Using Attack and Failure Propagation Analysis for Context-Aware Security Control Suggestions

Roman Trentinaglia[1][a], Thorsten Koch[1] and Eric Bodden[1,2][b]
[1]*Safe and Secure IoT Systems, Fraunhofer IEM, Paderborn, Germany*
{firstname.lastname}@iem.fraunhofer.de

[2]*Paderborn University, Department of Computer Science, Heinz Nixdorf Institute, Paderborn, Germany*
{firstname.lastname}@uni-paderborn.de

Keywords: Safety & Security by Design, Security Control Suggestions, SeCFT, Minimal Cut Sets

Abstract: Cybersecurity is becoming increasingly important, especially in safety-critical domains, where cyber attacks can pose significant safety risks. In response, standards and laws such as the Cyber Resilience Act (CRA) require product teams to conduct comprehensive assessments of cybersecurity threats and implement appropriate security controls throughout the product lifecycle. Despite the availability of structured catalogs for requirements and mitigations, there is currently no automated method for integrating threat analysis results with these catalogs or for determining optimal control deployment strategies. Furthermore, addressing threats in isolation often results in long and redundant lists of potential controls, which increases development costs and complexity. To bridge this gap, we propose a semi-automated, model-based approach to suggest security controls. Our approach utilizes Security-enhanced Component Fault Trees (SeCFT) to analyze attack and failure propagation and employs a structured catalog to generate context-specific control recommendations along with appropriate deployment locations. This approach helps engineers efficiently select a coherent set of controls, enabling them to build a robust, multi-layered defense. We validated our approach through a proof-of-concept implementation in a real-world case study.

1 Introduction

Modern software-controlled systems are increasingly connected, raising the likelihood of cyber attacks (Sommer et al., 2019). In safety-critical domains such as automotive, aviation, or industrial control, these attacks can cause hazards that endanger human life or the environment (Dantas and Nigam, 2023). Consequently, such systems must be both safe and secure, as required by current standards (e.g., ISO 21434, IEC 62443) that mandate joint consideration of safety and security. To limit costly redesigns, reduce development risk, and ensure compliance, threats, hazards, and their interdependencies must be identified early in development (Fockel et al., 2022). Identified threats, especially those that may lead to hazards, must then be mitigated by suitable *security controls* deployed in the system architecture.

However, for a whole system, it is not trivial to find a good set of suitable security controls. On the one hand, due to the vast amount of different potential threats and controls and since security expertise is rare, selecting a fitting control for each individual threat can be a hard task on its own. On the other hand, as controls may fail, miss an attack, or be deliberately disabled by an attacker, no control can provide a perfect protection against all attacks. Therefore, some threats may require multiple coordinated layers of independent controls. In addition, as the location of a control (e.g., a message filter) in the system architecture influences its effectiveness and its ability to mitigate other threats, finding suitable deployment locations for controls is not trivial. Current work on this topic is often detached from the actual system architecture, does not offer recommendations for suitable deployment locations, or does not take into account the reusability of controls for multiple threats.

In preliminary work (Fockel et al., 2022), we have shown that interdependencies between hazards and threats can be determined semi-automatically by calculating a *Security-enhanced Component Fault Tree*

[a] https://orcid.org/0000-0001-9728-4991
[b] https://orcid.org/0000-0003-3470-3647

(SeCFT) (Steiner, 2016). In this paper, we present a model-based approach that uses the results of a SeCFT analysis to suggest security controls from a structured catalog. Our approach enables context-specific suggestions of security controls along with potential deployment locations in the system architecture. We thereby support engineers in implementing a safety-informed threat modeling process.

Our paper is structured as follows: In Section 2, we illustrate the research challenges on a running example. Section 3 presents our approach and our contributions. In Section 4, we evaluate our approach based on a real-world case. Section 5 presents related work. In Section 6, we conclude our paper and present future work.

2 Research Challenges

In this section, we illustrate our research challenges based on a running example inspired by the ISO 21434 automotive security standard.

Running Example. Our exemplary system processes vehicular sensor data (e.g., camera-detected obstacles) to provide adaptive cruise control and automatic headlamp control. Figure 1 shows the corresponding safety-informed threat model as a Data Flow Diagram (DFD). Within the Internal Electronic Control Units (ECUs) Trust Boundary, three software *processes* (circles) interact with each other and with *external actors* (rectangles): Body Control Module (BCM), Headlamp System, and Adaptive Cruise Control (ACC). The Headlamp System uses inputs from a driver-operated dashboard button, connected to the BCM, and a Camera to control and automatically dim the headlamps when oncoming traffic is detected. The ACC uses the same camera data, cross-correlated with redundant radar_data, to maintain distance and brake if needed.

