterized by its targeted software library (CPE), the method of attack (CVE), and the conditions that must be met for the attack to be successful. The AT employs logical gates to illustrate the relationships and dependencies between various attack steps, enabling a nuanced understanding of how different attack paths can converge or diverge.

For instance, in an AT, multiple attack events may be represented to illustrate different ways to compromise a system. One attack path could specify that to disable the communication channel of a drone, an attacker could flood the system with an extremely high number of network packets (DoS). Another attack scenario might involve exploiting a software vulnerability to gain unauthorized access to the drone’s control system. These relationships can be represented using logical gates to show that both conditions must be satisfied for the attack to succeed. By analyzing these attack paths, security experts can prioritize defenses and mitigation strategies based on the likelihood and impact of various attack scenarios. More information on how attack models are generated can be found as part of our previous work (Pekaric et al., 2024).

**Obtained:** Automatically generated from a running system based on library and package information.

**Textual Representation:** Listing 3 demonstrates an example of the notation that we utilize for (ATs). Each attack is presented as an `IntermediateEvent`, which is generated for each identified CPE of the running system. The CPE itself is stated in the description of the `IntermediateEvent`. Since each CPE can be affected by multiple CVEs, these are presented as separate `AttackSteps` or nodes that an attacker can exploit. `AttackSteps` are connected to the `IntermediateEvent` by logical gates[14] (AND, OR, PAND, etc.). More sophisticated attacks can involve additional child nodes that are

[9] https://github.com/lsof-org/lsof
[10] https://linux.die.net/man/1/ldd
[11] https://wiki.debian.org/AptCLI
[12] https://wiki.debian.org/Teams/Dpkg/
[13] https://github.com/cve-search/cpe-guesser

[14] For clarity and consistency, only standard logical gates (AND, OR, PAND, SAND) are supported in the current implementation, since the semantics of dynamic or repair gates (e.g., FDEP, SPARE) remain controversial when combining safety and security aspects.

```
ros_nodes:Platform = {simple_trajectory_server, quad_state_node,
                      bebop_driver_node, optitrack_motive, ...}

rosbox (hostname="x.x.x.x",reachable="ssh",sshUser="ros") =
        {ROS_foxy, Ubuntu_20, Linux}
rosbox executes {ros_nodes}

RefComponent simple_trajectory_server
    (ros_name="simple_trajectory_server") =
        {rosidl_typesupport_fastrtps_cpp_so, libfastcdr_so_1_0_13,
         libfastrtps_so_2_1_1, ...}
```

**Listing 2.** Excerpt of the deployment model of our running example.

```
IntermediateEvent description = "Generated for search by cpe for
    keyword: cpe:2.3:a:eprosima:fast_dds:2.1.1 Insufficient Control
    of Network Message Volume (Network Amplification)" {
    OR {
        AttackStep  CVE202138425
        description = "eProsima Fast DDS
           versions prior to 2.4.0 (#2269) are susceptible to
           exploitation when an attacker sends a specially crafted
           packet to flood a target device with unwanted traffic,
           which may result in a denial-of-service condition and
           information exposure."
        probability = 0.0
        CVE = "CVE-2021-38425"
        CVSS = "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:H"
        BaseScore = 9.1
        ImpactScore = 5.2
```

**Listing 3.** Textual representation of an Attack Tree.

linked using more complex logical gates, such as AND or PAND gates wherein an attacker must exploit vulnerability A before vulnerability B. Each `AttackStep` also contains CVE's `description`, `CVSS` vector, `BaseScore`, `ImpactScore` and `ExploitabilityScore`.

### 4.4. Fault Tree model

**Summary:** The FT model is used to capture possible faults that can endanger humans or harm the safe operation of the system. Our proposed tool pipeline uses FTs to generate AFTs automatically at runtime. Therefore, it is required to indicate which events of a FT can also be caused by malicious actions. For example, in terms of the FT in Fig. 3, the channel that sends messages to control the trajectory of the drone is more likely to be attacked than the rotor of a drone.

To specify events that can also be caused by malicious actions, we propose an extension of the FT notation with attack events. An attack event describes the event caused by a malicious action and defines conditions that an attack needs to fulfill to trigger the respective attack event. We use the notation of CVSS vectors to specify the conditions for an attack. Each attack event also references either a model element from the Dataflow model or the Deployment model to specify the context of an attack.

Fig. 7 shows an example of a FT that is extended with an attack event (house shape). This attack event specifies that an attack that leads to loss of messages needs to target *ros_nodes*. This is a reference to the respective (abstract) model element in the Deployment model that represents the platform encompassing all ROS nodes in the system. Based on the definition of CVSS vectors, the attack event specifies that

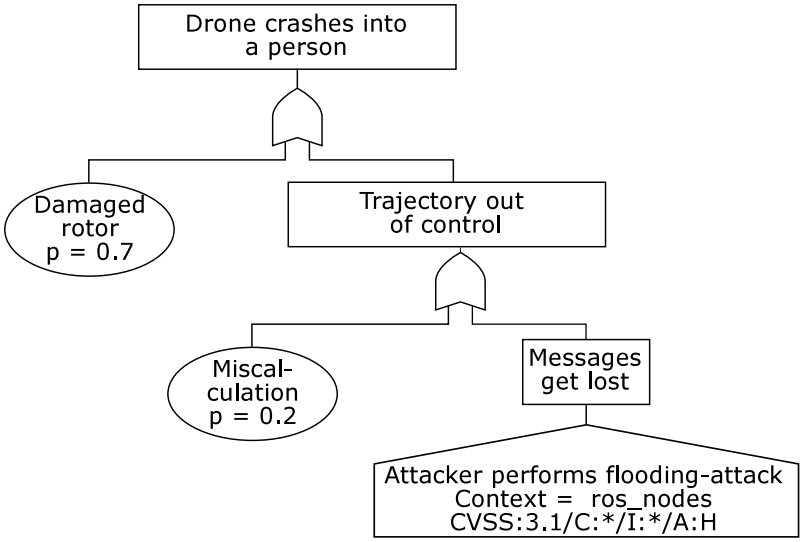



**Fig. 7.** Extended Fault Tree including an attack step (house-shaped node). This reflects that some faults (i.e., message loss) can result from intentional attacks and not just accidental failures.

```
Hazard description = "Drone crashes into a person"{
  OR{
    BasicEvent description="Damaged rotor" probability=0.7,
    IntermediateEvent description="Trajectory out of control"{
      OR{
        BasicEvent description = "Miscalculation"
         probability=0.2,
        IntermediateEvent description = "Messages get lost"{
          AttackEvent description="Attacker performs flooding-
                                   attack"
          deploymentElement=ros_nodes
          CVSSREQ=CVSS:3.1/C:*/I:*/A:H
        }
      }
    }
  }
}
```

**Listing 4.** Textual representation of the Fault Tree in Fig. 7.

an attack needs to have a high impact on the availability (A:H). Other properties, such as confidentiality (C) or integrity (I) are not significant for this particular attack type. Thus, we assign them * meaning “don’t care” in our extension.
**Obtained:** Manually crafted by a safety expert.

**Textual Representation:** Listing 4 shows an example of the textual representation we designed to denote our extended notation of FTs. We define for each event type a corresponding keyword (`Hazard`, `BasicEvent`, `IntermediateEvent`, `AttackEvent`) followed by a description of the respective event and its probability. Since attack events need to specify the preconditions an attack needs to fulfill to trigger the corresponding event, our textual representation also provides keywords to define the requirements for an attack using the notation of CVSS vectors (`CVSSREQ`) as well as to specify the context of an attack. We define terminal rules to enforce the correct use of the notation of CVSS vectors. The context of an attack can either be defined as a reference to a model element in the Dataflow model, using the keyword `dataflowElement`, or as a reference to a model element in the Deployment model, using the keyword `deploymentElement`.

```
Hazard description = "Drone crashes into a person"{
  OR{
    BasicEvent description="Damaged rotor" probability=0.7,
    IntermediateEvent description="Trajectory out of control"{
      OR{
        BasicEvent description="Miscalculation" probability=0.2,
        IntermediateEvent description="Messages get lost"{
          IntermediateEvent description ="Perform flooding-attack"{
            AND{
              AttackStep description="Identify
                          receiving components" probability=0.1,
              AttackStep description="Generate traffic"
                          probability=0.8
            }
          }
        }
      }
    }
  }
}
```

**Listing 5.** Textual representation of the Attack-Fault Tree in Fig. 4.

### 4.5. Attack-Fault Tree model

**Summary:** The AFT model combines safety and security aspects. We use AFTs as an intermediate representation, which we transform into DFTs (see Section 4.6). The transformed AFT is then used as input for the Storm model checker to analyze the occurrence probability of the root hazard. We describe our approach for attaching (ATs) to FTs to create AFTs in Section 5.1.
**Obtained:** Automatically generated by extending a FT by attaching (ATs) based on data from the Dataflow and Deployment models.
**Textual Representation:** Since the AFT model is a combination of the AT model and the FT model, our textual representation is just a combination of textual representations of these two models. Listing 5 shows an example of our textual representation of the AFT that is portrayed in Fig. 4.

### 4.6. Dynamic Fault Trees

**Summary:** DFTs were first introduced in 1992 by Dugan et al. (1992). They extend FTs by additional gates such as functional dependency (FDEP), priority AND (PAND), and simultaneous AND (SAND) to capture dynamic behavior, which in turn are reused in AFTs (see Section 2.2.2). A DFT can be converted into a Markov model to calculate the overall failure rate for given failure rates of basic events.

