Automating Vehicle SOA Threat Analysis Using a Model-Based Methodology

Yuri Gil Dantas[1], Simon Barner[1], Pei Ke[2], Vivek Nigam[2] and Ulrich Schöpp[1]
[1]*fortiss GmbH, Munich, Germany*
[2]*Huawei Technologies Düsseldorf GmbH, Düsseldorf, Germany*

Keywords: Automotive, Threat Analysis, Service-Oriented Architectures, Automation, Safe and Secure-by-Design.

Abstract: This article proposes automated methods for threat analysis using a model-based engineering methodology that provides precise guarantees with respect to safety goals. This is accomplished by proposing an intruder model for automotive SOA which together with the system architecture and the loss scenarios identified by safety analysis are used as input for computing assets, impact rating, damage/threat scenarios, and attack paths. To validate the proposed methodology, we developed a faithful model of the autonomous driving functions of the Apollo framework, a widely used open source autonomous driving stack. The proposed machinery automatically enumerates several attack paths on Apollo, including attack paths not reported in the literature.

1 INTRODUCTION

The automotive industry is under great transformation to meet challenges of implementing features such as Autonomous Driving and Over-the-Air Updates. Instead of using distributed architectures with domain-specific hardware, vehicles are using software-intensive *Service-Oriented Architectures* (SOA) with powerful centralized computer units. The open-source Apollo framework (Apollo, 2021) is an example of this transformation providing autonomous vehicle features that have been used in the development of real-world autonomous vehicle applications, such as autonomous taxis and buses.

This transformation has also increased concerns on how attackers can affect road-user safety. While security threats to safety have been known for more than a decade ago (WIRED, 2015), the upcoming/recent standards ISO 21434 (ISO/SAE 21434, 2020) and the UNECE (UN, 2021) have pushed industry to change its development process to enable safe and secure-by-design vehicles. For example, the ISO 21434 puts great emphasis on the development process and on the threat analysis, e.g., Damage/Threat Scenario/Attack Path enumeration, that shall be performed and addressed before putting the vehicle on the road. At the end, Original Equipment Manufactures (OEMs) shall provide compelling arguments and evidence, i.e., an assurance case, that their vehicles are safe also from a security perspective.

OEMs may pay a costly price if they develop autonomous vehicle features without previously producing analysis, argument, and evidence supporting vehicle safety and security. Without these artifacts, it is hard to expect that these vehicles will be accepted by certification agencies and be allowed to be used in several countries, once standards are more heavily enforced. Even more troublesome is that several attacks have been reported that can cause serious hazards to road-users, such as vehicle collisions. *As we claim here, many of these attacks could have been identified during the design of the system architecture by using a safe and secure-by-design approach with suitable threat analysis supported by automation.*

A key challenge for the development of safe and secure-by-design vehicles is handling the enormous complexity involved. For example, without adequate countermeasures, SOA allows any software component to publish any data including data that may be consumed by safety-critical functions. This has been a source of, e.g., overprivilege attacks (Hong et al., 2020) causing hazardous situation whenever a safety-critical function consumes data erroneously published by a malicious component (or even by a faulty component). For another example, malicious components may exploit SOA communication vulnerabilities to cause man-in-the-middle attacks (Zelle et al., 2021). Moreover, sensors, such as cameras and GPS radios,

Dantas, Y., Barner, S., Ke, P., Nigam, V. and Schöpp, U.
Automating Vehicle SOA Threat Analysis Using a Model-Based Methodology.
DOI: 10.5220/0011786400003405
In *Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP 2023)*, pages 180-191
ISBN: 978-989-758-624-8; ISSN: 2184-4356
Copyright © 2023 by SCITEPRESS – Science and Technology Publications, Lda. Under CC license (CC BY-NC-ND 4.0)

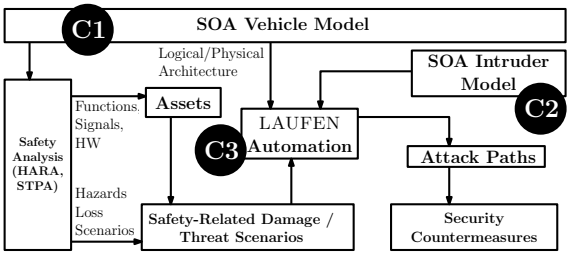

Figure 1: Illustration of the proposed Safe and Secure-by-Design methodology, tool-chain and key contributions (C1, C2 and C3).

are attack surfaces that may be exploited by attackers to cause hazards (Jha et al., 2020; Shen et al., 2020).

1.1 Safe and Secure-by-Design Methodology and Contributions

The proposed safety and security methodology and three key contributions are depicted in Figure 1. The methodology is built upon the following key ideas from the automotive safety and security co-engineering literature:

- Analysis Techniques for Software-Intensive Systems: System Theoretic Process Analysis (STPA) (Leveson and Thomas, 2018) has been recommended for safety analysis of autonomous functions by standards such as the ISO 21448 (Safety Of The Intended Functionality – SOTIF) (SOTIF, 2021) for assuring the safety of features such as autonomous driving. This is because STPA does not assume linear causal dependency and rather puts a greater emphasis on the faulty/malicious component interactions.
- Safety to Security: The approach recommended by Bosch engineers (Förster et al., 2019) uses safety artifacts, e.g., safety goals and hazards, as inputs to security analysis. There are two key motivations for this: 1) A safety analysis is typically carried out before a security analysis. 2) By using safety as input to security, one can claim, through appropriate traceability, completeness of security analysis w.r.t. to the results of the safety analysis. This is done, for example, by checking whether all causes of hazards (called loss scenarios in STPA terminology) have traces to appropriate security analysis.
- Model-Based Tool-Chains: Model-based engineering approaches are based on formal abstractions of the system under design and therefore help mitigate the complexity of nowadays software and hardware architectures and to boost development speed and quality when compared to traditional document-based approaches by means of automated analysis, design and validation tools.