As connected vehicles are both safety- and security-critical, hazards and threats need to be identified. Wrongly detected obstacles or oncoming vehicles can cause phantom braking (Hazard H1) or loss of headlight (Hazard H2), potentially leading to a crash. We consider two example threats: T1, a *Spoofing* threat sending fake messages in the name of the Camera, and T2, a *Denial of Service* threat flooding the BCM and making it unresponsive. In safety-critical systems, threats that lead to hazards must be mitigated with priority, as they potentially have life-threatening consequences.

Challenges in Finding Suitable Controls. To mitigate these threats and to reduce the overall risk of the system, suitable *security controls* need to be found and deployed in the system architecture. While there are already extensive catalogs for security controls like *MITRE ATT&CK*, finding a good set of controls for a whole system is not a trivial task.

On the one hand, these catalogs are very large and, due to n-m mappings between threats and controls, very complex. Without tool support, it is difficult to select the right controls for identified threats and find a fitting deployment location in the system architecture, especially for beginners. Furthermore, in safety-critical systems, interactions with existing safety controls need to be respected. This is further complicating the choice of the right control. For example, a *Watchdog* is a safety control which automatically restarts unresponsive devices that may be used by an attacker to deliberately carry out a *Denial of Service* attack. Similarily, a *Message Filter* is a security control that, if configured incorrectly, may block safety-critical messages and thus cause hazards. Without taking into account architectural factors and relationships between threats and controls, finding the right control becomes a hard task on its own.

On the other hand, every control involves implementation effort and costs money and, in some cases, has a performance impact. Therefore, a software engineer should carefully consider which controls to use and only implement those controls that are needed to reduce the risk efficiently. For example, a message filter deployed once (in the right location) to protect a subnetwork from external access can eliminate the need for alternative or additional measures within the subnetwork (e.g., encryption of each individual data flow). However, a message filter can fail to detect malicious messages or be deliberately deactivated by an attacker. To ensure a safe operation, the risk of safety-critical threats therefore needs to be reduced to the lowest feasible level, potentially with multiple controls. Multiple layered controls are also a building block of the *defense in depth* secure design principle (Stytz, 2004). As safety and security analyses are often performed in silos, it is difficult to determine which threats are safety-critical. Yet, to decide which control needs to be deployed where, one needs to know through which path a threat can trigger hazards. Our solution should therefore assist in finding an efficient set of security controls and give suggestions for suitable locations in the system architecture.

3 Suggesting Suitable Security Controls

In this section, we present our approach with two main contributions: (a) a meta-model for structuring a catalog of security controls, and (b) a semi-

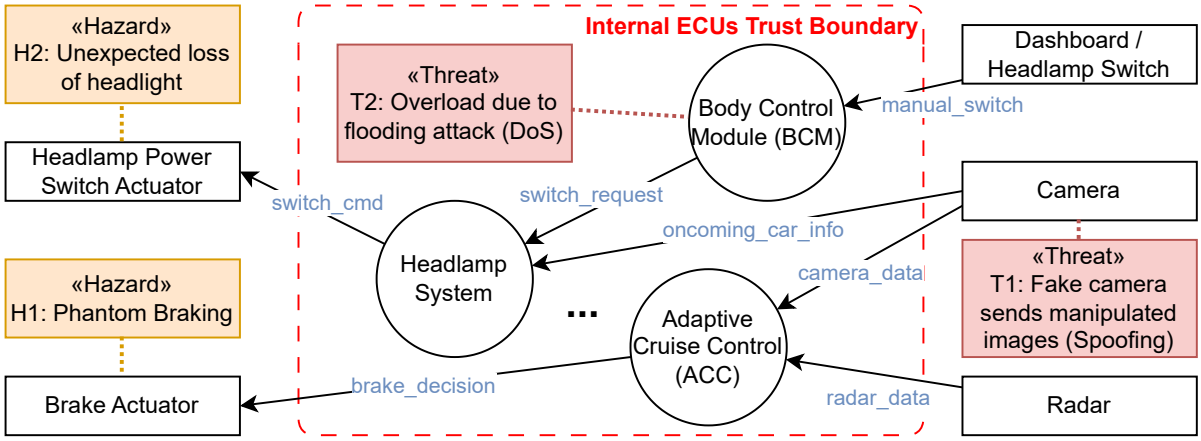

Figure 1: Safety-informed threat model for our automotive running example represented by a Data Flow Diagram (DFD) showing external actors (white rectangles), software processes (circles), messages (blue), threats (red), hazards (orange)

