EasyChair Preprint

№ 7706

Towards Model Co-Evolution Across Self-Adaptation Steps for Combined Safety and Security Analysis

Thomas Witte, Raffaela Groner, Alexander Raschke, Matthias Tichy, Irdin Pekaric and Michael Felderer

April 2, 2022

Towards Model Co-evolution Across Self-Adaptation Steps for Combined Safety and Security Analysis

Thomas Witte
Raffaela Groner
Alexander Raschke
Matthias Tichy
<firstname>.<lastname>@uni-ulm.de
Institute of Software Engineering
Ulm University, Germany

Irdin Pekaric
Michael Felderer
<firstname>.<lastname>@uibk.ac.at
Institute of Computer Science
University of Innsbruck, Austria

ABSTRACT

Self-adaptive systems offer several attack surfaces due to the communication via different channels and the different sensors required to observe the environment. Often, attacks cause safety to be compromised as well, making it necessary to consider these two aspects together. Furthermore, the approaches currently used for safety and security analysis do not sufficient take into account the intermediate steps of an adaptation. Current work in this area ignores the fact that a self-adaptive system also reveals possible vulnerabilities (even if only temporarily) during the adaptation. To address this issue, we propose a modeling approach that takes into account the different relevant aspects of a system, its adaptation process, as well as safety hazards and security attacks. We present several models that describe different aspects of a self-adaptive system and we outline our idea of how these models can then be combined into an Attack-Fault Tree. This allows modeling aspects of the system on different levels of abstraction and co-evolve the models using transformations according to the adaptation of the system. Finally, analyses can then be performed as usual on the resulting Attack-Fault Tree.

CCS CONCEPTS

• **Software and its engineering** → **System description languages**; **Fault tree analysis**; • **Computer systems organization** → *Embedded and cyber-physical systems*; **Dependable and fault-tolerant systems and networks**.

KEYWORDS

self-adaptive systems, attack-fault trees, safety, security, modeling

ACM Reference Format:
Thomas Witte, Raffaela Groner, Alexander Raschke, Matthias Tichy, Irdin Pekaric, and Michael Felderer. 2022. Towards Model Co-evolution Across Self-Adaptation Steps for Combined Safety and Security Analysis. In *17th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '22), May 18–23, 2022, PITTSBURGH, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3524844.3528062

SEAMS '22, May 18–23, 2022, PITTSBURGH, PA, USA

ACM ISBN 978-1-4503-9305-8/22/05.
https://doi.org/10.1145/3524844.3528062

1 INTRODUCTION

The use of self-adaptive systems is steadily increasing, especially, e.g, in the context of cyber-physical systems or autonomous vehicles. The advantage of such systems is that they are aware of their state and the state of their environment and, if necessary, decide for themselves whether to adapt when a state changes, and if so, how. However, this advantage also poses serious risks, since classical analyses are no longer applicable due to their inability to take the adaptation into account. Since a faulty system can cause damage to property, health and the environment, a consideration of the safety of self-adaptive systems is essential. But this is still not sufficient, since faulty system behavior can not only occur due to errors in the system, but can also be deliberately provoked by attacks.

Current research in the field of combining safety and security for self-adaptive systems is not sufficient. There is some existing work in this field, for example by Fovino et al. [13] or Kumar and Stoelinga [11], that combine Fault Trees and Attack Trees to Attack-Fault Trees. The Attack-Fault Trees presented are not specialized to the challenges of self-adaptive systems, since they are created manually and not updated automatically according to an adaptation performed. Čaušević et al. [6] present in their vision paper their idea for combining safety and security for self-adaptive systems using different models at design and at runtime. Another example is the work of Lui et al. [12], that presents an approach of defining safety and security goals and then determining which attacks could compromise them. These works are a first starting point, but both, Čaušević et al. [6] and Lui et al. [12], lack the consideration of the safety and security risks arising during an adaptation. The fact that an adaptation is not performed in zero time and cannot be considered as an atomic transaction is not taken into account.

To close this gap, we propose a modeling approach, which consists of several models, that describe different aspects of a system. We present a model to describe the data flow between individual components and a model to describe the deployment of the components of a system. These two models are closely representing the system and are able to adapt and co-evolve with the self-adaptive system. To model the individual steps of an adaptation, we use transformations which, when applied to the models, lead to the individual transient states of an adaptation process. The safety aspects are modeled using Fault Trees [29] and the security aspects are modeled using Attack Trees [23]. Since Fault Trees are modeled mainly at the component level and Attack Trees at a more detailed level, e.g., at the protocol level, these two models cannot be easily

combined. Therefore, we outline our idea on how we can combine the information from our models to generate an Attack-Fault Tree (AFT) [11]. The resulting AFT then serves as the basis for the associated safety and security analyses.

In the following section, we discuss related work and in Section 3 we present our proposed modeling approach using an illustrative example. In Section 4, we provide an outlook on the planned analyses, which are based on our modeling approach. Finally, in Section 5 we summarize and conclude this paper.