Since AFTs can be considered as DFTs, which are extended by attaching (ATs) with the help of PAND gates (Budde et al., 2021), we follow this notion and transform AFTs into DFTs using an exogenous model transformation (Mens and Van Gorp, 2006). Transforming AFTs into another modeling formalism is a common analysis approach and is usually done by defining the translation of individual AFT elements into their counterparts in the goal-formalism, as mentioned in other related work (André et al., 2019; Kumar and Stoelinga, 2017; Ponsard et al., 2020).

We translate AFTs to Galileo, a common language for DFTs (Coppit and Sullivan, 2000) which can be understood by a plethora of tools (see Section 5.2 for more details). The resulting DFT can be analyzed e. g. by a continuous-time Markov chain (CTMC) model checker such as Storm,[15] resulting in a probability value representing the MTTF of the top node of the original FT.

Although the syntax is very similar between FTs, AFTs, and DFTs, there are some slight differences regarding their semantics (Budde et al., 2021). Additionally, DFTs themselves exist in different dialects, each with its own subtle semantic variations (Junges et al., 2016). Nevertheless, we decided to use DFTs as our input model for a model checker instead of e. g. stochastic timed automata (STA) (for a translation of AFTs to STAs see (Kumar and Stoelinga, 2017)) because of a) the straightforward translation, b) the much faster model checking, and c) we are not interested in the absolute resulting value, but in its change over time (see Section 5.2).
**Obtained:** Automatically derived from AFTs by an exogenous model transformation.
**Textual Representation:** Listing 6 shows an excerpt of the generated DFT derived from the AFT in Listing 5.

[15] https://www.stormchecker.org/

```
toplevel Drone_crashes_into_a_person;
Drone_crashes_into_a_person or
    Trajectory_out_of_control Damaged_rotor;
Damaged_rotor lambda=0.7;
Trajectory_out_of_control or
    Miscalculation Messages_get_lost;
Miscalculation lambda=0.2;
Messages_get_lost or
    Poor_connection Too_many_incoming_messages;
Too_many_incoming_messages or
    Poor_configuration_of_a_node
    Attacker_performs_flooding_attack_on_trajectory_assignment_topic;
Attacker_performs_flooding_attack_on_trajectory_assignment_topic and
    Determine_communication_mechanism
    Position_in_between_the_target
    Attacker_performs_denial_of_service_attack;
Attacker_performs_flooding_attack_on_trajectory_assignment_topic_SEQ seq
    Determine_communication_mechanism
    Position_in_between_the_target
    Attacker_performs_denial_of_service_attack;
...
```

**Listing 6.** Shortened excerpt of the dynamic fault tree generated for our running example.

## 5. Running the SAFT-GT pipeline

After introducing the different models, we take a closer look at the "heart" of our approach. We explain in detail, how the combination of FTs and (ATs) into AFTs work and how the resulting AFT is translated into a DFT, the input model of a PMC such as Storm.

As shown by the icons in the lower right corner in Fig. 5, our approach welcomes manual additions. All automatically created or derived models can be adapted or supplemented manually at any time. This ensures that existing models can be reused.

### 5.1. Combination of the different models

In this section, we describe how we attach generated (ATs) to manually crafted FTs based on context information provided by the Dataflow and Deployment model. This combination process is shown in step 5. "Fragment Combination Model Transformation" in Fig. 5.

In contrast to our AT example in Fig. 2, the generated (ATs) are at a different level of abstraction (see Listing 3). Thus, there is also a difference in the abstraction level between manually created FTs and generated (ATs). The events in our FT describe — on a rather abstract level — the causes for a hazard without considering the concrete implementation details. However, the generated AT describes specific vulnerabilities of the used components. For example, the AT in Listing 3 is based on a vulnerability that can be exploited to perform a flooding attack. Nonetheless, the necessary intermediate steps to achieve this are missing if we just attach the generated AT to the FT. To solve this problem, we define AFT fragments, which describe the necessary intermediate steps to perform attacks. These AFT fragments are manually defined by the authors with modeling security expertise and are based on attack descriptions from the MITRE ATT&CK framework. The already defined AFT fragments are part of the pipeline and can therefore be reused by all its users. The AFT fragments are attached to the AFT before we attach the generated (ATs) to mitigate the different abstraction levels between FTs and generated (ATs). To accomplish this, we divided the AFT generation process we developed into three phases.

In the first phase, the manually crafted FT model, which is provided as input for the generation process, is transformed into an AFT model. This transformation is a simple horizontal, exogenous transformation (Mens and Van Gorp, 2006) (since the FT represents the top of our AFT), as illustrated in Section 4.5. It should be noted that we extend the FT notation to include so-called Attack Events (cf. Section 4.5). These Attack Events are also added to the AFT and form the possible attachment points for attacks in the following phases.

The second phase deals with different abstraction levels between FT and generated (ATs). For this purpose, we have defined AFT fragments which describe the necessary intermediate steps to perform common attacks. This allows us to break a more abstract attack scenario, e.g., a flooding attack, into multiple atomic steps, such as "corrupting a platform". Fig. 8 shows an example of AFT fragments.

For each AFT fragment, we define conditions that are checked against the requirements defined within each Attack Event in the AFT from phase one. These provided conditions consist of two parts, namely the context and the attack property. The context describes what type of

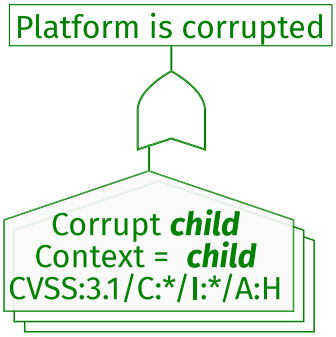



**Fig. 8.** AFT fragment modeling the corruption of a platform by corrupting its components.

model element is affected by the attack. For example, an Attack Event must reference a model element from the Deployment model of the type `Platform`.

The second provided condition represents the property of an attack and this is defined for each AFT fragment. To express this property, we use the notation used by CVSS vectors. Since the entries in CVSS vectors are on the nominal and ordinal scale, we developed a comparison mechanism to check that the provided conditions of an AFT fragment fulfill the requirements of an Attack Event. Accordingly, we compare values such as Attack Vector (AV), User Interaction (UI), or Remediation Level (RL) that lie on a nominal scale for equality. In regard to the values that lie on an ordinal scale, we compare them with "greater-than-equal". This means, for example, that if an Attack Event defines that Confidentiality Requirement (CR) should have the value Medium (M), the provided conditions of the AFT fragment must define that CR is Medium (M) or High (H). For the AFT fragment in Fig. 8, we defined that Availability (A) must be High (H). This means that this AFT fragment can only replace Attack Events in the AFT that reference the respective context and define A = H. If this fragment can be applied, it creates a new Attack Event for each component that is part of the previously referenced platform, describing that it is being corrupted (represented in Fig. 8 by the overlapping house shapes and ***child*** as a placeholder).

Regarding our running example, this means that after the FT in Fig. 7 was transformed into an AFT in the first phase, the Attack Event "Attacker performs flooding-attack" remains. In the second phase, our AFT generation approach checks whether there are AFT fragments whose preconditions fulfill the requirements defined in the Attack Event. This is the case with the AFT fragment in Fig. 8. Thus, the Attack Event is replaced with the fragment, and for each component that is part of *rose_nodes*, a new Attack Event is generated and connected to the intermediate event "Platform is corrupted" by an OR gate.

The third phase deals with attaching the generated (ATs) to the AFT resulting from the previous two phases. For this purpose, the referenced context is considered for each Attack Event. More precisely, each model element from the Deployment model referenced in the AFT is recursively scanned to determine whether it has a vulnerability or consists of/uses a component with a vulnerability. If a Dataflow element is referenced by an Attack Event, the corresponding model element in the Deployment model is identified before the recursive analysis. Consequently, when a system component with a vulnerability has been identified, we re-use the mechanism described in phase two to ensure that the requirements described using the CVSS notation are also served by the discovered vulnerability. This leads to the corresponding Attack Event in the AFT being replaced by an AT. If an Attack Event can be replaced by several (ATs), we combine them using OR gates to attach them to the AFT.

In our example, the generated AT in Listing 3 fulfills the conditions of the attack event that was generated as part of the AFT fragment in Fig. 8 for the Dataflow model element *simple_trajectory_server*. As a result, this Attack Event is converted into an intermediate event and the generated AT is attached to it. Thus, we obtain the AFT shown in Fig. 9.

### 5.2. Risk calculation

The resulting AFT has to be analyzed via a PMC returning the MTTF of the current system configuration. Thus, each AFT has to be transformed into an appropriate input language of a model checker. Stöelinga et al. present in Kumar and Stoelinga (2017) a translation of AFTs to stochastic timed automata (STA) (David et al., 2011). For model checking of this formalism, the UPPAAL Stochastic Model Checker (SMC) (David et al., 2015) can be used. Unfortunately, UPPAAL SMC only allows simulation and no closed-form solution. This means that the results depend on the number of simulated cycles and can vary due to randomness. For reliable results, 100,000 or more cycles are necessary and require corresponding significant computation times.

Instead, we decided to extend the toolchain with a closed-form solution provided by a continuous-time Markov chain (CTMC) model checker such as Storm. Although there are a couple of similarities and differences in the sense of syntax and semantics between FTs and (ATs), we follow the conclusion from Budde et al. (2021) that "AFTs trade scalability for versatility, by merging DFTs (with all its dynamic gates) with (ATs) plus PAND gates." DFTs in Storm already supports exponential probability distribution as well as OR and AND gates. PAND semantics can be simulated by combining an AND gate with a SEQ gate with the same children.