While these methods have been proposed, this article is the first to apply them together into an overarching model-based methodology for SOA vehicle architectures. As depicted in Figure 1, we start from a (SOA) Vehicle Model, specifying the key functions, logical components, and platform (a.k.a. physical) architecture. These model elements ensure the soundness of the approach, as the safety and security analysis that follow are traced to the model. From the Hazard Analysis and Risk Asessment (HARA) and STPA analysis, key safety functions, channels and physical elements are identified, which are then traced as assets from the security perspective that need to be protected. Loss scenarios obtained from STPA, i.e., the situations that may lead to hazards, are traced to damage and threat scenarios specifying how intruders can cause safety hazards. From this point onward, we carry out a security analysis, e.g., using the logical and platform architectures to identify attack paths that can cause threat scenarios. Ultimately, we discuss potential countermeasures to address threats.

The key benefits of the approach are three-fold: The first benefit is a full traceability between safety and security analysis and the vehicle model. This means that the analysis is reflected in the actual implementation that will be deployed in the vehicle. The second benefit is that the methodology provides guarantees that all loss scenarios for all hazards are considered by the security analysis, e.g., all loss scenarios are traced to damage/threat scenarios. This means that all identified safety issues shall be considered from the security perspective. The third benefit is that our model-based methodology enables the use of automated methods, e.g., the automated enumeration of attack paths based on intruder models.

The main contributions of this article are:

- **Apollo-Based Vehicle Model (C1):** By examining the relevant pieces of code in the Apollo code-base related to autonomous driving functions, we designed a faithful vehicle model. The model reflects the SOA publish and subscribe pattern, and the information (namely the topics) between the Apollo components. To the best of our knowledge, it is the first model based on the Apollo v7.0.0 code base.
- **Intruder Model for Vehicle SOA (C2):** By examining vehicle SOA security literature, we formalized an intruder model for vehicle SOA. The intruder is capable of carrying out Man-in-the-Middle (MITM) attacks, and carrying out spoofing attacks by infiltrating the system

from public interfaces to, e.g., exploit perception sensors, such as LiDAR and Camera.

- **Attack Path Automation (C3):** We developed a machinery (LAUFEN) to automate the enumeration of attack paths on the vehicle system architecture. LAUFEN takes as input the model, assets, damage/threat scenarios, and the implementation of the intruder model, and outputs all attack paths.

We demonstrate and validate our approach and automation on the developed Apollo Vehicle Model. Our focus is on safety assets as it is the main concern for autonomous driving. The developed machinery identified **246** attack paths. The attack paths include attacks that have been reported in the literature. Given the traceability to safety analysis, our machinery identifies a much greater number of attack paths that would need to be mitigated (or for which some security rational shall be provided) by security countermeasures. Indeed, based on the generated attack paths, we identified potential attacks that have not yet been reported.

Structure of This Article: Section 2 describes the Apollo vehicle model. Section 3 describes how to trace assets, damage/threat scenarios to artifacts produced by safety analysis. Sections 4 and 5 describe the intruder model and the attack paths generated, including its validation using the Apollo model. After discussing related and potential future work in Section 6, we conclude in Section 7.

2 APOLLO MODELING

Apollo (Apollo, 2021) is an open-source autonomous driving stack enabling highly autonomous vehicle features (more precisely, at Level 4 in the SAE ranking (sae, 2018)), such as Highway and Traffic Jam Pilots, where a vehicle can drive with limited human supervision. Apollo v7.0.0 (Apollo, 2021) consists of more than 500k lines of `C++` code.

A central part of the Apollo implementation is the Cyber RT middleware (cyb, 2019). Cyber RT provides a publish/subscribe pattern to enable the communication between software components running over it. Components can communicate via tagged channels, a.k.a. topics. Components may publish data to topics by writing messages to a named topic and may subscribe to any topic of interest by referring to the topic name. Whenever a publisher writes data to a topic, this data is received by all subscribers. Cyber RT allows more than one component to publish data on a topic, and more than one component to subscribe to it. The announcement of (new) topics and the subscription of components to topic names are performed by a mechanism called service discovery.

This section describes the designed Apollo model used to demonstrate our methodology. The model focuses on the parts of the code-base that are related to autonomous vehicle features, namely, sensors (Camera, LiDAR), localization, perception, prediction, planning, control, and HMI.

The Apollo system architecture has been modeled in the model-based system engineering tool AutoFOCUS3 (fortiss GmbH, 2022). The model comprises of **9** functions, **61** logical components, **341** ports, transmitting **73** data structures with **361** members, **16** execution units, **12** transmission units, and **6** sensors. We developed an experimental metamodel (Aravantinos et al., 2015) extension in AutoFOCUS3 to describe publish/subscribe communication by means of dedicated *topic* port data types. Due to the lack of space, the remainder of this section describes only selected parts of the logical and platform architecture, since the security results presented in this article mainly focus on the logical architecture and platform architecture.