2 RELATED WORK

In this section we relate our research to other existing approaches. In the context of SAS several work regarding security aspects were published. One large topic in this area is the automatic detection of attacks by establishing intrusion detection systems (IDS), e. g [17, 18]. The authors of [24] mention a "threat database" that is used for automatic adaptation of Android Application's permissions, but in contrast to this work, this database is fed manually and not by mining CVS notifications. Pianini et al. "look at security in the context of CASs [collective adaptive systems] by reasoning on how existing patterns and recommendations may apply under the assumptions and characteristics of such systems." [19]. According to their taxonomy our work covers all of the four layers presented in the context of the "Defense in depth" principle: application, middleware, platform, and close-to-metal. While the mentioned research focus on security, there exist also some work regarding safety of SASs (e. g. [5, 8]). Only some researchers consider safety and security aspects together as shown in the following.

In [12] the authors present a safety-security co-design engineering process that is tailored for platooning system. In this process the security goals are derived from the safety goals and an attack model. Similarly, in [28] the usage of digital twins is suggested to support the identification of safety and security goals. Whereas these approaches focus on the identification of abstract safety and security problems, the vision described in [6] introduces some ideas how these models could be used. The authors mention architecture and behavioral models that shall be used to analyze safety and security requirements at run-time of a self-adaptive system. However, no details are given, how these models should look like. In our work we introduce concrete architecture and deployment descriptions that allow for an integration of known or arising vulnerabilities of used components into an analysis framework at run-time, thus bridging the gap between abstract high-level safety goal descriptions and the low-level distributed execution of a system on different platforms.

The approach of Khakpour et al. [7] focuses on vulnerabilities of the system in transient states that occur especially during architecture-based adaptations. They manually add vulnerability information to the architecture description by utilizing Acme's properties of components. Finally, they use MulVAL[16] to generate a probabilistic attack graph. We also take the adaptation process into account by modeling the adaptation as graph transformation (similarly to Bucchiarone et al. [4]) consisting of different steps. We extend this approach by considering security with respect to safety aspects. Instead of defining vulnerabilities manually, we mine them for the components involved from CVE (Common Vulnerabilities and Exposures) databases and the architecture description. Furthermore, we model not only components but also protocols or hardware in our architecture description. The generation of attack trees (in particular for a network-based model) was also done by Kotenko et al.[9] and Ou et al. [15].

The approach presented by Priesterjahn et al. [20] analyzes the time needed to finish a system adaptation before a hazard occurs. The authors introduce min-max execution time intervals for each component in order to analyze the propagation time of a failure through the system, but they do not take security aspects into account.

Several approaches influenced our ideas how to combine the different models: The combination of fault trees with attack trees can be done by introducing new gates and symbols as suggested by Kumar and Stoelinga in [11] or Stoelinga et al. in [1]. Another alternative is a less tight coupling just by allowing roots of attack trees as basic events in fault trees. We adopted this suggestion introduced by Steiner and Liggesmayer [25] and Fovino et al. [14].

Finally, the comparison of used libraries to CVE database entries is realized by commercial products like snyk[1] and for Java by Viertel et al. [30].

3 PROPOSED APPROACH

Safety and security properties are intertwined and must not be analyzed in separation. In addition, the resulting model must consider possible architecture adaptations of the system (which have an impact on these safety and security properties) and allow the model to adapt accordingly and automatically.

Figure 1 gives a broad overview over our proposed modeling approach for self-adaptive systems. Sections 3.1 and 3.5 describe how Fault and Attack Trees can be modeled independently by experts or mined from vulnerability databases. Section 3.2 introduces the logical Dataflow Graph that can be tied to events in these Fault or Attack Trees. These links may be annotated with additional impact requirements–minimum CVSS (Common Vulnerability Scoring System) impact scores required to trigger the linked event. In order to allow attacks to reference specific libraries, protocols or hardware, a Deployment Model is introduced in Section 3.3. Graph transformation rules are used to describe the system adaptation (Section 3.4). From these distinct models, a combined AFT is generated by using general combination rules, representing common weaknesses and attack patterns (Section 3.6). Analysis (Section 4) is then performed in two dimensions: state space exploration of the transformation steps and stochastic model checking on the combined AFTs.

In order to illustrate our proposed approach, we use a small, simplified example, due to space limitations. Our example consists of a ground control station (GCS) that communicates with an unmanned aerial vehicle (UAV), i.e. quadcopter, during a given mission via WiFi. The GCS sends flight commands to the UAV and receives telemetry data. If the UAV passes areas where the WiFi signal is too weak, the adaptation process starts and the system switches to longer range radio communication. The UAV then receives the command to fly back closer to the GCS. If the UAV is close enough to the GCS again, the system completes its adaptation by switching back to communicate via WiFi.