Regarding the probability of a successful attack, we assume an exponential distribution that reaches the given EPSS value after 30 days (Jacobs et al., 2021). Thus, the obtained EPSS scores of the Basic Attack Events are translated into a rate $\lambda$ according to the following formula:

$$\lambda = -\frac{\ln(1 - \text{epss})}{30 \cdot 24 \cdot 60 \cdot 60s} \tag{1}$$

For specific attacks modeled in manually created (ATs), this value has to be provided by the modeler.

The resulting DFTs are then passed to the Storm model checker to calculate the MTTF (according to Equation 1 in seconds) of the top event of each original FT. An MTTF indicates how long it takes in average until the top event fails. Estimation of the various probabilities is an extremely difficult task, even when the EPSS scores are taken into account. Due to this uncertainty, we suggest not considering the absolute MTTF but its change over time to get insights, if the overall risk (of this particular) hazard in- or decreased, e. g. due to newly discovered CVEs.[16] If the value decreases, a more secure and thus also more reliable state has been reached. Finally, the resulting MTTF is fed back to the MAPE-K system in order to integrate the new information into new adaptations, if necessary.

**Interpretation of probabilities.** The estimation of attack and failure probabilities remains challenging computationally and conceptually. Prior works do not provide consensus on how to meaningfully combine stochastic failures with deliberate attacks. Our interpretation follows the line of work by Fovino et al. (2009) and Kumar and Stoelinga (2017), who demonstrated that treating attack events as probabilistic basic events within AFTs is methodologically sound when the probabilities are grounded in empirical exploit likelihoods (e.g., EPSS/CVSS). Accordingly, the MTTF values in our analysis should not be interpreted as absolute predictions of joint safety-security risk.[17] On contrary, they should be taken as relative indicators used to compare configurations and runtime adaptation decisions within the same system context.

[16] We acknowledge that this transformation assumes a monotonic increase in exploit likelihood, which may not hold for all vulnerabilities. Thus, the resulting rates should be interpreted as approximations.

[17] For instance, cybersecurity outcomes can be strongly influenced by organizational context and human factors, which can introduce variability that is not reflected in purely probabilistic or data-driven models (Pfister et al., 2025).

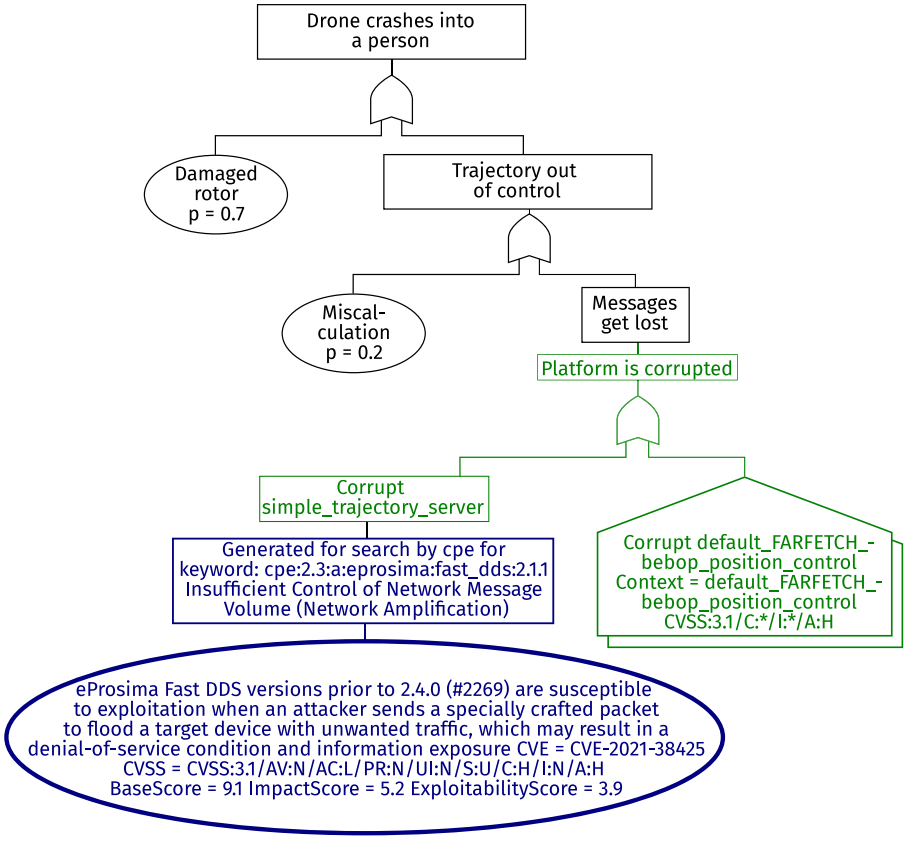



**Fig. 9.** Simplified excerpt from the AFT generated by our approach.

### 5.3. Closing the loop

We integrate the toolchain into the MAPE-K loop in a way that allows us to trigger each step separately. Together with the possibility to reuse already created artifacts, this allows for an incremental approach depending on the changes that occurred in the overall system. In case of a reconfiguration, the dataflow within the analyzed system might have changed and thus, a complete rerun is necessary. This means that the dependencies might have changed with newly activated nodes, which can include new components and libraries. For this reason, it is necessary to identify new vulnerabilities that would affect these new nodes and these need to be discovered. In the case of a system update (or migration of particular ROS2 nodes to another underlying system), it is enough to update the dependencies of the deployed components and thus, the first two steps can be skipped. Finally, in cases when the public CVE database is updated — for example, once a day — a new AT generation can be triggered after this update. In regard to the EPSS values, the EPSS database is updated only once a year.[18]

Since the execution of the different parts of the pipeline might take a certain amount of time (see Section 6.3), the ROS2 service call just validates the input parameters and returns immediately. The resulting MTTF value is sent asynchronously via a separate ROS topic. This enables the use of multiple receivers and can also be used to signal to the MAPE-K system that a triggered toolchain run has finished.

## 6. Evaluation

To determine the overall approach's perceived relevance and applicability, we conducted a study where subject-matter experts of varying acumen were asked to provide their opinions on the developed approach. In this regard, a survey and a comprehensive discussion with the participants were conducted. Moreover, we provide an experimental evaluation of the proposed approach. This was also done because we did not provide any evaluation of the presented approach in our previous works due to the large scope and the considerable effort that was required.

### 6.1. User study design

The study was designed as a part of a two-hour workshop, which consisted of two surveys, three presentations (*SAFT-GT Overview and Agenda*, *Input Models and Model Generation*, and *AFT Model Combination and Analysis*), and three discussions (*SASs: Experiences and Challenges*, *Safety and Security Models and Modeling*, and *SAFT Toolchain and Model Combination*). Each presentation was followed by a discussion, which targeted that particular presentation. The workshop was recorded and was agreed to be recorded by all the participants. All participants were formally informed and provided written consent. This included consent for data collection, processing, anonymization and eventual publication of their anonymized responses. Consent also explicitly covered the linkage between discussion data and surveys and participants were assured of their right to withdraw at any time. In order to promote transparency and reproducibility of the user study, we release transcribed discussions, linked survey questions and consent form in our open-source repository (see Section 1) to support independent verification and replication efforts.

**Participants.** As a part of the recruitment process, we reached out to multiple experts, each having expertise in one or more fields that tackle the context of our developed approach. They were asked if they were willing to partake in a workshop by providing them with a short description of our research project. We conducted all communication via

[18] https://www.first.org/epss/model

Email. In total, seven experts gave a positive response and they were provided with a link to a survey[19] in which we anonymously collected some background information. This was done one week before the workshop. Each of the seven experts was from the research domain. In addition, some of them also had additional backgrounds as research and development consultants in the industry domain. They all had at least 3–5 years of experience in safety modeling and engineering of cyber-physical systems (three of them had 10+ years of experience). In regards to security modeling, the experience ranged from 1 to 9 years. Moreover, all the experts had knowledge and experience in the following domains: domain-specific languages, self-adaptive systems, and FTs. However, the participants had slightly less expertise concerning security topics and domains utilized in our proposed extended approach. This relates to CVEs, CVSS, CWEs, EPSS, (ATs), and AFTs. Nonetheless, this is the case with only one or two participants. Thus, it can be concluded that the safety background of experts was on the stronger side compared to their security background.

**Survey.** Two surveys were conducted during the workshop. One survey focused on the applicability of the proposed extended approach, while the other one targeted its relevance. Every question in the surveys was answered by all the participants of the workshop. The answers to each of the questions were provided in accordance with the Likert scale: *very (applicable/relevant)*, *(applicable/relevant)*, *neutral*, *not (applicable/relevant)*, and *not (applicable/relevant) at all* for applicability and relevance respectively. We hosted surveys on unpublished websites, and we never asked for participants' sensitive or personal information.

**Expert opinions.** Expert opinions were collected at three different intervals during the workshop. This was done after each presentation. The discussions took 10, 20, and 20 minutes, respectively. The first discussion targeted various challenges and experiences with SASs. This provided us with insights regarding how experienced the experts were in this particular topic. In the second discussion, the relationship between safety and security models, as well as their combination, was addressed. Finally, the last discussion examined the approach that we extended, including the full toolchain and model combinations. During this process, we adhered to the relevant empirical standards for qualitative studies as outlined in the SIGSOFT Empirical Standards.[20]

### 6.2. Results

**Survey.** Each of the seven participants provided answers to all of the questions from both surveys. Fig. 10a and b present the results regarding the applicability[21] and relevance of the developed approach respectively.[22] The findings revealed that in Q1, 29% of experts rated the approach as applicable, while 71% were neutral. In Q2, 14% found the approach very applicable, 57% rated it as applicable, and 29% were neutral. Furthermore, the overall applicability of the approach received ratings of 14% applicable and 86% neutral in Q3.