Logical Architecture. The designed logical architecture is complex and consists of four hierarchical levels with multiple components. Figure 2 depicts the second highest level of our Apollo model containing the main autonomous driving components.

The **localization** component receives sensor data from GNSS and computes the vehicle's position. The vehicle's position is received by the following components. The **perception** component receives sensor data from cameras, radars and LiDAR, and the vehicle's position. Perception identifies obstacles, such as other vehicles on the road, as well as the state of traffic lights. The **prediction** component takes the list of obstacles from perception and the vehicle's position, and tries to predict the intention of obstacles, which may be other vehicles or pedestrians. The prediction includes aspects such as whether a vehicle intends to change lanes. The **relative map** component aggregates the list of obstacles and combines it with map data, which contains information about the road, such as lanes and traffic lights. The **planning** component takes as input all the data computed by localization, perception, prediction and relative map. Planning uses this data to plan a safe and comfortable trajectory for the vehicle. The **control** component receives the planned trajectory and produces control commands (steering, acceleration, etc.) for the vehicle to follow the trajectory.

A key challenge was to ensure the faithfulness of the model to the Apollo code. To accomplish this, we

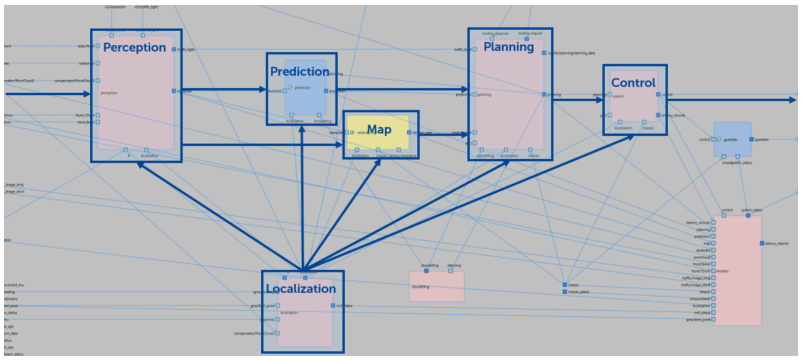

Figure 2: Logical architecture: Main autonomous driving components.

extracted the model elements by manually inspecting the Apollo code. For example, to find all Cyber RT components implemented in Apollo, we inspect the code to find all implementations of the `class cyber::Component`. The next step was to identify the topics and which components publish to them and subscribe to them. The Apollo implementation specifies the topic communication using the following mechanisms: DAG configuration files, C++ code implementing readers for topics and producers of topics, and library code. We inspected each of these mechanisms to map the topics that are subscribed and published to components.

Platform Architecture. Figure 3 illustrates our platform architecture that follows the trend for modern smart car architectures consisting of a few, but powerful ECUs and using network interfaces (i.e., switches) between ECUs.

The main ECUs in the platform architecture are: (1) **MDC:** Mobile Data Center: This hardware is responsible for the autonomous function related components, such as inferring objects from camera input, predicting the movement of objects in the environment, planning trajectories. The MDC is further sub-divided into sub-systems with different types of processing units with different levels of safety assurance levels, such as an ASIL-D MCU. (2) **CDC:** Intelligent Cockpit: This hardware is responsible for all the cockpit related functions, such as driver monitoring systems and entertainment functions. (3) **VDC:** Vehicle Controller: This hardware is responsible for the basic vehicle control functions, such as Electric Power Steering, Battery Management, and Anti-lock Braking functions. (4) **VIU 1-4:** Vehicle Integration Units: These hardware are powerful gateways that interface the MDC, CDC, and VDC, connected through network interfaces, to the domain specific hardware connected through CAN buses.
The yellow shade in the model represents the system boundary (a.k.a. item boundary). We consider as part of the system all components that are implemented in the Logical Architecture. For example, Sensors (e.g., LiDAR and GPS radio) are not part of the system itself. They are third-party devices that are connected to the system and provide inputs from the environment. We consider them as public interfaces that are outside and may be accessed by external users.

3 SAFETY-INFORMED SECURITY ANALYSIS

Our main focus is to identify assets, damage and threat scenarios related to safety as it is the main concern to autonomous driving. We describe how key safety artifacts are consumed by security analysts to identify key security artifacts, establishing a traceability between security and safety concerns. One can argue from such traces the (relative) completeness with respect to safety of the security analysis in the sense that threats that can cause any one of the safety loss scenarios are identified. As there is existing literature that advocates similar traceability between safety and security (Förster et al., 2019; Dantas and Nigam, 2022a), albeit not using loss scenarios and artifacts mentioned in the ISO 21434 (ISO/SAE 21434, 2020), we simply exemplify the method on examples using the Apollo system architecture. Our mapping from safety artifacts to security artifacts is inspired by the following work (Sabaliauskaite et al.,)(Macher et al., 2015)(Förster et al., 2019).

Safety Analysis. We carried out a safety analysis for the Apollo system architecture. In compliance to ISO 26262-3 (ISO26262, 2018), we have identified hazards by using Hazard Analysis and Risk Assessment (HARA). Furthermore, we use System Theoretic Process Analysis (STPA) (Leveson and Thomas, 2018) to identify how such hazards may occur.