[1] https://snyk.io

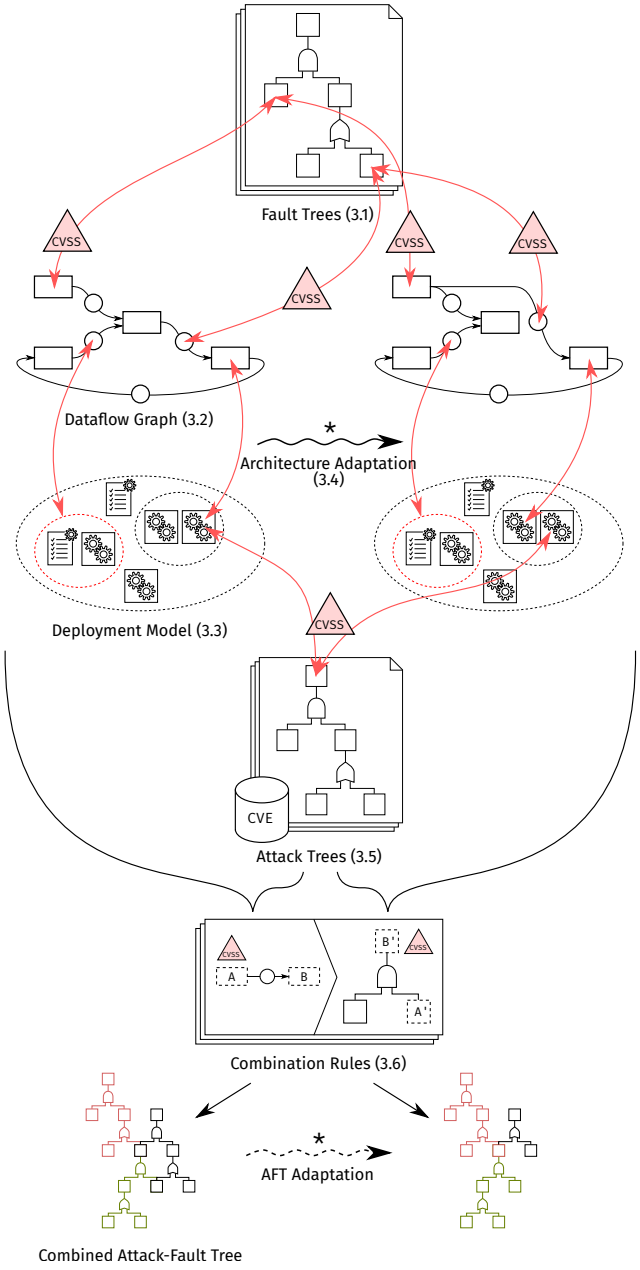

Figure 1: Overview of the modeling approach.

3.1 Fault Trees

In order to model possible safety hazards, we use *Fault Trees* (FT). Starting from an undesired event and taking into account the system and its environment, it is analyzed which events lead to the occurrence of the undesired event analyzed [29].

There are several variants of Fault Trees [22], we use a simple one as a starting point, because it can be easily extended if necessary. The leaves of a Fault Tree are so-called basic events that do not depend on other events and do not require further specification. The inner vertexes of a Fault Tree are so-called intermediate events, which are events that depend on other events. Gates are used to express the dependencies between events by linking them logically, e.g. by AND or OR [29]. Basic events in a Fault Tree can link to elements of the Dataflow Graph (cf. Section 3.2). We introduce an additional element, which is modeled as an external event (house shape in Figure 2) that contains the name of the referenced Dataflow Graph element and the minimum required CVSS impact on the dataflow element to trigger the linked event. The impact specification corresponds to the CVSS metrics: for example, an event can cause or require an impact on the Confidentiality, Integrity or Availability of the channel or component. The basic event "connection lost" in a Fault Tree might reference one or more dataflow channels and require a high impact on the availability (e.g. caused by a DoS-attack) in order to activate this basic event. Vice versa, the top event "data corruption in component" of an Attack or Fault Tree can reference a component of the *Dataflow Graph* and cause a high integrity impact on the component if the event is activated.

In our approach, we assume that a Fault Tree is modeled for each fault manually by a safety expert, resulting in a forest of fault trees. These Fault Trees are immutable, however, the linked dataflow elements or their deployment might change during reconfiguration.

Figure 2 displays an excerpt of a Fault Tree for our UAV example. The analyzed safety risk is the loss of control of the UAV, which can lead to theft of the UAV, property damage or even injuries. One way that this undesirable event occurs is through the occurrence of the two basic events (shown as ellipses in Figure 2) "No WiFi connection" AND "No radio connection". The intermediate event "Incorrect return message" is linked to the "commands"-channel in the Dataflow Graph using an external event. Since there are other combinations of events that lead to loss of control, the AND-gate and the intermediate event are connected to the top event using an OR-gate. For reasons of readability, we have omitted further parts of the Fault Tree.