In terms of relevance, 14% found the data and deployment model very relevant, 71% rated it as relevant, and 15% were neutral in Q4. For the relevance of the presented CVE mining and AT generation approach, 14% found it very relevant, 71% rated it as relevant, and 15% were neutral in Q5. Moreover, regarding the relevance of the AFT combination in reality, 14% found it very relevant, 14% rated it as relevant, and 72% were neutral in Q6. Lastly, the relevance of the overall approach was rated as 28% very relevant, 58% relevant, and 14% neutral in Q7. These responses provide valuable insights into the perceived applicability and relevance of the approach, indicating areas for further consideration and potential improvement, which will be covered in Section 7.

**Expert opinions.** After carrying out the discussions, we qualitatively analyzed the recording of the workshops (we cannot share the recording due to NDA). The aspects covered can be organized into the following five topics:

- **Design vs. Runtime Approach**: The question of "*why the approach was not implemented at design time*" was raised by one of the experts. However, the other expert stated: "*There will be some unexpected conditions. There exist many states that cannot be foreseen at design time, which can be recorded quite well by monitoring the system runtime. These states of the system can actually be recorded at runtime, including any unexpected conditions. There is also the case that not everything is known at design time. Perhaps new things will be added.*" According to the aforementioned statements, the design time approach was found to be challenging due to unexpected conditions and the necessity for all components to be known in advance. Moreover, the possibility of adding new components to the system that cannot be predicted at design time was acknowledged. This also includes the possible space exploration in regards to the system and its environment, which may lead to a state space explosion. Therefore, addressing these concerns from the design time perspective may be infeasible and experts agreed that the runtime perspective was appropriate, considering the dynamic nature of system components and conditions.

- **Closed vs. Open System**: The expert stated: "*Simply because there are a lot of systems that include networking, you have an incredibly large attack surface because of the networking itself. But once the system is closed, at least physically, and there is no flow of information to the outside and only the sensors are the attack surface.*" Thus, the proposed approach was deemed suitable for open systems, as it accounts for the networking aspect, resulting in a larger attack surface. In contrast, for closed systems where the networking aspect is absent, the focus would need to be shifted to attacks occurring over sensors.

- **Trusting Experts vs. Probabilities**: Another expert noted the following: "*There are many successful attacks that occur in everyday life that are not recorded, which require significant expert knowledge in order to be addressed. Do we then want to rely on these calculated figures? I would be skeptical at first.*" Based on the aforementioned statement, it was evident that the implementation raised concerns regarding trust in automated evaluations versus manual expert assessments. Despite the prevailing tendency to trust manual evaluations more, the increasing complexity of systems necessitates the adoption of automated approaches. The implementation supports the need for continuous system checks due to the inherent possibility of unforeseen events, reinforcing the importance of automated evaluations. Nevertheless, the proposed approach includes the possibility of experts adding additional states manually, such as zero-day attacks.

- **Modularity**: During the workshop, the following was mentioned: "*Several data models that reference different components can be simply plugged together, which at the end form a complete system. For example, even different competitors can share model fragments. However, then it is not very clear at which point they should probably share something.*" Thus, the approach's ability to generate AFTs for specific parts or components of a system was emphasized. This modularity facilitates the sharing of AFT fragments among industry competitors for specific system parts, reducing the cost of evaluating larger systems by allowing fragments to be plugged into other AFTs, which makes the evaluation of larger systems less costly.

[19] This survey is independent of the two that were conducted during the workshop.

[20] https://www2.sigsoft.org/EmpiricalStandards/docs/standards?standard=QualitativeSurveys

[21] The applicability survey has only three questions because it was executed before the AFT combination topic was covered in the workshop.

[22] The percentage values were rounded so they may not sum up to 100%.

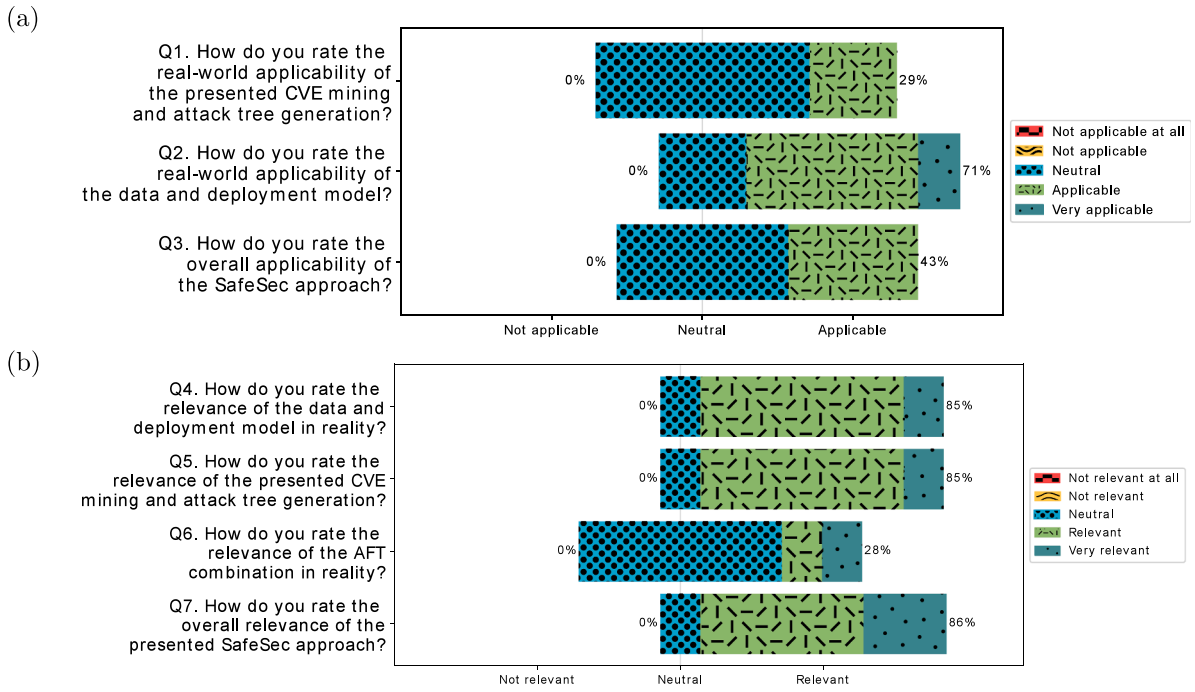



**Fig. 10.** Workshop Survey Response Results.

- **Variety of Security Attacks**: One of the experts mentioned: "*But the first that came to my mind was, well, if you look at the functional safety of systems and if you can significantly change the system in operation…This refers specifically to SASs, then you open up a whole additional world of possibilities and attacks that can influence the system's behavior considerably.*" Another expert said: "*If there is an attack that is not publicly recorded and I would like to add this attack manually, this does not bring too many advantages. This is because a lot of things need to be done manually by creating or updating separate models. However, these could theoretically be used again and inserted into this automated setup here. I see that models are theoretically integrable in this modeling approach and that automation depends a lot on the data flow model. Also, if, for example, in the hardware I have somehow a hardware component such as an electric motor and there are magnetic beams that lead to bit flips that target the RAM memory…Then that is not about the data flow model and requires additional consideration, which requires the further extension of the proposed approach.*" Based on the aforementioned statements, it can be noted that the experts mentioned it is necessary to consider attacks when the system adapts to new configurations, which highlights the potential of targeting reconfiguration mechanisms by attackers. In addition, attacks targeting adaptation mechanisms need to be addressed. While the approach relies on public security databases, it allows for the manual addition of other types of attacks and emphasizes the necessity to consider mechanical attacks. In cases involving mechanical attacks (such as using beams to create bit flips), traditional models like the data-flow model may not be as relevant, necessitating a broader consideration of attack vectors.

**Disclaimer:** The statements above stem from our own interpretation of the (sparse and unstructured) discussions. However, they reflect the opinion of experts involved in the workshop.

### 6.3. Performance

The following opinion was stated by an expert: "*Industrial experts, who are in this particular field of expertise, would need some assurances regarding if the developed modeling approach and generation of specific models is too expensive. In my previous experiences, which specifically relate to hazard and risk analysis, it is important to know and measure the costs of running such implementations or toolchain.*" We agree with this statement, especially because our previous work (Groner et al., 2023) does not contain such evaluation.

In general, it is not easy to provide a holistic experimental evaluation of this approach, as it strongly depends on many parameters: the size of the models, the amount and size of FTs, manually added (ATs), vulnerabilities found in a system, to name but a few. Nevertheless, we set up the ROS2 quadcopter example mentioned above in our quadcopter lab and measured the elapsed time for each of the steps in the integrated toolchain. This was done to determine which steps are expensive and if the toolchain could scale.

In our setup, the entire ROS2 part (managed system, MAPE-K loop) ran on a different machine than the toolchain (both i7-3770 @ 3.40GHz, 16GB RAM). The dataflow model of our ROS-based example consists of 21 nodes that are connected via 43 ROS topics. We created 3 FTs manually with 13 nodes/4 gates (*battery*), 11 nodes/6 gates (*spying*), and 33 nodes/9 gates (*injure*). The deployment model generator extracted 265 dependencies on the machine running both the managed system and the MAPE-K loop.