automated, tool-supported process for suggesting these controls. Our goal is to help engineers to select appropriate controls from large catalogs and identify suitable deployment locations in the system architecture. Building on our preliminary work (Fockel et al., 2022), we use SeCFTs to guide control selection and placement based on failure propagation information. With this paper, we do not provide a ready-to-use catalog, but a meta-model that engineers can instantiate with their own controls.

Meta-Models for Catalog and Threat Model. Figure 2 presents the meta-model for the security control catalog (top), a sample instantiation of the catalog (center), and a meta-model for the safety-informed threat model (bottom).

The *catalog meta-model* consists of three main elements: a ControlType for defining security control types that can later be instantiated in the actual threat model, a ThreatType to define types of threats, and a ControlMapping to define relationships between ThreatTypes and ControlTypes to enable suggestions of security controls.

The *threat model meta-model* defines types to be used in the actual safety-informed threat model. This meta-model allows the creation of DFDs, as presented in Figure 1, which represent the system architecture and include all identified hazards and threats. In the DFD, ThreatTypes from the catalog are instantiated as Threats (depicted as «Threat» in Fig. 1) to enable suggestions for applicable ControlTypes. ControlTypes can be derived from existing catalogs, e.g., *MITRE ATT&CK* or company-specific catalogs, and are instantiated by the user as concrete Controls in the threat model.

A Threat is annotated to, i.e., affects a DFDElement (e.g., a Process, External Actor, etc.) and may cause FailureModes corresponding to the FailureKinds specified by the ThreatType. We omit FailureModes in Figure 1 for readability and since they are usually not part of a security analysis. For each Message in the DFD, two FailureModes can exist for each FailureKind: an *outgoing* FailureMode at the sending component, and an *incoming* FailureMode at the receiver. FailureModes can lead to other FailureModes resulting in a failure propagation chain that eventually results in a hazard. Existing failure propagation models, like SeCFTs, can be reused to import failure propagation information into the threat model. In turn, as presented in (Fockel et al., 2022), SeCFTs can be derived (semi-)automatically from system architecture and dynamic behavior models. Finally, by mitigating certain FailureModes (depending on the mitigatesFailureKinds attribute of the respective ControlMapping), Controls avert the negative effect of Threats on the system.

The center of Figure 2 shows a sample catalog instance based on our catalog meta-model with two ThreatTypes: Flooding (a *Denial of Service* threat) and SpoofComponent (a *Spoofing* threat). Each ThreatType specifies, via causesFailureKinds, which FailureKinds (**O**mission, **C**ommission, **V**alue) it can cause. For example, Flooding can crash a component by sending excessive requests, leading to omission failures of its outgoing messages. By assigning concrete failure kinds to ThreatType instances, we refine the generic STRIDE–failure mapping of (Fockel et al., 2022) and can specify effects of individual threats more precisely.

The catalog also defines security control types (ControlType) that directly affect failure modes in the system's failure propagation. For example, a *Message Filter*[1] (cf. MsgFilter) can detect manipulated