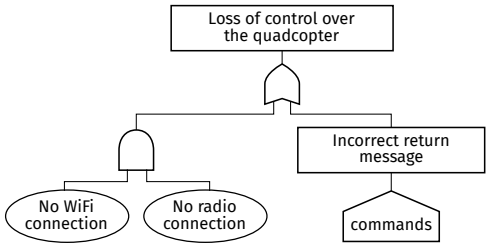

Figure 2: Excerpt of a Fault Tree for the example.

3.2 Dataflow Graph

The dataflow between the software components of an application, subsystems and physical components contains information how failures or attacks may spread and proliferate. A *Dataflow Graph* that contains the system's components and abstract connections, i.e. not regarding the deployment or realization of the data channel, is used to determine whether an event in one part of the system can have an impact on other parts of the system.

The high level of abstraction of this *Dataflow Graph* simplifies modeling implicit or indirect information flow, e.g. side channels that may leak information or sensors measuring the system state

through the environment. This logical dataflow is intentionally architecture and technology agnostic to possibly model any kind of system. The specific protocols or technology used is specified in the *Deployment Model*.

The *Dataflow Graph* consists of *components* and *channels* between these components. Components can send messages through directed channels with an arbitrary number of senders and receivers per channel.

A fault or attack event (in the respective tree) can reference a component or channel that is affected by or the source of the event. The relation of linked Fault or attack events, e.g. that an attack event has a cascading effect affecting other components, is expressed through combination rules (Section 3.6) that define the structure of the generated AFT.

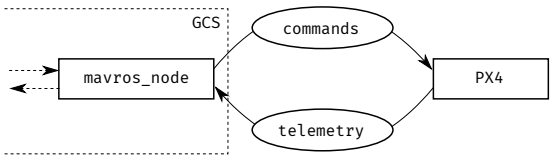

Figure 3: Excerpt of the Dataflow Graph for the example.

Figure 3 shows an excerpt of the Dataflow Graph for our running example. The mavros_node in the GCS communicates with the quadcopter (PX4) through a command and telemetry channel. The Fault Tree can reference the *commands* through an external event irrespective of the realization of the channel (WiFi or radio).

The Dataflow Graph can be generated if the platform or framework used offers appropriate introspection capabilities. For example, the node graph of a running ROS [21] system can be obtained at runtime and directly transformed into a Dataflow Graph. Alternatively, the Dataflow Graph can be derived statically from e.g. SysML internal block diagrams. Additional dataflow information, implicit or indirect channels can be added manually ahead of time.

3.3 Deployment Model

Every component in the Dataflow Graph runs on a specific hardware (HW) with a given operating system (OS) that provides a set of libraries that might be used by the components. Similarly, every communication channel can use a protocol (stack) that is typcially also implemented in libraries running on an OS on a given HW. Each involved unit (protocol, library, HW) can have one or more vulnerabilites that could be exploited to attack the system.

The goal of the Deployment Model is on the one side to model the overall architecture of a system on different levels of abstraction and on the other side to define the deployment of the components and channels of the Dataflow Graph on the different parts of the system.

For this purpose, we define a transitive dependency relation $c \rightarrow D$ with c is the identifier (e. g. a common platform enumeration entry (CPE)[2]) of a component/channel/protocol/library/set of libraries/hardware/proxy, D is a set of protocols/libraries/hardware components, and c depends on the entries given in D. Types are added to each of these elements for a better differentiation. The combination of dependencies describes the set of platforms on which components can be deployed. In this context, a *proxy* is an abstract placeholder for a concrete instance of a platform. In the following example, *WiFi* is a proxy for a concrete instance of an existing WiFi network with certain properties. The instantiation of this proxy is done in a separate entry.

The deployment of a component (of the Dataflow Graph) is just a top level dependency of the Deployment Model. Thus, the highest level of the Deployment Model correspond to elements of the Dataflow Graph and the lowest level correspond to (potential) vulnerable elements like protocols, libraries, and hardware components. An architecture reconfiguration can be modeled as a change of the deployment of a component which includes the exchange of a component by a new one.

In our running example, an excerpt of a simplified Deployment Model might look like this (for sake of brevity, only left-hand-side types are shown):
PX4:Component $\rightarrow$ {*Mavlink2.0, pixhawk, ...*}
Mavlink2.0:Protocol $\rightarrow$ {*MavlinkLib, UDP, UART, ...*}
commands:Channel $\rightarrow$ {*Mavlink2.0, WiFi*}
WiFi:Platform $\rightarrow$ {*IEEE 802.11n, WPA2, UDP, TCP/IP, ...*}
Radio:Platform $\rightarrow$ {*UART*} ...

Note, that some of the dependencies can be derived automatically (e.g. by examining the dependencies of a linux binary or by retrieving the installed library versions of an OS).

Due to the transitive property of the relation it is possible to combine the dependency sets, e.g.: *commands* $\rightarrow$ {*Mavlink2.0, MavlinkLib, IEEE 802.11n, WPA2, UDP, TCP/IP, UART, ...*}.