Our pipeline mined a local copy of the NVD database with around 272,400 entries and identified 77 CVEs affecting system parts connected to causing the hazard modeled by our FT. Based on these CVEs, our pipeline generated 5 (ATs) by matching CPEs ($2\times1$ CVE, $1\times8$ CVEs, $1\times17$ CVEs, $1\times18$ CVEs). Another 6 (ATs) were generated by full text search in the CVE's description, which is more likely to lead to false positives ($2\times1$ CVE, $2\times2$ CVEs, $1\times3$ CVEs, $1\times23$ CVEs).

The AFTs combined from the FTs and the amount of (ATs) has three different sizes: The smallest (battery) has 38 nodes and 14 gates, the mid-sized (spying) has 926 nodes and 461 gates. Finally, the largest AFT (injure) has 1845 nodes and 814 gates. The DFTs generated from the resulting AFTs and served as input for the Storm model checker have

**Table 1**
Results of Running Example Model Generation and Evaluation Experiment.

| | Model Generation Time | | | | | |
|---|---|---|---|---|---|---|
| [s] | Datafl. | Deploy. | AT | | | |
| avg | 4.979 | 273.925 | 552.596 | | | |
| sd | 1.022 | 0.743 | 1.457 | | | |
| | **Model Generation/Analysis Time** | | | | | |
| | battery | | spying | | injure | |
| **[s]** | **AFT** | **DFT + MC** | **AFT** | **DFT + MC** | **AFT** | **DFT + MC** |
| avg | 10.168 | 3.124 | 17.619 | 3.042 | 21.708 | 3.650 |
| sd | 1.075 | 0.270 | 1.906 | 0.088 | 0.936 | 0.127 |

between 26 nodes and 14 gates up to 990 nodes with 918 gates. This difference arises because some nodes can be combined and some additional intermediate nodes have to be added during the conversion.

Table 1 shows the average result of 100 runs for each step. We focus on the execution time of the toolchain and not on the duration of the reconfiguration process. This is due to the fact that the duration of the reconfiguration of the system strictly depends on the implementation of the reconfiguration mechanism, which is out of the scope of this paper. The entire toolchain can be executed in about 15 minutes (~860s), wherein the most time is needed for the full text search of the database (~10min) followed by the extraction of the package dependencies (~5min). Since repeating these steps is only necessary if the system changes (e.g., by removing or adding ROS nodes, updating the underlying system (and thus the dependencies), or adding new CVEs), we did not repeat these steps 100 times for each FT. In contrast, the remaining steps (AFT generation and model checking (MC)) depend on the FT and were therefore executed separately for each FT.

In the following, we will discuss the factors influencing the size of the generated AFTs, as these are used as input for the model checker and can therefore have a significant influence on performance. The size of an AFT depends on several factors. Its size has a limited relationship with the system size and is independent of the number of hazards considered. If several hazards are considered, this results in a forest of FTs that leads to a forest of AFTs. These AFTs must then each be examined with the aid of a model checker, whereby it can be assumed that not all modeled hazards are relevant in all possible system configurations and every environment. The analysis of a forest of AFTs, for example, by optimizing parallelization, is out of scope for the current state of work but should be considered as a future research direction. As mentioned above, the system size has a secondary role, since the pure number of CVEs in a system does not influence the size of an AFT. As described in Section 5.1, AFT fragments and (ATs) are attached based on the preconditions defined as Attack Events in our proposed extension of FTs (cf. Section 4.4). This means that the number of Attack Events and their fulfillment influences the size of an AFT.

It should be noted that an Attack Event can be replaced by several AFT fragments linked by an OR gate. However, it is never possible to use all currently defined AFT fragments, as some of them are mutually exclusive. For example, no component can be a channel and a component, so the context defined in the Attack Event already limits the number of possible AFT fragments. As can be seen in Fig. 8, AFT fragments in fact also define Attack Events. A distinction can be made as to whether the appending of an AFT fragment leads to several new Attack Events, as in our running example (cf. Section 5.1). Usually, AFT fragments only define one or two Attack Events independently of the system. For example, our AFT fragment describing an adversary in the middle attack defines an Attack Event that describes the determination of the communications protocol used.

The number of attached (ATs) depends on whether Attack Events reference components as a context that are related to the CVEs contained in the system. Since we perform an elaborate analysis of the textual description of CVEs, our generated (ATs) consist of some more nodes and thus have influence on the size of the generated AFT. Although this additional full text search may lead to more false positives, we ignore this circumstance in this case study to demonstrate that our tool chain can also handle (rather unrealistic) large models.

Finally, we acknowledge that model checking offers a potential performance disadvantage. However, our performance measurements show that the biggest performance bottleneck in our pipeline is currently the extraction of dependencies (resp., the full text database search). Since this is part of our implementation, it makes sense to focus on optimizing this particular aspect first before focusing on the used third-party components.

## 7. Future research and limitations

In this section, we discuss future research directions and limitations of the proposed approach.

**Future research directions.** Considering the discussion with experts, future research should consider several directions. The following key points provide a roadmap for future research directions:

- **Automation as the Key Focus**: Emphasize a strong focus on automating as much of the approach as possible, including specific models that are part of the approach. This is in stark contrast to manual investigation of security incidents in self-adaptive systems, particularly those leading to safety and reliability issues. Automation not only enhances efficiency but also ensures accuracy, thereby mitigating the risks associated with manual investigation. The expert mentioned the following: "*I am really happy when people at least think about the fact that something can go wrong, and that is what I actually like very much about this approach. What also helps to a high degree is the automation, i.e., recognizing things as soon as possible and even if they are not perfect, but at least automatically recognizing them is a big plus compared to manually filling many forms and tables, and then you have to do all the calculations.*"
- **Behavioral Models for System States**: Focus on developing behavioral models to capture different system states and states of the monitored environment. This can lead to the identification of state-specific AFTs, thereby bolstering the system's resilience against potential threats. Behavioral models can provide a deeper understanding of system states, enabling proactive measures to be taken in response to potential security threats, which was mentioned by an expert: "*You may also consider the use of behavioral models and these usually closely relate to SASs because the system depends a lot on the state of the environment and specific conditions. This is especially significant for Fault Trees.*"
  In a similar direction, future work could also explore the integration of attacker behavioral profiles into the analysis. While the current work focuses on observable vulnerabilities and structural system properties to achieve automation and scalability, attacker profiles (e.g., motivations, capabilities and behavior patterns) could potentially provide a more targeted security assessments. However, such modeling would require well-defined abstractions (e.g., attacker categories or skill levels) and the ability to translate these into probabilistic model parameters. Frameworks such as MITRE ATT&CK could offer a foundation for including these aspects in a structured and extensible way. However, due to their variability and context dependency, attacker profiles are challenging to formalize and thus remain an open research direction. Additionally, the integration of a formal specification of the behavior of each node would allow a deeper analysis of the models. For example, one could define formal properties (e.g., in temporal logic) and use the probabilistic model checker to prove that these apply to the system.
- **Consideration of Non-Conventional Attacks**: Extend the scope of research to consider other types of attacks that may not be readily available in public databases. The expert stated: "*If you do not have*

*any components in your system that are related to specific CVEs yet, you do not generate any Attack Trees, or do you? Systems are becoming more complex and thus offer many possibilities for adversaries and not all vulnerabilities that they exploit are reported.*" This means that mechanical attacks and those targeting self-adaptation mechanisms also need to be considered in the future. However, these attacks (specifically mechanical attacks) necessitate novel approaches to security assessment and mitigation that would benefit the system's reliability. In addition, zero-day attacks (Bilge and Dumitraş, 2012) can be considered an unforeseen menace due to the fact that they exploit previously unknown vulnerabilities, for which no specific defense was prepared or implemented. Nonetheless, these represent a serious threat not only to self-adaptive systems but to any other system too. A possible direction that could tackle this issue is a utilization of NLP (natural language processing) techniques and Large Language Models (LLMs) to conduct an analysis of social networks and underground forums (Arazzi et al., 2023) — a place where adversaries communicate freely. These commonly include hints as well as detailed information on planned and ongoing attacks, including tactics and techniques. By considering a wider range of potential attacks, the security measures can be more comprehensive and adaptive, ensuring robust protection against diverse threats.

- **Development at Design Time**[23]: Shift the focus towards developing the safety and security approach at design time, with an emphasis on predicting possible threats and fortifying the system's safety and security from the outset. The following was mentioned in the workshop: "*There are actually things that can be planned in advance or perhaps even have to be planned for the design side. However, you would theoretically have to consider all the faults for every configuration and every transition, so to speak. In case the adaptation is implemented in such a way that this configuration space is not even known for the designs, this becomes extremely difficult, but the design-time development can serve as a starting point.*" Thus, it can be said that by addressing security concerns at design time, significant resources can be saved at runtime, especially in scenarios where the modeling approach becomes expensive due to the system's scale and complexity. However, it is clear that all the security and safety-related issues cannot be simply resolved at the design time due to the unpredictable nature of CPSs — and specifically in the case of SASs. Optimally, a combination of both the design and runtime approaches, such as the one described by Fournaris et al. (2019), should be utilized. As a result, this would lead to a better overall reliability of a system.

**Limitations.** The limitation of the presented toolset is that the requests are sent to the remote computer via SSH. This could probably be optimized by combining several requests and not opening a new SSH connection for each request to the remote computer. In addition, during the performance evaluation, it can be seen that the standard deviation in the generation of (ATs) is relatively high. This can be explained as follows: When searching for CVEs, CPEs are used in the first instance and, if no CPEs can be determined for a component, a full-text search is performed on the description of the CVE database on the other. The toolset uses a REST API from NIST for the CPE search, as this automatically takes the version number ranges into account. The response time of this network query varies considerably depending on the load on the server. In contrast, the full-text search on a local copy of the database is considerably faster. In our example, this only accounts for around 3s of the 62s. Obviously, one could also optimize this by using the local database for the CPE search and re-implementing the check of the matching version numbers of the packages.