[1]https://attack.mitre.org/mitigations/M0937

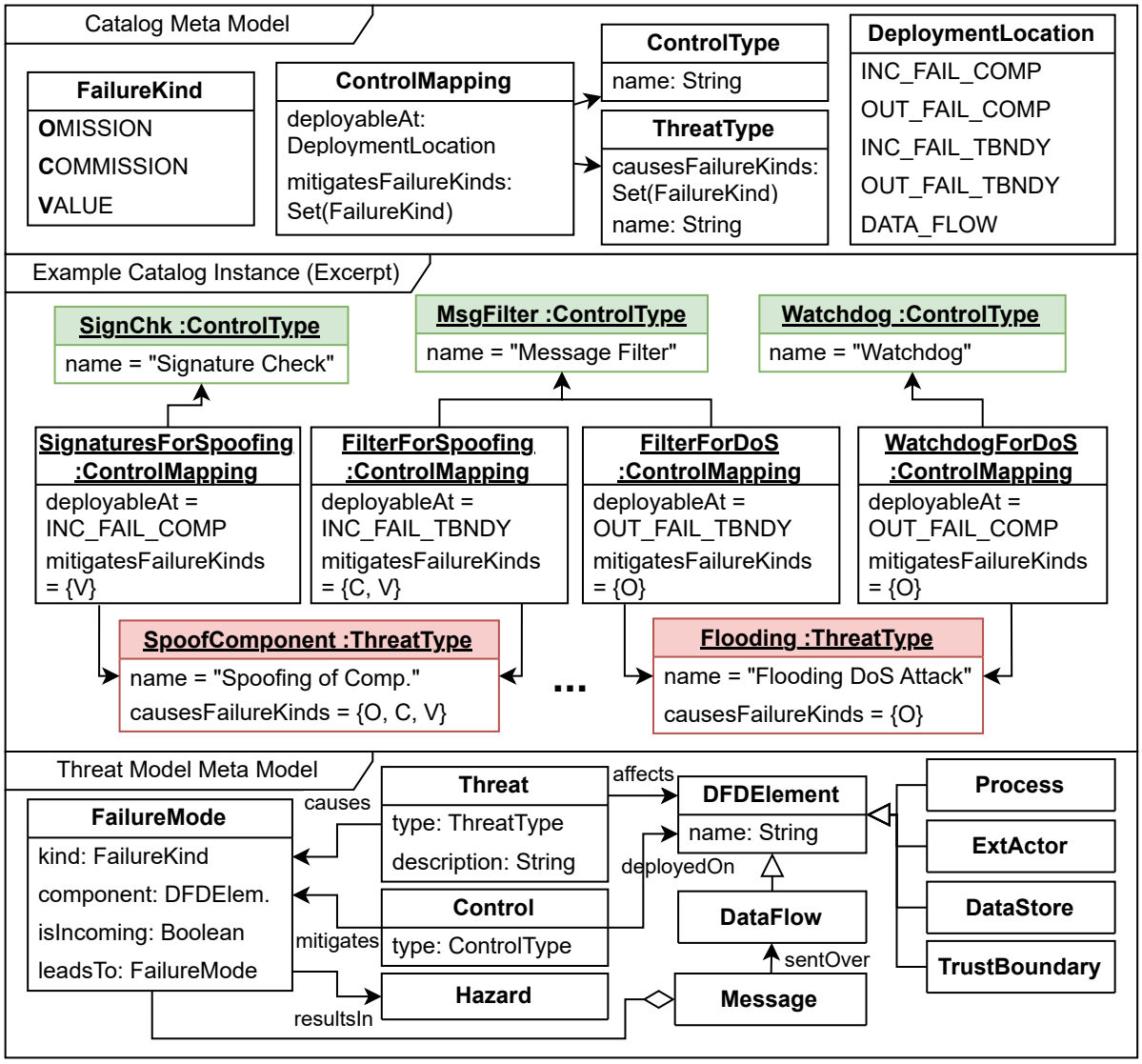

Figure 2: Meta-models for the threat and security control catalog (top) and the safety-informed threat model (bottom), as well as a sample catalog instance with potential threats, controls, and suitable mappings (center)

messages and thus mitigate *Value* failures at the receiver. Our sample catalog further includes a *Signature Check* (cf. SignChk) to authenticate messages and a *Watchdog* to restart unresponsive components.

To map ControlTypes to corresponding ThreatTypes, the catalog meta-model defines a ControlMapping. The mitigatesFailureKinds attribute of the ControlMapping allows to specify which (subset of) failure kinds caused by the corresponding threat the control is able to mitigate. For example, as our sample message filter ControlType is able to detect and block spoofed messages, it can mitigate *Commission* (fake messages being sent inadvertently) and *Value* (messages being sent with manipulated payload) failures of corresponding messages, but it cannot detect, if a legitimate message was omitted. Correspondingly, the FilterForSpoofing mapping specifies that the MsgFilter is able to mitigate C and V failures from the set of caused failure kinds of the SpoofComponent ThreatType. OCL rules can be used to enforce that only failure kinds that are actually caused by the ThreatType are mitigated by the ControlType. Since the failure kinds that a control is able to mitigate depends on where it was placed, we have included the mitigatesFailureKinds attribute in the ControlMapping instead of the ControlType itself.

To calculate suggestions for controls at suitable locations in the system architecture, the ControlMapping allows to define a desired deployment location for a

control through the deployableAt attribute. We identified five different possible deployment locations for a control, relative to the location where the threat is annotated in the system architecture:

INC_FAIL_COMP Each receiving node at which compromised messages arrive, i.e., components that contain the *incoming failure modes* for failures caused by the specific threat.