Relevant for this article are hazards and loss

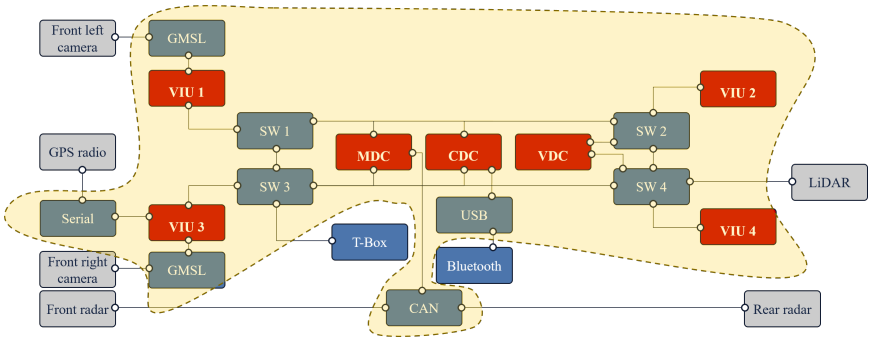

Figure 3: Platform architecture based on modern smart car architecture (yellow shading represents the system boundary).

Table 1: Example of safety analysis results.

Hazard HZ1	Unintended distance between the ego vehicle and other objects.		
Severity	Life-Threatening (S3)	**Safety Risk Level:** ASIL D	
Exposure	High Probability (E4)		
Controllability	Difficult to Control (C3)		
Loss Scenario LS1 that causes Hazard HZ1			
Source	**Target**	**Message**	**Failure Mode**
planning	control	trajectory	erroneous

scenarios provided, respectively, by HARA and STPA. A *hazard* is a potential source of loss (e.g., loss of life) caused by malfunctioning behavior of the item (i.e., Apollo system architecture). A *loss scenario* describes the casual factors that may lead to a hazard.

We have identified **4** hazards and **21** loss scenarios. We will use the hazard (HZ1) and loss scenario (LS1) described in Table 1 to demonstrate the model-based methodology for threat analysis described in Section 1.1. HZ1 is a high risk level (ASIL D) related to the autonomous driving functions. LS1 is a possible cause for HZ1. LS1 is traced to two components in the model, planning and control, and to the topic containing the trajectory produced by planning. LS1 specifies that if the computed trajectory is erroneous, e.g., instead of recommending a low acceleration, it recommends a right acceleration, HZ1 may occur, i.e., the vehicle may collide with obstacles.

Assets and Damage/Threat Scenarios from Safety Analysis. Following the ISO 21434 (ISO/SAE 21434, 2020), *assets* are objects (e.g., software components, hardware units) for which the compromise of its cybersecurity property can lead to the damage of the item. A *damage scenario* denotes the adverse consequence due to the compromise of a cybersecurity property of an asset. A *threat scenario* denotes the potential actions (or simply attack) on assets that can lead to damage scenarios. The hazards and loss scenarios obtained from the safety analysis can be directly used to identify such security artifacts related to safety-related damages.

Damage scenarios are traced to hazards. The damage scenario traced to HZ1 specifies that unintended distance shall be avoided also from a security perspective. There are three main **assets** that can be traced to the loss scenarios: *Safety Functions:* The safety related functions (typically implemented as pieces of software) shall be protected. For LS1, the functions planning and control are such assets. *Topic/Messages:* The safety-related signals/messages mentioned in the loss scenarios shall be protected. For LS1, the topic carrying the trajectory information shall be protected. *Hardware/Physical:* The hardware in which safety functions are deployed shall be protected. The functions associated to LS1 are deployed at the MCU (inside of MDC) hardware unit. Moreover, the failure mode of loss scenarios indicates which cyber-security properties (CIA properties) are associated to the these assets. The failure mode erroneous and loss indicate, respectively, that the integrity and availability of the corresponding assets shall be ensured. Notice that the confidentiality property cannot be extracted from safety analysis as lack of confidentiality does not lead to safety-related damages. From the loss scenario and its derived assets, one can elaborate **threat scenarios** by using, e.g., the STRIDE methodology (Shostack, 2014). For example, the integrity of safety functions and of physical assets can be violated by tampering attacks,

while the integrity of topic/messages can be violated by spoofing and elevation of privilege.

These artifacts are used to enumerate attack paths that shall be considered, namely those that can lead to threat scenarios. The enumeration of attack paths depends on the technology that is being used. For example, if a software may be updated using Over-the-Air mechanisms, then attack paths shall consider how these mechanisms can be exploited to tamper the software with malicious updates. For the Apollo system architecture considered in this article, one needs to consider the use of SOA machinery, e.g., protocols for service discovery, publish-subscribe communication patterns, sensors, and other public interfaces, e.g., Bluetooth and WiFi. These are considered in the next section.

4 INTRUDER MODEL FOR VEHICLE SOA

We formalize an SOA intruder model defined by the rules in Figure 4. The intruder model is based on the main attacks against vehicle SOA with centralized architecture, described in Section 6. Intuitively, SOA contain two main attack surfaces that may be exploited if no suitable countermeasures are deployed.