We provide a detailed description of our generation approach that relies on these models in Section 5.1. Moreover, while the toolset integrates standard libraries and public data sources (e.g., ROS2 introspection tools, CVE/CPE databases), all core components responsible for model generation, transformation and analysis are self-implemented. These external resources serve only as inputs or utilities. Thus, their potential errors have limited impact on the correctness of the toolchain's internal logic.

In regard to the user study, the recruitment of experts who possess experience in various domains that are part of our approach proved very challenging due to their scarcity. Thus, we do not attempt to generalize the findings – due to the small number of participants – which prevents us from deriving quantitatively grounded conclusions. This is due to their scarcity, which is common in security-related user studies (Braun et al., 2024) in which the participant size is rarely greater than 20. Moreover, the experts' background has a stronger safety focus compared to the security focus. As a result, it is possible that some security-related aspects and conclusions were overlooked. Finally, the experts could only grasp as much of our approach as it was presented to them during the three presentations at the workshop. This means they did not test the approach in real-world or practical settings. Hence, it is expected that the survey answers they provided are strictly related to the presented content. Nonetheless, according to the discussions conducted, it was safe to assume that they grasped the presented approach thoroughly.

**Limitations of combining safety and security probabilities.** The quantitative integration of safety-related failure probabilities and security-related attack probabilities remains debated in the safety and security-engineering communities. While AFTs provide a unified framework, their semantic foundations depend on assumptions that are not "universally" accepted. Safety failures can be considered random and stochastic, while security attacks are intentional and adaptive (Kriaa et al., 2015; Bloomfield et al., 2013). Several prior works argue that no single probabilistic metric can fully capture both aspects (Kriaa et al., 2015; Bloomfield et al., 2013).

In past, literature has demonstrated that a probabilistic treatment of attack events within fault-tree formalisms is feasible under clear assumptions. Fovino et al. (2009) explicitly concluded that *"it is possible to calculate the probability associated with the attack-tree goal exactly in the same way as it is done for the fault trees"*. Kumar and Stoelinga (2017) later provided quantitative semantics for AFTs and showed their applicability for joint safety-security analyses. Accordingly, we interpret the probability of a fault event as the likelihood that a component fails, and the probability of an attack event as the likelihood that a known vulnerability is successfully exploited in the wild. We treat both as stochastic basic events within the same formalism.

To maintain interpretability, we deliberately restrict our model semantics to a subset of well-understood logical gates (AND, OR, PAND, and SAND). We exclude dynamic or repair gates such as FDEP or SPARE, whose cross-domain semantics remain debated in the community. Logical gates define combinations of events based purely on logical or temporal relationships (conjunction, disjunction, ordered conjunction, and sequential conjunction), which are considered interpretable in both cases when the input events represent random failures and attacks. Limiting the model to the four logical gates ensures semantic transparency, consistency across domains, and computational tractability when transforming AFTs to DFTs for model checking.

Our quantitative assessment does not serve as an absolute predictor of joint safety-security risk. On contrary, it should be considered as a relative indicator for comparing designs or runtime configurations of cyber-physical and self-adaptive systems. Nonetheless, a universally accepted probabilistic framework for combined safety-security aspects is still an open research challenge.

**Limitations of probabilistic modeling based on EPSS.** Our approach approximates the probability of successful exploitation using an exponential distribution derived from EPSS scores. This directly implies a monotonically increasing hazard rate. However, empirical EPSS trajectories are not always monotonic and may fluctuate due to factors such as patch availability or changing attacker incentives. Consequently, this

[23] Development at design time needs to be complemented with the development at runtime.

assumption introduces approximation errors that can affect the accuracy of the resulting MTTF values. This is particularly the case for vulnerabilities with highly dynamic exploitation likelihoods.

To address this, MTTF is not interpreted as an absolute prediction but as a relative indicator for comparing system configurations and guiding adaptation decisions. This reduces sensitivity to modeling inaccuracies by focusing on trends rather than exact probabilities. For cases in which EPSS behavior deviates significantly from exponential assumptions, alternative modeling approaches (e.g., piecewise or time-series-based) may provide improved accuracy and could represent potential directions for future work. This limitation reflects broader challenges related to cybersecurity data, where empirical indicators such as exploit likelihood are inherently uncertain and may not reliably capture real-world attack dynamics (Karaosman et al., 2026).

## 8. Related work

The body of evidence (Pekaric et al., 2023; Oueidat et al., 2020) suggests a growing research interest in the joint analysis of the safety and security of cyber-physical systems, one of which includes self-adaptive systems. This section examines related works as well as their differences to our approach.

### 8.1. Safety and security modeling approaches

Some of the techniques that have been deployed for safety and security modeling include system theory, AFT modeling, and threat and attacker-profile modeling, among others.

**System Theory Analysis.** A foundational work in this regard was done by Young and Leveson (2014) wherein they proposed Systems-Theoretic Accident Model and Processes (STAMP), an accident causality modeling and analysis approach. In their study, security was determined as another property within modern systems. Hence, they argued for the extension of Systems-Theoretic Process Analysis (STPA), hitherto widely used only for safety analysis, to incorporate security analysis and dubbed it STPA-Sec. Further to Young and Leveson's study, Pereira et al. (2017) proposed an integrated approach that combines STPA and NIST SP800-30, a well-known threats, events and vulnerabilities framework, to automatically analyze and detect conflicts between pairwise reinforcements of various safety and security limitations. However, Pereira et al.'s solution considers safety concerns as a control problem rather than a reliability problem, i.e., the ability to manipulate and direct a system's actions, as opposed to the system's likelihood of experiencing unexpected breakdowns or errors. Furthermore, their solution requires that high-potential risks are known at design time. Although the authors claim that the process of identifying main system risks requires little effort, they discount the need for an advanced understanding of these CPSs and their operational contexts.

**AT/FT/AFT Generation Approaches.** Different approaches have been proposed to generate AFT models and are discussed briefly in the ensuing paragraphs. One of the earliest ideas for generating fault- or attack graphs is the one by Swiler et al. (2001) wherein they present a tool that generates attack graphs based on an assessment of security attributes and vulnerabilities in computer networks. The authors use vulnerability scanning tools and attack templates to determine network vulnerabilities and introduce configuration files that are similar to the aforementioned dataflow and deployment models. Furthermore, they mention that the integration of an attacker profile might be interesting for determining attack potential. Our implementation differs from Swiler et al.'s because our approach combines the generated (ATs) with FTs and additionally, we generate dataflow and deployment models.

Fovino et al. introduce the concept of integrating the so-called "attack goal" resulting from multiple layers of (ATs) in FTs. In comparison, Kumar and Stoelinga (2017) also introduce new model elements to combine FTs with (ATs). One of the earliest works in AT generation, Kotenko and Chechulin (2013) generated an AT by using CPEs to identify CVEs for components in which CAPEC metrics are used to generate more complex attack scenarios. However, they focus only on network communication scenarios and thus utilize their network security detection tools to identify possible vulnerabilities. The use of CPEs is similar to our proposed approach, in which AT fragments are generated by mining vulnerabilities from CVE databases. However, we utilize software package-related information from a running (ROS) system for the generation of AT. In contrast, Ou et al. (2006) generated attack graphs for network topologies using logic programming. One disadvantage of their approach is that all information from the system must be provided manually in advance by "facts" in the logical programming language.

**Hybrid Analysis Approaches.** Jablonski et al. (2022) present a top-down formal method approach based on systems theory for creating attack-defense trees of CPSs by analyzing the system's threat profile for possible risk scenarios depending on its underlying control attributes and communication flows between relevant internal hardware and software sub-components. The systems theoretic approach then aids the objective selection of causal events when utilizing Attack and Fault Tree models. They further demonstrated how the generated attack-defense trees may be reduced to (ATs), FTs, and AFTs using a control system case study.

### 8.2. Runtime analysis of self-adaptive systems

Studies on runtime monitoring and verification of SASs are not new. In this regard, it is important to mention an earlier study carried out by Villegas et al. (2011) that highlights the adaptation quality framework, summarised by Tamura et al. (2012) and provides the relevant background on which further studies such as ours are built upon. With respect to dynamic analysis based on feedback loops in self-adaptive systems, the framework proposed by Klös et al. (2018), leverages adaptation logic by online learning on executable run-time models (RTMs) that capture the system and environment behavior although they do not make use of AFTs. In our application of the SAFT-GT pipeline, we generate AFTs when a reconfiguration of our ROS-based system starts or ends and then leverage the mean time to failure (MTTF) value from the Storm PMC on AFTs (inputted as DFTs (see Section 5.2)) to notify our MAPE-K system of the completion of a reconfiguration of the managed system, which can be considered as a methodological contribution not covered in the work of Klös et al. (2018).

In a previous work, Barbosa et al. (2017) presented a tool, Lotus@Runtime which employs a Probabilistic Labelled Transition System (PLTS) to represent the system, and sends a notification when runtime checks of reachability properties of an updated model of a self-adaptive system results in a property violation. Similar to our work, the tool also allows the determination of probabilities and updates them in the model for each transition at runtime. However, one of the limitations of the tool is that the planning and execution phases of the MAPE-K loop are completed outside the LoTuS tool. Similarly, Carwehl et al. (2023) implemented a runtime verification method that checks the current execution of a body network sensor (BSN) system in the form of a trace against formalized requirements mapped from natural language to Metric Temporal Logic (MTL) properties. In their work, they used adaptation templates contained in a Property Adaptation Pattern (PAP) catalog to automatically instantiate adaptation properties that mirror any requirement changes in the observers of a so-called change manager connected to the managed system. Nonetheless, the monitoring, analysis and planning steps are still coordinated by a *human-in-the-loop* paradigm with the help of a requirements manager. Furthermore, their work did not mention the safety and security concerns of the system under test.