OUT_FAIL_COMP The sending node of message(s) that are affected by the threat, i.e., the component containing the *outgoing failure mode* that is linked to the threat.

INC_FAIL_TBNDY The *trust boundary* in which the component containing the *incoming failure mode* is located in.

OUT_FAIL_TBNDY The *trust boundary* in which the component containing the *outgoing failure mode* is located in.

DATA_FLOW The control shall be deployed directly on the data flow the affected message is sent over.

Two main ideas lead to this list of deployment locations. First, *preventive* controls aim to prevent threats from occurring in the first place, e.g., a message filter that blocks a flooding attack before a component is overloaded. Second, *reactive* controls respond after attacks have already occurred and attempt to mitigate their negative effects (i.e., failures) on the victim's side, e.g., a message signature check that discards compromised messages. With this knowledge and based on the failure propagation, we can then either place controls locally at the sender (OUT_FAIL_COMP), receiver (INC_FAIL_COMP), or on the data flow itself (DATA_FLOW), or we can protect the trusted component and its surroundings, i.e., other components in the same trust boundary of the sender (OUT_FAIL_TBNDY) or receiver (INC_FAIL_TBNDY).

Control Suggestion Process. Figure 3 visualizes our proposed process for suggesting security controls. We foresee that the process will be carried out by a person with security expertise, e.g., a security engineer. However, most security expertise is needed to create and fill the catalog, which occurs less frequently than using the catalog for recommendations. During the usage phase, the process supports the user by suggesting sets of controls, from which the user then has to select suitable candidates.

To explain our proposed process and the corresponding meta-model by example, we demonstrate the effects of each process step on our running example in Figure 4.

In **Step 1**, after filling the catalog and creating the DFD with annotated hazards and threats, a failure propagation model, e.g., an SeCFT, must be calculated for the system. The result of the SeCFT analysis for our running example is shown on the left side of Figure 4. Failure modes of different kinds (e.g., ***O****mission*) are depicted as labeled triangles. Based on the failure propagation model, *Minimal Cut Sets (MCS)* can be derived, that show which attack and failure combinations have to occur in combination to cause a hazard (Haasl et al., 1981). Without any security controls in place, assume the following three MCS were calculated for our sample system (each individual cut set is contained in curly braces):
$MCS_{H1} : \{T_1, V_{radar\_data}\}$, meaning that T1 *and* a value failure of the message radar_data (at ACC) have to occur simultaneously to cause H1, as the redundant inputs used to calculate the brake_decision are cross-correlated by ACC.
$MCS_{H2} : \{T_1\}, \{T_2\}$, meaning that *either* T1 *or* T2 causes H2, as the inability to switch on the headlamp due to an overloaded BCM or a spoofed oncoming car both independently lead to a loss of headlight.

In **Step 2**, all threats from the threat model are sorted by priority. For this prioritization the MCS from Step 1 can be reused, as threats appearing in multiple MCS are potentially more critical (cf. (Steiner, 2016, p. 73)). However, we do not prescribe how threats actually have to be prioritized, as we only expect the *most critical* threat to be dealt with first, so that its controls can potentially be reused for other threats. For our running example, threat T1 appears in two of three MCS (for H1 and H2; cf. Step 1) and is therefore ordered before T2.

In **Step 3**, we select the first threat from the ordered set of unmitigated threats from Step 2 (e.g., the one that causes the most hazards) and retrieve applicable ControlTypes for it from the catalog (via the ControlMapping relation).

In **Step 4**, the ControlTypes from Step 3 are sorted by their applicability to other threats. This is done to identify controls that, depending on their deployment location, are able to mitigate other threats in the same area and therefore only need to be implemented once.

In **Step 5**, the sorted list of applicable ControlTypes for each threat is presented to the user who can select ControlTypes from the list and deploy them as actual Controls in the system architecture until they deem the corresponding threat mitigated.