- *Outsider Attackers* can exploit public interfaces, such as sensors and communication interfaces, to infiltrate the system and attack vehicle assets, such as safety functions. For example, attackers can spoof GPS coordinates thus violating the integrity of published position information by localization.
- *Insider Attackers* can exploit vulnerabilities in the underlying SOA protocols and carry out MITM attacks thus violating the integrity of topics. For example, attackers can carry out MITM attacks between localization and perception to violate the integrity of position information.

Figure 4 introduces the rules of the intruder model reflecting these type of attacks. These inference rules derive three judgments described below. Γ contains system specifications which are extracted from the vehicle model. These specifications are formalized as atomic formulas using the predicate symbols described in Table 2.

$\Gamma \vdash \mathsf{wrt}(X,Y)$ and $\Gamma \vdash \mathsf{rd}(X,Y)$ denote that the port X of model element may write, respectively, read on Y. Rule write_1 specifies that an output eo of an ECU may write on an input port ni of a network element if an output port co of a component is allocated to eo (specified by $\mathsf{cpo}(\mathsf{c},\mathsf{co}), \mathsf{alloc}(\mathsf{co},\mathsf{eo})$), and there is a channel from eo to ni (specified by $\mathsf{ch}(\mathsf{eo},\mathsf{ni})$). Rule

Table 2: Description of the predicates used to define the intruder's capabilities.

Predicate	Denotation
$\mathsf{ecui}(\mathsf{ecu},\mathsf{ei})$	ECU ecu and its input port ei.
$\mathsf{ecuo}(\mathsf{ecu},\mathsf{eo})$	ECU ecu and its output port eo.
$\mathsf{neti}(\mathsf{net},\mathsf{ni})$	net. interface net and its input port ni.
$\mathsf{neto}(\mathsf{net},\mathsf{no})$	net. interface net and its input port no.
$\mathsf{ch}(\mathsf{out},\mathsf{inp})$	channel from output port out to input port inp.
$\mathsf{wrt}(\mathsf{el1},\mathsf{el2})$	element el1 writes data to el2.
$\mathsf{rd}(\mathsf{el1},\mathsf{el2})$	element el1 reads data from el2.
$\mathsf{cpi}(\mathsf{c},\mathsf{ci})$	component c and its input port ci.
$\mathsf{cpo}(\mathsf{c},\mathsf{co})$	component c and its output port co.
$\mathsf{alloc}(\mathsf{el},\mathsf{ecu})$	element el is allocated to ecu.
$\mathsf{pub}(\mathsf{c},\mathsf{co},\mathsf{tp})$	component c publishers the topic tp through output port co.
$\mathsf{sub}(\mathsf{c},\mathsf{ci},\mathsf{tp})$	component c subscribers to the topic tp through input port ci.
$\mathsf{if}(\mathsf{ecu},\mathsf{ci},\mathsf{tp})$	topic tp is published within ecu through an information flow from ci.
$\mathsf{pro}(\mathsf{tp})$	topic tp is protected by a cryptographic primitive.
$\mathsf{publico}(\mathsf{el},\mathsf{po})$	public el and its output port po.
$\mathsf{i_reach}(\mathsf{el})$	element el is reachable by the intruder.
$\mathsf{i_attack}(\mathsf{el})$	el may be attacked by the intruder.

write_4 is similar, but for public elements. Rule write_2 specifies that an input port of an ECU may write to its own output port – we assume that there exists an internal transmission within the ECU ($\mathsf{ch}(\mathsf{ei},\mathsf{eo})$), e.g., components exchanging messages within the ECU. Rule write_3 is similar, but for network interfaces. Rule read_2 specifies when an ECU reads from a network interface (similar to write_1). Rule read_1 specifies that subscriber ports may read from publisher ports.

$\Gamma \vdash \mathsf{i_reach}(X)$ denotes when a port X of a model element is reachable by an intruder. Rule basic_out specifies that any port of a public element in the architecture can be reached by the (outsider) intruder. Rule reach_wrt specifies that a port p2 of a model element can be reached by the (outsider) intruder if a port p1 writes on p2. Respectively, reach_rd specifies

Write and Read Rules

$$\frac{\mathsf{cpo}(\mathsf{c},\mathsf{co}),\mathsf{alloc}(\mathsf{co},\mathsf{eo}),\mathsf{ecuo}(\mathsf{ecu},\mathsf{eo}),\mathsf{neti}(\mathsf{net},\mathsf{ni}),\mathsf{ch}(\mathsf{eo},\mathsf{ni}) \in \Gamma}{\Gamma \vdash \mathsf{wrt}(\mathsf{eo},\mathsf{ni})}\ \mathsf{write}_1$$

$$\frac{\mathsf{ecui}(\mathsf{ecu},\mathsf{ei}),\mathsf{ecuo}(\mathsf{ecu},\mathsf{eo}),\mathsf{ch}(\mathsf{ei},\mathsf{eo}) \in \Gamma}{\Gamma \vdash \mathsf{wrt}(\mathsf{ei},\mathsf{eo})}\ \mathsf{write}_2 \qquad \frac{\mathsf{neti}(\mathsf{net},\mathsf{ni}),\mathsf{neto}(\mathsf{net},\mathsf{no}),\mathsf{ch}(\mathsf{ni},\mathsf{no}) \in \Gamma}{\Gamma \vdash \mathsf{wrt}(\mathsf{ni},\mathsf{no})}\ \mathsf{write}_3$$