Stemming from the foregoing, our AFT generation approach follows the general outline of Steiner and Liggesmeyer (2013) of combining FTs and (ATs) and is in tandem with Nai Fovino et al. (2009) work. However, in contrast to their works, our study not only reduces inaccuracies

that may occur in manually generated models by making several aspects of the model generation processes semi-/fully automated, such as the generated (ATs), but our work also facilitates adaptation in SASs by leveraging the outcome of runtime model checking evaluations. An important aspect of our extended work is the automatic conversion of AFTs into DFTs and the utilization of the DFTs for further analysis to determine, on the one hand, the viability of our approach based on the execution time of the EPSS process already explained in detail in Section 5.2 and to trigger adaption in a self-adaptive system example.

## 9. Conclusion

In this work, we presented the SAFESEC toolchain, including all the models and how these models are combined. Furthermore, we extended the toolchain by incorporating it within a SASs. Particularly, we used a ROS2-based quadcopter example to demonstrate the application of the extended approach in SASs by integrating the toolchain in the feedback loop of the system and performing time measurements. The process of adaptation in the system is achieved through a probabilistic evaluation of the "eventuality" of an attack event with an exponential function obtained from an incorporated Storm PMC within the toolchain.

Then, we carried out an evaluation campaign through expert surveys and discussions during a workshop event. According to the obtained evaluation outcomes, we discussed the strengths and limitations of the toolchain and our extension of the pipeline, as well as possible future research directions. Some of the emerging areas of improvement include the need to make as many parts of the approach automated as possible, considering combining design time and runtime strategies for safety and security to save resources and extending the scope of vulnerabilities (attacks) beyond those that may not be available in public databases.

In the same vein, we identified that the strong safety backgrounds of the recruited experts in comparison to their security expertise in CPSs may have resulted in key security aspects being missed. As such, the results of our evaluation cannot be generalized. Yet, the results indicate that our approach is both applicable and relevant for the joint analysis of safety and security in complex cyber-physical systems. Finally, we also conducted an experimental evaluation of the extended toolchain that was integrated into the feedback loop of the SASs. While we recognize that broader applicability requires further empirical evidence, we consider this an important direction for future work and plan to evaluate the approach across more diverse systems.

## CRediT authorship contribution statement

**Irdin Pekaric:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Raffaela Groner:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Alexander Raschke:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Thomas Witte:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Jubril Gbolahan Adigun:** Writing – review & editing, Writing – original draft, Investigation, Data curation; **Michael Felderer:** Supervision, Methodology, Funding acquisition, Conceptualization; **Matthias Tichy:** Supervision, Methodology, Funding acquisition, Conceptualization.

## Data availability

No data was used for the research described in the article.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgment

This work was partially supported by the Austrian Science Fund (FWF): I 4701-N and the German Research Foundation (DFG): 435878599. It is also partially supported by Hilti and the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation.

## Reference

André, E., Lime, D., Ramparison, M., Stoelinga, M., 2019. Parametric analyses of attack-fault trees. In: 2019 19th International Conference on Application of Concurrency to System Design (ACSD), pp. 33–42. https://doi.org/10.1109/ACSD.2019.00008

Arazzi, M., Arikkat, D.R., Nicolazzo, S., Nocera, A., Rafidha Rehiman, K.A., Vinod, P., Conti, M., 2023. NLP-based techniques for cyber threat intelligence. CoRR abs/2311.08807. https://doi.org/10.48550/ARXIV.2311.08807

Arcaini, P., Riccobene, E., Scandurra, P., 2015. Modeling and analyzing MAPE-k feedback loops for self-adaptation. In: 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 13–23. https://doi.org/10.1109/SEAMS.2015.10

Barbosa, D.M., de Moura Lima, R.G., Maia, P. H.M., Junior, E.C., 2017. Lotus@runtime: a tool for runtime monitoring and verification of self-adaptive systems. In: Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. IEEE Press, p. 24-30. https://doi.org/10.1109/SEAMS.2017.18

Bilge, L., Dumitraş, T., 2012. Before we knew it: an empirical study of zero-day attacks in the real world. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA, p. 833-844. https://doi.org/10.1145/2382196.2382284

Bloomfield, R., Netkachova, K., Stroud, R., 2013. Security-informed safety: if it's not secure, it's not safe. In: International Workshop on Software Engineering for Resilient Systems. Springer, pp. 17–32.

Braun, T., Pekaric, I., Apruzzese, G., 2024. Understanding the process of data labeling in cybersecurity. In: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing. Association for Computing Machinery, New York, NY, USA, pp. 1596–1605. https://doi.org/10.1145/3605098.3636046

Budde, C.E., Kolb, C., Stoelinga, M., 2021. Attack trees vs. fault trees: two sides of the same coin from different currencies. In: Abate, A., Marin, A. (Eds.), Quantitative Evaluation of Systems. Springer International Publishing, Cham, pp. 457–467. https://doi.org/10.1007/978-3-030-85172-9_24

Carwehl, M., Vogel, T., Rodrigues, G.N., Grunske, L., 2023. Runtime verification of self-adaptive systems with changing requirements. In: 2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE Computer Society, Los Alamitos, CA, USA, pp. 104–114. https://doi.org/10.1109/SEAMS59076.2023.00024

Ceccarelli, A., Montecchi, L., 2023. Evaluating object (mis) detection from a safety and reliability perspective: discussion and measures. IEEE Access 11, 44952–44963. https://doi.org/10.1109/ACCESS.2023.3272979

Coppit, D., Sullivan, K.J., 2000. Galileo: a tool built from mass-market applications. In: Proceedings of the 22nd International Conference on Software Engineering. Association for Computing Machinery, New York, NY, USA, p. 750-753. https://doi.org/10.1145/337180.337622

David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., van Vliet, J., Wang, Z., 2011. Statistical model checking for networks of priced timed automata. In: Fahrenberg, U., Tripakis, S. (Eds.), Formal Modeling and Analysis of Timed Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 80–96. https://doi.org/10.1007/978-3-642-24310-3_7

David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., 2015. Uppaal SMC tutorial. Int. J. Softw. Tool. Technol. Trans. 17 (4), 397–415. https://doi.org/10.1007/s10009-014-0361-y

Dugan, J.B., Bavuso, S.J., Boyd, M.A., 1992. Dynamic fault-tree models for fault-tolerant computer systems. IEEE Trans. Reliab. 41 (3), 363–377. https://doi.org/10.1109/24.159800

Edifor, E., Walker, M., Gordon, N., 2012. Quantification of priority-OR gates in temporal fault trees. In: Ortmeier, F., Daniel, P. (Eds.), Computer Safety, Reliability, and Security. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 99–110. https://doi.org/10.1007/978-3-642-33678-2_9

Fournaris, A.P., Komninos, A., Lalos, A.S., Kalogeras, A.P., Koulamas, C., Serpanos, D., 2019. Security and quality in cyber-physical systems engineering. Springer International Publishing, Cham. Design and Run-Time Aspects of Secure Cyber-Physical Systems. pp. 357–382. https://doi.org/10.1007/978-3-030-25312-7_13

Fovino, I.N., Masera, M., De Cian, A., 2009. Integrating cyber attacks within fault trees. Reliab. Eng. Syst. Safe. 94 (9), 1394–1402.

Gherardi, L., 2013. Variability modeling and resolution in component-based robotics systems. Ph. D. Thesis .

Giese, H., Tichy, M., 2006. Component-based hazard analysis: optimal designs, product lines, and online-reconfiguration. In: Górski, J. (Ed.), Computer Safety, Reliability, and