In **Step 6**, the mitigated threat is removed from the set of unmitigated threats and the MCS for the system are recalculated to reflect applied controls. Afterwards, the user can continue with the next unmitigated threat. As a termination criterion for the entire process, we use the *Shortest Length of Attack (SLOA)* metric (Krautsevich et al., 2010; Bodden et al., 2024)

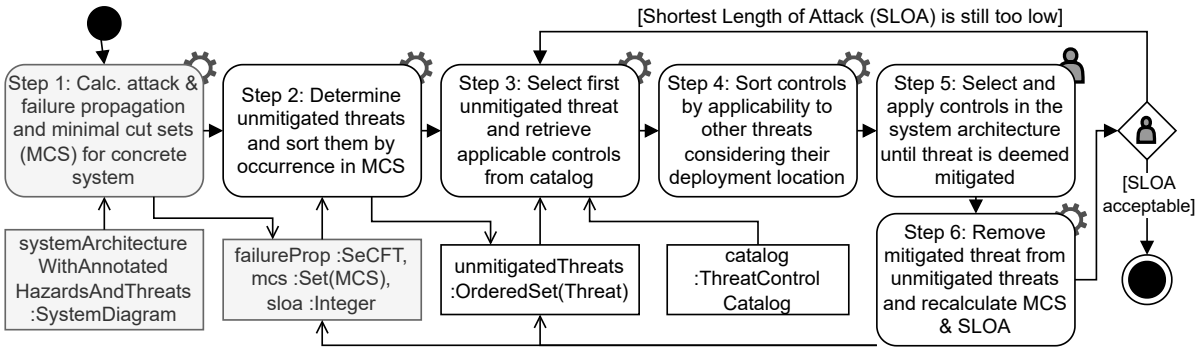

Figure 3: Process Overview: Calculating suggestions for efficient controls. Process steps are depicted as rectangles with round corners whereas process artifacts are depicted as rectangles with sharp corners. External artifacts and process steps that are out of scope for this process have a light gray background. In addition, we annotated steps that are (partly) automatable with a gear symbol and steps that need to be done manually with an icon of a human.

that describes how many actions an attacker has to perform to achieve their attack goal, e.g., causing a hazard. We adapt this metric to MCS by correlating the SLOA with the size of the smallest MCS of the system. If the SLOA is still too low, i.e., the user determines that there are too few defensive layers in the system, the process goes on with Step 3 by selecting the next threat from the set of unmitigated threats. If the user decides that the SLOA reached an acceptable level the process is finished.

4 Evaluation

This case study is based on the guidelines defined by Kitchenham et al. (Kitchenham et al., 1995) and Runeson and Höst (Runeson and Höst, 2009). Our case study investigates the applicability of our approach in practice. In preparation of the case study, we implemented a prototype of our approach in a graphical editor based on the Eclipse platform.

Case Study Context. We examine the three evaluation questions:

EQ1 Does the presented method enable security engineers to model system architectures, including hazards and threats, for realistic examples?

EQ2 Does the presented method enable security engineers to model control catalogs for realistic control and threat categories?

EQ3 Does the presented method aid security engineers in finding suitable controls for realistic examples?

To answer these questions, we specify the headlamp system presented in the ISO 21434 automotive security standard and create a control catalog based on real-world controls from MITRE.

Setting the Hypotheses. We define the following hypotheses for this case study.

HYP1 The system architecture of the headlamp system can be specified using the meta-model presented in Section 3. We consider HYP1 as fulfilled if the system architecture can be specified and if threats and hazards can be annotated using solely the meta-model presented in Section 3.

HYP2 The control catalog can be specified using the meta-model presented in Section 3. We consider HYP2 as fulfilled if the control catalog can be specified using solely the meta-model presented in Section 3.

HYP3 The automated determination of security controls is correct and complete. For evaluation HYP3, we generate the list of suggested controls based on the modeled headlamp system and the control catalog. We check if the deployment location enum entries (e.g., INC_FAIL_TBNDY), as presented in Section 3, are reasonable and complete. In addition, we manually investigate the results and check whether the suggested controls and their deployment locations are reasonable. We consider HYP3 as fulfilled if all threats are mitigated by means of reasonable controls.