3.4 Architecture Adaptation

In order to describe the adaptation of a system, we use graph transformations. Thereby, the individual steps of an adaptation to be performed are described by several individual transformation rules. These transformation rules can then be applied to the Dataflow Graph and the Deployment Model to automatically achieve the transient states during an adaptation.

We use Henshin [26] to define the transformations. This is a declarative transformation language where transformation rules are created by defining a precondition that a model must satisfy in order for the rule to be applied. If the precondition is satisfied, the model is updated to satisfy the postcondition of the rule.

The first step of the adaption performed in our UAV example is the switch between the communication via WiFi to radio. This can be described in the textual syntax of Henshin with the following excerpt:

```
graph {
  node commands:Channel
  node WiFi:Platform
  node Radio:Platform
  edges[(delete commands->WiFi:depends),
        (create commands->Radio:depends)]}
```

The precondition specifies that the *commands*-channel depends on *WiFi*. The postcondition states that the *commands*-channel should depend on *Radio* and no longer on *WiFi*. This change is defined in Henshin by the keywords delete and create. Where delete marks the

[2]a unified naming scheme for hardware, software, and packages, see https://nvd.nist.gov/products

parts of the precondition that must not exist after the application. And create marks the parts of the postcondition that will be created after the application. Parts without one of the two keywords are part of the precondition and the postcondition, and must be valid before and after the transformation application

In order to model the entire adaptation process, further transformations must be defined manually, which describe also the individual transient adaptation steps. In our example, this means, e.g, that a transformation must be defined, which switches the communication back again, in order to describe the entire adaptation.

3.5 Attack Trees

Potential attacks on platforms and protocols are modeled using *Attack Trees* (AT). An Attack Tree represents various ways and necessary steps through which an adversary can attack the system. Due to possible system reconfiguration, it is necessary to generate Attack Trees for every platform, library or protocol used in the system. The target of such an Attack Tree (e.g. a library) is modeled as an external event that can be linked to the respective resource in the Deployment Model.

Attack Trees are generated using a semi-automated approach, similar to [15] by incorporating the data from three sources: CVE, CVSS, and system library data. Each attack targets a platform, identified by its CPE entry, and may use one or more vulnerabilities to impact it. The vulnerability data is obtained by implementing active monitors allowing the collection of the most recently published CVE data, the generation of corresponding Attack Trees and creation of an attack database for the system. The characteristics and severity of vulnerabilities are represented using CVSS data. Finally, the system library data includes all the libraries and applications, as well as their specific versions that are used by a self-adaptive system. These can be linked to the CVE data because it also includes information regarding the library that was affected by a specific vulnerability.

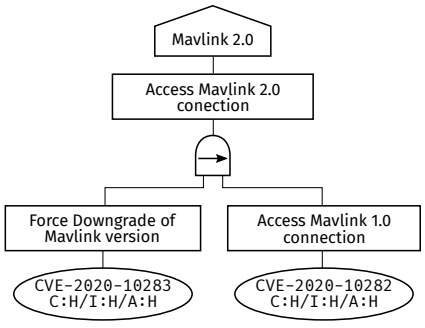

Figure 4: Example of an attack tree.

The root element (external event) describes the target of an attack and serves as an interface to link the attack to the combined AFT. For example, the exploitation of two vulnerabilities could result in an attack on the *Mavlink* Protocol (Figure 4). Multiple vulnerabilities (CVEs) can be linked together to build an attack path. In addition, the CVSS:3.1 vector is added to the CVE identifier in order to provide additional vulnerability information such as exploitability, impact, temporal score, and environmental score metrics for each basic attack step. This also includes impact measures on CIA metrics. By utilizing the vector data, we limit applicable combination rules to rules matching the characteristics of the attack.

3.6 Model Combination

Joint analysis of safety and security requires combining Attack and Fault Trees into AFTs using additional information from the Dataflow Graph and Deployment Model. Attack and Fault Trees often operate on different levels of abstraction, i.e. attacks mostly target libraries, frameworks, protocols or hardware while basic events of Fault Trees are mostly triggered by missing or corrupted logical data flow or components. This different levels of abstraction are mirrored in the logical Dataflow Graph and the Deployment Model which specifies concrete systems and software implementing this data flow.

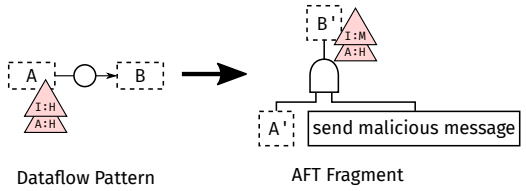

Figure 5: Example combination rule for CWE-20: Improper Input Validation.