$$\frac{\mathsf{publico}(\mathsf{el},\mathsf{po}),\mathsf{neti}(\mathsf{net},\mathsf{ni}),\mathsf{ch}(\mathsf{po},\mathsf{ni}) \in \Gamma}{\Gamma \vdash \mathsf{wrt}(\mathsf{po},\mathsf{ni})}\ \mathsf{write}_4 \qquad \frac{\mathsf{sub}(\mathsf{c1},\mathsf{ci},\mathsf{tp}),\mathsf{pub}(\mathsf{c2},\mathsf{co},\mathsf{tp}) \in \Gamma}{\Gamma \vdash \mathsf{rd}(\mathsf{ci},\mathsf{co})}\ \mathsf{read}_1$$

$$\frac{\mathsf{cpi}(\mathsf{c},\mathsf{ci}),\mathsf{alloc}(\mathsf{ci},\mathsf{ei}),\mathsf{ecui}(\mathsf{ecu},\mathsf{ei}),\mathsf{neto}(\mathsf{net},\mathsf{no}),\mathsf{ch}(\mathsf{no},\mathsf{ei}) \in \Gamma}{\Gamma \vdash \mathsf{rd}(\mathsf{ei},\mathsf{no})}\ \mathsf{read}_2$$

Intruder Reachability Rules

$$\frac{\mathsf{publico}(\mathsf{el},\mathsf{po}) \in \Gamma}{\Gamma \vdash \mathsf{i_reach}(\mathsf{po})}\ \mathsf{basic_out} \qquad \frac{\mathsf{pub}(\mathsf{c},\mathsf{co},\mathsf{tp}) \in \Gamma}{\Gamma \vdash \mathsf{i_reach}(\mathsf{co})}\ \mathsf{basic_ins}$$

$$\frac{\Gamma \vdash \mathsf{wrt}(\mathsf{p1},\mathsf{p2}) \quad \Gamma \vdash \mathsf{i_reach}(\mathsf{p1})}{\Gamma \vdash \mathsf{i_reach}(\mathsf{p2})}\ \mathsf{reach_wrt} \qquad \frac{\Gamma \vdash \mathsf{rd}(\mathsf{p2},\mathsf{p1}) \quad \Gamma \vdash \mathsf{i_reach}(\mathsf{p1})}{\Gamma \vdash \mathsf{i_reach}(\mathsf{p2})}\ \mathsf{reach_rd}$$

$$\frac{\mathsf{pub}(\mathsf{c},\mathsf{co},\mathsf{tp}),\mathsf{sub}(\mathsf{c},\mathsf{ci},\mathsf{tp1}),\mathsf{pub}(\mathsf{c1},\mathsf{co1},\mathsf{tp1}) \in \Gamma \quad \Gamma \vdash \mathsf{rd}(\mathsf{ci},\mathsf{co1}) \quad \Gamma \vdash \mathsf{i_reach}(\mathsf{co})}{\Gamma \vdash \mathsf{i_reach}(\mathsf{ci})}\ \mathsf{reach_ins_rd}$$

Intruder Attack Rules

$$\frac{\mathsf{if}(\mathsf{ecu},\mathsf{p},\mathsf{tp}) \in \Gamma \quad \Gamma \vdash \mathsf{i_reach}(\mathsf{p})}{\Gamma \vdash \mathsf{i_attack}(\mathsf{tp})}\ \mathsf{at_out}$$

$$\frac{\mathsf{sub}(\mathsf{c1},\mathsf{ci},\mathsf{tp}),\mathsf{pub}(\mathsf{c},\mathsf{co},\mathsf{tp}),\neg\mathsf{pro}(\mathsf{tp}) \in \Gamma \quad \Gamma \vdash \mathsf{i_reach}(\mathsf{ci}) \quad \Gamma \vdash \mathsf{i_reach}(\mathsf{co})}{\Gamma \vdash \mathsf{i_attack}(\mathsf{tp})}\ \mathsf{at_ins}$$

Figure 4: Intruder model for SOA.

that a port p2 of a model element can be reached by the (outsider) intruder if p2 reads on a port p1. Rule basic_ins specifies that any publisher port in the architecture can be reached by the (insider) intruder. Rule reach_ins_rd specifies that the (insider) intruder can reach a subscriber port ci if ci reads on a reached publisher port co.

$\Gamma \vdash \mathsf{i_attack}(X)$ denotes when a topic X can be attacked. Rule at_out specifies that any topic published within an information flow ($\mathsf{if}(\mathsf{ecu},\mathsf{p},\mathsf{tp})$) from a reached ECU's input port may be attacked. Rule at_ins specifies that any topic between publisher and subscriber ports reached by the (insider) intruder may be attacked if the topic is not protected.