Security. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 156–169. https://doi.org/10.1007/11875567_12
Groner, R., Witte, T., Raschke, A., Hirn, S., Pekaric, I., Frick, M., Tichy, M., Felderer, M., 2023. Model-based generation of attack-fault trees. In: Guiochet, J., Tonetta, S., Bitsch, F. (Eds.), Computer Safety, Reliability, and Security (SAFECOMP). Springer Nature Switzerland, Cham, pp. 107–120. https://doi.org/10.1007/978-3-031-40923-3_9
Jablonski, M., Wijesekera, D., Singhal, A., 2022. Generating cyber-physical system risk overlays for attack and fault trees using systems theory. In: Proceedings of the 2022 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems. Association for Computing Machinery, New York, NY, USA, p. 13-20. https://doi.org/10.1145/3510547.3517922
Jacobs, J., Romanosky, S., Edwards, B., Adjerid, I., Roytman, M., 2021. Exploit prediction scoring system (EPSS). Digit. Threat. 2 (3). https://doi.org/10.1145/3436242
Junges, S., Guck, D., Katoen, J.-P., Stoelinga, M., 2016. Uncovering dynamic fault trees. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 299–310. https://doi.org/10.1109/DSN.2016.35
Karaosman, E., Rizvani, A., Pekaric, I., 2026. Security Barriers to Trustworthy AI-Driven Cyber Threat Intelligence in Finance: Evidence from Practitioners. In: Proceedings of the Sixteenth ACM Conference on Data and Application Security and Privacy (CODASPY). https://doi.org/10.1145/3800506.3803505
Kephart, J.O., Chess, D.M., 2003. The vision of autonomic computing. Comput. (Long Beach Calif.) 36 (1), 41–50. https://doi.org/10.1109/MC.2003.1160055
Klös, V., Göthel, T., Glesner, S., 2018. Comprehensible and dependable self-learning self-adaptive systems. J. Syst. Archit. 85–86, 28–42. https://doi.org/10.1016/j.sysarc.2018.03.004
Kotenko, I., Chechulin, A., 2013. A cyber attack modeling and impact assessment framework. In: 2013 5th International Conference on Cyber Conflict (CYCON 2013), pp. 1–24.
Kriaa, S., Pietre-Cambacedes, L., Bouissou, M., Halgand, Y., 2015. A survey of approaches combining safety and security for industrial control systems. Reliab. Eng. Syst. Safe. 138, p. 156-178. https://doi.org/10.1016/j.ress.2015.02.008
Kumar, R., Stoelinga, M., 2017. Quantitative security and safety analysis with attack-fault trees. In: 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE), pp. 25–32. https://doi.org/10.1109/HASE.2017.12
Lallie, H.S., Debattista, K., Bal, J., 2020. A review of attack graph and attack tree visual syntax in cyber security. Comput. Sci. Rev. 35, 100219. https://doi.org/10.1016/j.cosrev.2019.100219
Lenin, A., Willemson, J., Sari, D.P., 2014. Attacker profiling in quantitative security assessment based on attack trees. In: Bernsmed, K., Fischer-Hübner, S. (Eds.), Secure IT Systems. Springer International Publishing, Cham, pp. 199–212. https://doi.org/10.1007/978-3-319-11599-3_12
Macenski, S., Foote, T., Gerkey, B., Lalancette, C., Woodall, W., 2022. Robot operating system 2: design, architecture, and uses in the wild. Sci. Rob. 7 (66), eabm6074. https://doi.org/10.1126/scirobotics.abm6074
Mauw, S., Oostdijk, M., 2006. Foundations of attack trees. In: Won, D.H., Kim, S. (Eds.), Information Security and Cryptology - ICISC 2005. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 186–198. https://doi.org/10.1007/11734727_17
Mens, T., Van Gorp, P., 2006. A taxonomy of model transformation. Electron. Note. Theor. Comput. Sci. 152, 125–142. https://doi.org/10.1016/j.entcs.2005.10.021
Muzammil, M.B., Bilal, M., Ajmal, S., Shongwe, S.C., Ghadi, Y.Y., 2024. Unveiling vulnerabilities of web attacks considering man in the middle attack and session hijacking. IEEE Access 12, 6365–6375. https://doi.org/10.1109/ACCESS.2024.3350444
Nai Fovino, I., Masera, M., De Cian, A., 2009. Integrating cyber attacks within fault trees. Reliab. Eng. Syst. Safe. 94 (9), 1394–1402. ESREL 2007, the 18th European Safety and Reliability Conference. https://doi.org/10.1016/j.ress.2009.02.020
Ou, X., Boyer, W.F., McQueen, M.A., 2006. A scalable approach to attack graph generation. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA, p. 336-345. https://doi.org/10.1145/1180405.1180446
Oueidat, T., Flaus, J.-M., Massé, F., 2020. A review of combined safety and security risk analysis approaches: application and classification. In: 2020 International Conference on Control, Automation and Diagnosis (ICCAD), pp. 1–7. https://doi.org/10.1109/ICCAD49821.2020.9260512
Pai, G.J., Dugan, J.B., 2002. Automatic synthesis of dynamic fault trees from UML system models. In: 13th International Symposium on Software Reliability Engineering, 2002. Proceedings., pp. 243–254. https://doi.org/10.1109/ISSRE.2002.1173261
Pekaric, I., Felderer, M., Steinmüller, P., 2021a. VULNERLIZER: cross-analysis between vulnerabilities and software libraries. In: 54th Hawaii International Conference on System Sciences, HICSS 2021, Kauai, Hawaii, USA, January 5, 2021. ScholarSpace, pp. 1–10.
Pekaric, I., Frick, M., Adigun, J.G., Groner, R., Witte, T., Raschke, A., Felderer, M., Tichy, M., 2024. Streamlining attack tree generation: a fragment-based approach. In: Bui, T.X. (Ed.), 57th Hawaii International Conference on System Sciences, HICSS 2024, Hilton Hawaiian Village Waikiki Beach Resort, Hawaii, USA, January 3–6, 2024. ScholarSpace, pp. 7447–7456.
Pekaric, I., Groner, R., Witte, T., Adigun, J.G., Raschke, A., Felderer, M., Tichy, M., 2023. A systematic review on security and safety of self-adaptive systems. J. Syst. Softw. 203, 111716. https://doi.org/10.1016/j.jss.2023.111716
Pekaric, I., Sauerwein, C., Haselwanter, S., Felderer, M., 2021b. A taxonomy of attack mechanisms in the automotive domain. Comput. Stand. Interface. 78, 103539. https://doi.org/https://doi.org/10.1016/j.csi.2021.103539
Pereira, D., Hirata, C., Pagliares, R., Nadjm-Tehrani, S., 2017. Towards combined safety and security constraints analysis. In: Tonetta, S., Schoitsch, E., Bitsch, F. (Eds.), Computer Safety, Reliability, and Security. Springer International Publishing, Cham, pp. 70–80. https://doi.org/10.1007/978-3-319-66284-8_7
Pfister, M., Apruzzese, G., Pekaric, I., 2025. Department-Specific Security Awareness Campaigns: A Cross-Organizational Study of HR and Accounting. In: 2025 APWG Symposium on Electronic Crime Research (eCrime), pp. 1–17.
Ponsard, C., Deprez, J.-C., Darimont, R., 2020. Formalizing security and safety requirements by mapping attack-fault trees on obstacle models with constraint programming semantics. In: 2020 IEEE Workshop on Formal Requirements (FORMREQ), pp. 8–13. https://doi.org/10.1109/FORMREQ51202.2020.00009
Prikler, L.M., Wotawa, F., 2022. Challenges of testing self-adaptive systems. In: Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume B. Association for Computing Machinery, New York, NY, USA, p. 224-228. https://doi.org/10.1145/3503229.3547048
Raiteri, D.C., Franceschinis, G., Iacono, M., Vittorini, V., 2004. Repairable fault tree for the automatic evaluation of repair policies. In: International Conference on Dependable Systems and Networks, 2004, pp. 659–668. https://doi.org/10.1109/DSN.2004.1311936
Ruijters, E., Stoelinga, M., 2015. Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. 15–16, 29–62. https://doi.org/10.1016/j.cosrev.2015.03.001
Samonas, S., Coss, D., 2014. The CIA strikes back: redefining confidentiality, integrity and availability in security. J. Inform. Syst. Secur. 10 (3).
Sauerwein, C., Pekaric, I., Felderer, M., Breu, R., 2019. An analysis and classification of public information security data sources used in research and practice. Computers & Security, 82, pp. 140–155.
Schneier, B., 1999. Modeling security threats. Dr. Dobb's J. 24 (12).
Steiner, M., Liggesmeyer, P., 2013. Combination of safety and security analysis - finding security problems that threaten the safety of a system. In: Workshop DECS (ERCIM/EWICS Workshop on Dependable Embedded and Cyber-physical Systems) of the 32nd International Conference on Computer Safety (SAFECOMP).
Swiler, L.P., Phillips, C., Ellis, D., Chakerian, S., 2001. Computer-attack graph generation tool. In: Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01. Vol. 2, pp. 307–321 vol.2. https://doi.org/10.1109/DISCEX.2001.932182
Tamura, G., Villegas, N., Müller, H., João, P.S., Becker, B., Pezzè, M., Karsai, G., Mankovskii, S., Schäfer, W., Tahvildari, L., Wong, K., 2012. Towards practical runtime verification and validation of self-Adaptive software systems. In: de Lemos, R., Giese, H., Müller, H., Shaw, M. (Eds.), Software Engineering for Self-Adaptive Systems 2. Springer. Vol. 7475 of *LNCS*, pp. 116–141. https://doi.org/10.1007/978-3-642-35813-5_5
Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F., 1981. Fault tree handbook. Technical Report. Nuclear Regulatory Commission Washington DC.
Villegas, N.M., Müller, H.A., Tamura, G., Duchien, L., Casallas, R., 2011. A framework for evaluating quality-driven self-adaptive software systems. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. Association for Computing Machinery, New York, NY, USA, p. 80-89. https://doi.org/10.1145/1988008.1988020
Weyns, D., Gerostathopoulos, I., Abbas, N., Andersson, J., Biffl, S., Brada, P., Bures, T., Di Salle, A., Galster, M., Lago, P., Lewis, G., Litoiu, M., Musil, A., Musil, J., Patros, P., Pelliccione, P., 2023. Self-adaptation in industry: a survey. ACM Trans. Auton. Adapt. Syst. 18 (2). https://doi.org/10.1145/3589227
Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., Göschka, K.M., 2013. Software Engineering for Self-Adaptive Systems II. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg. Vol. 7475. chapter On Patterns for Decentralized Control in Self-Adaptive Systems. pp. 76–107. https://doi.org/10.1007/978-3-642-35813-5_4
Witte, T., Groner, R., Raschke, A., Tichy, M., Pekaric, I., Felderer, M., 2022. Towards model co-evolution across self-adaptation steps for combined safety and security analysis. In: Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems. Association for Computing Machinery, New York, NY, USA, p. 106-112. https://doi.org/10.1145/3524844.3528062
Young, W., Leveson, N.G., 2014. An integrated approach to safety and security based on systems theory. Commun. ACM 57 (2), 31-35. https://doi.org/10.1145/2556938