Using a rule based translation scheme, patterns in the data flow or deployment model create partial AFTs – representing generic attack patterns or common weaknesses – that are connected to external events in the Attack and Fault Trees, combining them and bridging the abstraction gap. Rules are derived from Common Weakness Enumeration (CWE) as well as Common Attack Pattern Enumeration and Classification (CAPEC) data. These combination rules can, for example, translate a logical data flow between two components A and B into a tree fragment shown in Figure 5. Any Attack or Fault Tree referencing component A or B can be connected to the events A' or B' respectively. Such a rule might represent the additional attack step needed to forge an invalid message in component A and sending it in order to compromise component B. To forge such a manipulated message, the attack or fault event connecting to A' needs to satisfy additional impact criteria, in this case on the *integrity* or *availability* of component A. These impact requirements, i.e. required values in the CVSS vector, serve as an interface to combine only compatible events. Combination rules, fault or attack events as well as references to elements in other models can be annotated with such impact specifications.

Similarly, combination rules can express the spreading of faults along data flow or how an attack on a vulnerable library allows the attacker to also compromise a component using it – thereby connecting a logical component from the Dataflow Graph with the libraries its realization uses.

In our example scenario, the Fault Tree leading to control loss and theft of the drone (Figure 2) and the Attack Tree in Figure 4 can be combined into the AFT shown in Figure 6. The Dataflow Graph and Deployment Model—in the transient reconfiguration state, when the WiFi connection is lost, but the return signal is not yet sent—provide the necessary information to join the models

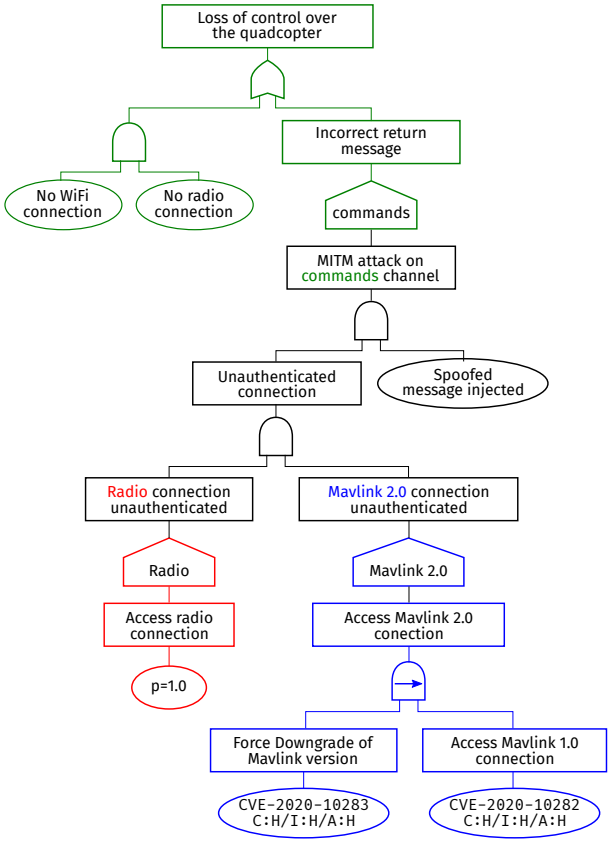

Figure 6: Combined AFT of the example.

despite their incompatible abstraction levels. Two combination rules are used to combine the trees. These are derived from two CAPEC patterns:

CAPEC-115: Authentication Bypass A channel is accessible without authentication if all constituent protocols, hardware, etc. is accessible without authentication.

CAPEC-594: Traffic Injection A malicious actor can insert a spoofed message into a channel, if it requires no authentication.

These rules are used to generate the partial tree (black) that connects the external events of the Attack and Fault Trees. The first rule matches the deployment of the *commands* channel, connecting it to attacks on the authentication of the platforms and protocols used. Next, the second rule matches the *commands* channel in the dataflow and is connected to the external event in the fault tree.

The resulting AFT shows, that the Mavlink vulnerability in combination with using radio communication can lead to the control loss over the quadcopter. This state occurs during adaptation, after the quadcopter flies out of WiFi range by injecting a false return message into the commands channel.

4 ANALYSIS APPROACH

In this section, we provide an outlook on the two dimensions of our planned analysis approach.

Attack-Fault Tree Analysis. For each state of the system (transient as well as steady), the corresponding combined AFT is generated. Existing approaches for analyzing AFTs can be used, e.g. first translating the AFT into stochastic timed automata as input to existing model checkers, e. g. Uppaal SMC [11]. Our proposed combination approach for ATs and FTs will be extended to provide the necessary timing and probability information. Already, events in ATs and FTs are often annotated with timing or probability information. By enriching the Dataflow Graph with additional timing information, representing the delay and frequency of messages or the processing time of these messages in components, timing constraints on adaptations can be derived in the analysis, similar to [20]. For example, an adaptation to avoid the spread of an error across multiple components must be faster than the spreading of the error.