Outsider Intruder (Example). Consider the platform architecture depicted in Figure 5. The black and white circles connected to hardware units are, respectively, output and input ports. We assume that `Sensor` is a public interface. The output port `o1` of `Sensor` can be reached by the intruder based on the rule basic_out. The output port `o1` writes on the input port `i1` of the network interface `Network1`, then based on reach_wrt the intruder can reach `i1` and `o2`. We assume that the subscriber port (light blue square) of component `CP1` is allocated to the input port `i2` of `ECU1`, and that `i2` reads from `o2`. The intruder can then reach `i2` and `o3` based on reach_rd. Neither `i3` nor `o4` can be reached by the intruder. The intruder can reach `i4` as `o3` writes to `i4`. The intruder cannot reach `i5` and `o5`. Finally, an intruder may carry out, e.g., a spoofing attack from `Sensor` to violate the integrity of the topics published by either `CP1` or `CP2` since there is an information flow from `i2` (at_out).

Insider Intruder (Example). Consider the logical architecture depicted in Figure 6. The dark and light blue squares connected to components are, respectively, publisher and subscriber ports. The intruder can reach all publisher ports `o1...o6` based

on basic_ins. Based on reach_ins_rd, the intruder can reach the subscriber ports `i1...i7`, as these ports read from publishers, e.g., `i7` reads from `localization` via port `o5`. The intruder cannot reach the subscriber port `i8`, as `infotainment` is not a publisher. We assume the topics published by ports `o4` and `o5` are protected. Assume the topic published by `planning` through port `o2` is the intruder's target. As a result, the intruder has the following options to carry out MITM attacks. An attack may be carried out between `routing` and `planning` or even between `perception` and `prediction` given that the topic published by `perception` may affect the topic published by `planning`. The intruder can neither carry out attacks between `localization` and `planning` (same for `perception` and `prediction`), nor between `prediction` and `planning` since the topics are protected (at_ins). The intruder cannot carry out attacks from `infotainment`.

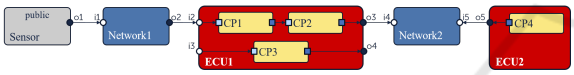

Figure 5: Illustration of the outsider intruder.

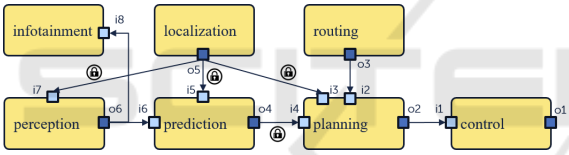

Figure 6: Illustration of the insider intruder.

Table 3: Number of identified attack paths and the execution time taken by LAUFEN to computed the attack paths.

Intruder	#Attack Paths	Execution time (s)
Outsider	152	1.11
Insider	94	0.06

5 AUTOMATING ATTACK PATH ANALYSIS

LAUFEN (vehic**L**e thre**A**t analysis a**U**tomation **F**or s**E**rvice-orie**N**ted architectures) is an SOA machinery that enables the automated computation of several activities of the Threat Assessment and Remediation Analysis (TARA) analysis. This section focuses on the automated computation of attack paths that can cause threat scenarios to vehicle SOA, i.e., the paths that violate cybersecurity properties of assets (Section 3). To this end, LAUFEN implements the proposed intruder model in the logic programming tool DLV (Leone et al., 2006). LAUFEN encodes the system specification as facts using the predicates described in Table 2, and the intruder model described in Section 4. Then the DLV solver is used to enumerate the attack paths. We validate LAUFEN on the modeled Apollo system architecture. The implementation and the experimental results are available at (Dantas and Nigam, 2022b).

Given the high complexity of the Apollo model, naively computing the attack paths based on reachability does not scale, in particular for the outsider intruder. To address this problem, the computation is divided into two steps. The first step, *Intruder reachability*, computes all the model elements that are reachable by the intruder as specified by the write and read, and reachability rules. Since no paths are computed, the DLV engine computes the reachable elements in the range of milliseconds. We then use the reachable elements as input to the second step, *Path computation*, where we make use of the attack rules. Instead of enumerating all paths, we proceed using a goal-oriented search to enumerate only the attack paths on assets (a.k.a. asset-centric approach). This means that DLV does not require to compute all paths.

We run the experiments on a 1.90GHz Intel Core i7-8665U with 16GB of RAM running Ubuntu 18.04 LTS with kernel 5.4.0-113-generic and DLV 2.1.1. Table 3 shows the number of identified attack paths, and the execution time of LAUFEN. The execution time in enumerating the attack paths is rather low, i.e., 1.11 and 0.06 seconds for the outsider and insider intruder, respectively. The number of identified attack paths is high due the complexity of the system, e.g., the great number of public elements and the great number of information flows in the architecture. We do not rule out any attack path to guarantee a complete coverage of possible steps exploited by the intruder. Section 5.1 elaborates on countermeasures that may mitigate several of the identified attack paths.

We analyzed the generated attack paths w.r.t. potential attacks against safety-critical topics. Table 4 organizes selected attack paths into attacks carried out by an outsider attacker and insider attacker.

Firstly, our analysis was able to identify several attacks that have been reported in the literature, namely those attacks associated with a citation. The set of attack paths computed includes attacks carried out by outsider attackers exploiting `Bluetooth`, `LiDAR`, `Camera`, `GPS` and `Radar` that may target safety-critical topics, cause loss scenarios and harm to road-users, as well as by insider attackers exploiting SOA communication vulnerabilities to target topics.

Secondly, we also identified potential attacks that up to the best of our knowledge have not been reported

Table 4: Potential attacks derived from selected attack paths. *From* and *To* denote, respectively, the model element where the attack starts and ends. *Affected Topic* denotes the actual target of the attacker. The upper and lower part of the table describe, respectively, selected attacks carried out by the outsider and insider intruder, incl. attacks reported in the literature. NA denotes attacks that up to the best of our knowledge have not been reported in the literature. *#Attack Paths* denotes that number of computed attack paths *From* and *To*.