Validating the Hypotheses. To validate the hypotheses, we used the prototypical implementation of our approach and applied the process presented in Section 3 to the modeled case. First, we modeled the system architecture for the headlamp system and conducted a hazard and threat analysis for the headlamp system. We annotated the resulting hazards and threats on elements of the system architecture. Second, we selected exemplary controls from MITRE and defined a control catalog. Finally, we executed the automatic determination of security controls and manually inspected the results for correctness and rea-

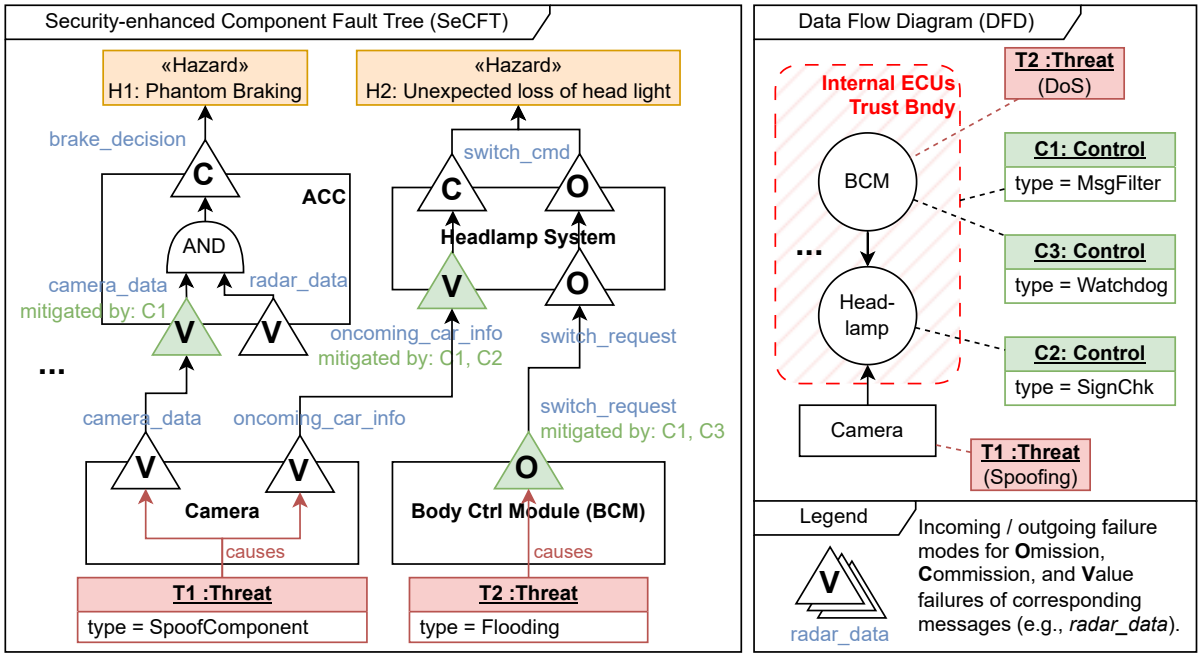

Figure 4: Failure Modes from an SeCFT (left) are used to identify and locate suitable Security Controls in the DFD (right)

sonability.

Analyzing the Results. We were able to model the system architecture for the headlamp system and annotate the results of hazard and threats analysis to elements of the system architecture (cf. Figure 1) using only modeling elements from the meta-model presented in Section 3. In addition, we were able to model a control catalog based on real-world controls from MITRE using only modeling elements from the meta-model presented in Section 3. Thus, we conclude that **HYP1** and **HYP2** are fulfilled. Finally, the suggested controls (cf. Figure 4) mitigate all threats contained in the system architecture and their locations were reasonable. Thus, we conclude that **HYP3** is fulfilled.

Concluding the case study, the fulfilled hypotheses indicate that our approach and the corresponding meta-models are applicable and provide reasonable results for real-world examples.

Threats to Validity. We see the following threats to validity:

First, we only considered the headlamp system as a use case for our case study and, thus, cannot generalize the fulfillment of the hypotheses for all possible systems. Nevertheless, the selected headlamp system is used as a reference system in the ISO 21434. Moreover, the controls used to model the control catalog have been selected from MITRE, which presents a commonly used control catalog. Thus, we do not expect large deviations for other automotive examples and further controls.

Second, the case study was designed and conducted by the same researcher who developed the approach. Since the researcher might be biased toward the approach, the case study would be more significant if security engineers had modeled the use case. To mitigate this, we discussed our approach with security experts from academia and security engineers from industry. Moreover, we discussed the case study design and its evaluation questions with other researchers.

5 Related Work

Most related approaches (e.g., (Fila and Wideł, 2020; Berro et al., 2019)) use variants of *attack (defense) trees* (Kordy et al., 2014) with annotated countermeasures to formally compute optimal sets of controls, sometimes extended with safety aspects and dependencies (Soltani et al., 2024). A key limitation is that they typically determine *which* controls to select, but not *where* to deploy them in the system architecture. Controls are usually attached to leaf nodes, so each control mitigates a specific threat and reuse across multiple threats is not explicitly supported.