Henshin state space analysis. Since we model the individual steps of an adaptation using Henshin transformation rules, we can also use the state space analysis provided by the Henshin tool set [2]. This allows to explore the entire state space resulting from the transformations, including the transient states that may occur during adaptation with respect to the Dataflow Graph and the Deployment Model. By examining the state space, resulting reachable states can subsequently be examined in more detail with the help of the AFT. This requires applying the individual transformations to the models and then generating a new AFT. In addition, the correctness of the modeling of the adaptation can be examined with the help of invariants. For example, to ensure that there exists no combination of transformations that results in the command channel communicating via radio and WiFi at the same time.

5 CONCLUSION AND FUTURE WORK

In this vision paper, we present an integrated approach to model and analyze safety and security aspects of a self-adaptive system. Bridging the gap between high-level modeling and low-level implementation on specific platforms enables the analysis of vulnerabilities of system components with respect to safety requirements. We also consider intermediate steps of an adaptation phase, which may provide new attack surfaces. The next steps of our research are to implement the different models and to define precisely the translation into AFTs. Currently, our approach does not consider time and probability aspects, although this is an important point since, e.g., long lasting high effort attacks on vulnerabilities that exist for only a short time during adaptation are less likely than attacks that can be executed within a short time [1, 10]. In addition, do we plan to use our approach also at run-time to continuously analyze a system with respect to safety and security. This allows us to include emerging security vulnerabilities even during the runtime of the system. To do this, however, we must ensure that the models are kept in sync with the system state as efficiently as possible and that the translation and analysis of the AFT can be performed within fixed time limits [3, 27]. Additionally, we plan a case study in which we use our approach to analyze larger and realistic systems in order to evaluate its scalability.

ACKNOWLEDGMENTS

This work was partially supported by the Austrian Science Fund (FWF): I 4701-N and German Research Foundation (DFG): 435878599.

REFERENCES

[1] Étienne André, Didier Lime, Mathias Ramparison, and Mariëlle Stoelinga. 2019. Parametric analyses of attack-fault trees. *CoRR* abs/1902.04336 (2019). arXiv:1902.04336 http://arxiv.org/abs/1902.04336

[2] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. 2010. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In *Model Driven Engineering Languages and Systems*, Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 121–135. https://doi.org/10.1007/978-3-642-16145-2_9

[3] Amel Bennaceur, Robert France, Giordano Tamburrelli, Thomas Vogel, Pieter J. Mosterman, Walter Cazzola, Fabio M. Costa, Alfonso Pierantonio, Matthias Tichy, Mehmet Akşit, Pär Emmanuelson, Huang Gang, Nikolaos Georgantas, and David Redlich. 2014. *Mechanisms for Leveraging Models at Runtime in Self-adaptive Software.* Springer International Publishing, Cham, 19–46. https://doi.org/10.1007/978-3-319-08915-7_2

[4] Antonio Bucchiarone, Hartmut Ehrig, Claudia Ermel, Patrizio Pelliccione, and Olga Runge. 2015. *Rule-Based Modeling and Static Analysis of Self-adaptive Systems by Graph Transformation.* Springer International Publishing, Cham, 582–601. https://doi.org/10.1007/978-3-319-15545-6_33

[5] Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. 2021. Runtime Equilibrium Verification for Resilient Cyber-Physical Systems. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS).* 71–80. https://doi.org/10.1109/ACSOS52086.2021.00025

[6] Aida Causevic, Alessandro V. Papadopoulos, and Marjan Sirjani. 2019. Towards a Framework for Safe and Secure Adaptive Collaborative Systems. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2. 165–170. https://doi.org/10.1109/COMPSAC.2019.10201

[7] Narges Khakpour, Charilaos Skandylas, Goran Saman Nariman, and Danny Weyns. 2019. Towards Secure Architecture-Based Adaptations. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS).* 114–125. https://doi.org/10.1109/SEAMS.2019.00023

[8] Dominik Klumpp, Axel Habermaier, Benedikt Eberhardinger, and Hella Seebach. 2016. Optimising Runtime Safety Analysis Efficiency for Self-Organising Systems. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W).* 120–125. https://doi.org/10.1109/FAS-W.2016.37

[9] Igor Kotenko and Andrey Chechulin. 2013. A Cyber Attack Modeling and Impact Assessment framework. In *2013 5th International Conference on Cyber Conflict (CYCON 2013).* 1–24.

[10] Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. 2015. Quantitative Attack Tree Analysis via Priced Timed Automata. In *Formal Modeling and Analysis of Timed Systems*, Sriram Sankaranarayanan and Enrico Vicario (Eds.). Springer International Publishing, Cham, 156–171. https://doi.org/10.1007/978-3-319-22975-1_11

[11] Rajesh Kumar and Mariëlle Stoelinga. 2017. Quantitative Security and Safety Analysis with Attack-Fault Trees. In *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE).* 25–32. https://doi.org/10.1109/HASE.2017.12

[12] Jiafa Liu, Di Ma, Andre Weimerskirch, and Haojin Zhu. 2017. A Functional Co-Design towards Safe and Secure Vehicle Platooning. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security* (Abu Dhabi, United Arab Emirates) *(CPSS '17).* Association for Computing Machinery, New York, NY, USA, 81–90. https://doi.org/10.1145/3055186.3055193

[13] Igor Nai Fovino, Marcelo Masera, and Alessio De Cian. 2009. Integrating cyber attacks within fault trees. *Reliability Engineering & System Safety* 94, 9 (2009), 1394–1402. https://doi.org/10.1016/j.ress.2009.02.020 ESREL 2007, the 18th European Safety and Reliability Conference.