Roy et al. (Roy et al., 2010) allow countermeasures to be assigned also to intermediate nodes, enabling reuse in principle. However, their analysis still focuses on attack steps of a single threat, rather than

identifying a suitable set of measures for many threats in a complete system. In contrast, our approach directly links control selection and deployment within the threat model by combining SeCFTs (attack and failure propagation) with DFDs (e.g., trust boundaries) to identify deployment locations and thereby support secure design.

Dantas and Nigam (Dantas and Nigam, 2023) use a *Knowledge Representation and Reasoning (KRR)*-based mechanism to propose safety and security measures at architectural locations based on attacker reachability. They group attacks by type (e.g., integrity) rather than distinguishing specific threats and their controls. Our approach instead considers which failures are caused by concrete catalog threats and how these failures propagate, allowing more precise placement suggestions.

6 Conclusion

Cyber attacks can lead to safety hazards, particularly in automotive or industrial control systems. However, identifying suitable security controls to protect these systems can be challenging due to the complexity and interdependencies of threats. This paper presented a novel approach for strategically suggesting and selecting sets of security countermeasures to implement a defense-in-depth strategy in safety-critical systems. The proposed approach utilizes failure and attack propagation information to suggest efficient security controls, emphasizing multiple complementary layers of defense to address potential threats effectively. The suggestion mechanism further respects potential deployment locations of controls to enhance control reusability. The approach was evaluated using a real-world case study, showing promising results in terms of identifying effective security measures.

By focussing on threats that cause omission, commission, or value failures of messages, our approach is currently limited to suggest controls only for threats that have a safety impact. We consider threats that, for example, target confidentiality out of scope for our approach for now. Nevertheless, it would probably be possible to also take into account the effects of these threats by adding an fictional confidentiality failure type to the list. Investigating this is a potential direction for future work.

REFERENCES

Berro, S., Apvrille, L., and Duc, G. (2019). Optimizing system architecture cost and security countermeasures. In *International Workshop on Graphical Models for Security*, pages 50–67. Springer.

Bodden, E., Pottebaum, J., Fockel, M., and Gräßler, I. (2024). Evaluating security through isolation and defense in depth. *IEEE Security & Privacy*, 22(1):69–72.

Dantas, Y. G. and Nigam, V. (2023). Automating safety and security co-design through semantically rich architecture patterns. *ACM Transactions on Cyber-Physical Systems*, 7(1):1–28.

Fila, B. and Wideł, W. (2020). Exploiting attack–defense trees to find an optimal set of countermeasures. In *2020 IEEE 33rd computer security foundations symposium (CSF)*, pages 395–410. IEEE.

Fockel, M., Schubert, D., Trentinaglia, R., Schulz, H., and Kirmair, W. (2022). Semi-automatic integrated safety and security analysis for automotive systems. In *10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*.

Haasl, D. F., Roberts, N. H., Vesely, W. E., and Goldberg, F. F. (1981). Fault tree handbook. Technical report, Nuclear Regulatory Commission.

Kitchenham, B., Pickard, L., and Pfleeger, S. L. (1995). Case studies for method and tool evaluation. *IEEE software*, 12(4):52–62.

Kordy, B., Mauw, S., Radomirović, S., and Schweitzer, P. (2014). Attack–defense trees. *Journal of Logic and Computation*, 24(1):55–87.

Krautsevich, L., Martinelli, F., and Yautsiukhin, A. (2010). Formal approach to security metrics. what does" more secure" mean for you? In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pages 162–169.

Roy, A., Kim, D. S., and Trivedi, K. S. (2010). Cyber security analysis using attack countermeasure trees. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–4.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164.

Soltani, R., Lopuhaä-Zwakenberg, M., and Stoelinga, M. (2024). Safety-security analysis via attack-fault-defense trees: Semantics and cut set metrics. In *International Conference on Computer Safety, Reliability, and Security*, pages 218–232. Springer.

Sommer, F., Dürrwang, J., and Kriesten, R. (2019). Survey and classification of automotive security attacks. *Information*, 10(4):148.

Steiner, M. (2016). *Integrating Security Concerns into Safety Analysis of Embedded Systems Using Component Fault Trees*. PhD thesis, Technische Universität Kaiserslautern.

Stytz, M. R. (2004). Considering defense in depth for software applications. *IEEE Security & Privacy*, 2(1):72–75.