[14] Igor Nai Fovino, Marcelo Masera, and Alessio De Cian. 2009. Integrating cyber attacks within fault trees. *Reliability Engineering & System Safety* 94, 9 (2009), 1394–1402. https://doi.org/10.1016/j.ress.2009.02.020 ESREL 2007, the 18th European Safety and Reliability Conference.

[15] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. 2006. A Scalable Approach to Attack Graph Generation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (Alexandria, Virginia, USA) *(CCS '06).* Association for Computing Machinery, New York, NY, USA, 336–345. https://doi.org/10.1145/1180405.1180446

[16] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. 2005. MulVAL: A Logic-Based Network Security Analyzer. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14* (Baltimore, MD) *(SSYM'05).* USENIX Association, USA, 8.

[17] Jesus Pacheco and Salim Hariri. 2016. IoT Security Framework for Smart Cyber Infrastructures. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W).* 242–247. https://doi.org/10.1109/FAS-W.2016.58

[18] Jesus Pacheco, Xiaoyang Zhu, Youakim Badr, and Salim Hariri. 2017. Enabling Risk Management for Smart Infrastructures with an Anomaly Behavior Analysis Intrusion Detection System. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W).* 324–328. https://doi.org/10.1109/FAS-W.2017.167

[19] Danilo Pianini, Roberto Casadei, and Mirko Viroli. 2019. Security in Collective Adaptive Systems: A Roadmap. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W).* 86–91. https://doi.org/10.1109/FAS-W.2019.00034

[20] Claudia Priesterjahn, Dominik Steenken, and Matthias Tichy. 2013. *Timed Hazard Analysis of Self-healing Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 112–151. https://doi.org/10.1007/978-3-642-36249-1_5

[21] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. Kobe, Japan, 5.

[22] Enno Ruijters and Mariëlle Stoelinga. 2015. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review* 15-16 (2015), 29–62. https://doi.org/10.1016/j.cosrev.2015.03.001

[23] Bruce Schneier. 2015. *Secrets and Lies: Digital Security in a Networked World.* Wiley.

[24] Gian Luca Scoccia, Marco Autili, and Paola Inverardi. 2020. A self-configuring and adaptive privacy-aware permission system for Android apps. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS).* 38–47. https://doi.org/10.1109/ACSOS49614.2020.00024

[25] Max Steiner and Peter Liggesmeyer. 2013. Combination of Safety and Security Analysis - Finding Security Problems That Threaten the Safety of a System. In *32nd International Conference on Computer Safety, Reliability and Security (SAFECOMP 2013). Workshops and Tutorials : CARS, SASSUR, DECS, ASCOMS.* http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-43604

[26] Daniel Strüber, Kristopher Born, Kanwal Daud Gill, Raffaela Groner, Timo Kehrer, Manuel Ohrndorf, and Matthias Tichy. 2017. Henshin: A Usability-Focused Framework for EMF Model Transformation Development. In *Graph Transformation*, Juan de Lara and Detlef Plump (Eds.). Springer International Publishing, Cham, 196–208. https://doi.org/10.1007/978-3-319-61470-0_12

[27] Michael Szvetits and Uwe Zdun. 2016. Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. *Software & Systems Modeling* 15, 1 (2016), 31–69. https://doi.org/10.1007/s10270-013-0394-9

[28] Omar Veledar, Violeta Damjanovic-Behrendt, and Georg Macher. 2019. Digital Twins for Dependability Improvement of Autonomous Driving. In *Systems, Software and Services Process Improvement*, Alastair Walker, Rory V. O'Connor, and Richard Messnarz (Eds.). Springer International Publishing, Cham, 415–426. https://doi.org/10.1007/978-3-030-28005-5_32

[29] William E Vesely, Francine F Goldberg, Norman H Roberts, and David F Haasl. 1981. *Fault tree handbook.* Technical Report. Nuclear Regulatory Commission Washington DC.

[30] Fabien Patrick Viertel, Fabian Kortum, Leif Wagner, and Kurt Schneider. 2019. Are Third-Party Libraries Secure? A Software Library Checker for Java. In *Risks and Security of Internet and Systems*, Akka Zemmari, Mohamed Mosbah, Nora Cuppens-Boulahia, and Frédéric Cuppens (Eds.). Springer International Publishing, Cham, 18–34.