From	To	Affected Topic	Article	#Attack Paths
		Outsider Intruder		
Bluetooth	VDC	signal	(Chowdhury et al., 2020)	3
LiDAR	MCU	obstacles	(Hau et al., 2021)	24
Front Left Camera	MCU	obstacles	(Jha et al., 2020)	18
GPS	MCU	localization pose	(Shen et al., 2020)	18
Front Radar	MCU	obstacles	(Komissarov and Wool, 2021)	6
T-Box	MCU	traffic light	NA	18
		Insider Intruder		
gnss driver	velodyne detection	tf	(Hong et al., 2020)	1
gnss driver	msf localization	gnss best pose	(Hong et al., 2020)	1
compensator	velodyne detection	pointcloud2	(Hong et al., 2020)	1
control	chassis	signal	(Hong et al., 2020)	1
chassis	gnss driver	chassis	(Hong et al., 2020)	1
v2x proxy	traffic light	traffic light	NA	1
routing	planning	routing response	NA	1
relative map	planning	map	NA	1

in the literature (marked with NA). We analyzed in further detail some of the attacks by using the model and its connection to the Apollo code to find out how such attacks can lead to safety problems.

```
void TrafficLightsPerceptionComponent::OnReceiveImage(
    const std::shared_ptr<apollo::drivers::Image> msg,
    const std::string& camera_name) { ...
/** Set traffic light status based on camera data **/
 traffic_light_pipeline_->Perception(
 camera_perception_options_,frame_.get()); ...
/** Overwrites traffic light status if valid v2x data **/
 SyncV2XTrafficLights(frame_.get()); ...}
```

Figure 7: Overwriting traffic light status with V2X data.

V2X Traffic Light Overwrite Attack. LAUFEN has identified attack paths targeting `v2v proxy` from both outside and inside. Figure 8 illustrates the attack from the outside. The `v2v proxy` component publishes data on the traffic light status obtained from the road infrastructure. This data is subscribed by the `traffic light` component which also subscribes data from the cameras to identify and publish the traffic light status. Since there is a traceability from the Apollo model and the Apollo source code, it is straightforward to find the relevant classes for vulnerabilities. Indeed, we found out that the function TrafficLightsPerceptionComponent gives priority to the data received by `v2v proxy` over the data received by the cameras. Figure 7 shows a code snippet from the TrafficLightsPerceptionComponent function. As a result, an attacker may manipulate the traffic light status from either outside (i.e., spoofing attack) or inside (i.e., MITM attack). As illustrated by Figure 8, a spoofing attack from `T-Box` manipulating the traffic light status can cause serious harm to passengers and pedestrians as the vehicle can cross a red-light.

Route/Mapping Injection Attack. The seriousness of some of the identified attack paths may not be too obvious from a safety perspective. For example, LAUFEN has identified the following attack path: {`routing, planning`} targeting the `routing response` topic. An insider attacker may carry out a MITM attack between `routing` and `planning` to provide a malicious route for the ego vehicle. From a safety perspective, a loss scenario of type erroneous from `routing` to `planning` may be easy to control, and hence it would lead to a low criticality hazard (e.g., ASIL A or B). From a security perspective, however, there are several serious consequences, including hijacking of passengers. The `planning` component may also be affected by the road map published by `relative map`, as `planning` takes the map into account while computing the vehicle trajectory.

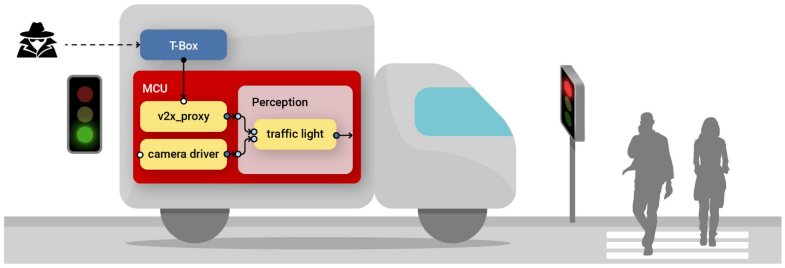

Figure 8: Illustration of a spoofing attack from `T-Box` to manipulate traffic light status received by `v2v proxy`.

5.1 Potential Countermeasures

We have performed an attack path analysis to deduce potential locations for instantiating security countermeasures. Our analysis focused on the computed attack paths using the outsider intruder. This choice was made because we noticed that many of such attack paths have the same prefix, which may be a hint for instantiating security countermeasures.

Table 5 presents the main results of our attack path analysis. Specifically, Table 5 shows the public element that can be reached the outsider intruder, the number of attack paths computed by LAUFEN from the public element, and the common prefix for all attack paths from the same public element.

Table 5: Attack paths analysis (outsider intruder).

Public element	#Attack Paths	Prefix
`Front Left Camera`	21	`Front Left Camera` → `GMSL` → `VIU 1`
`Front Right Camera`	21	`Front Right Camera` → `GMSL` → `VIU 3`
`GPS`	21	`GPS` → `Serial` → `VIU 3`
`Front Radar`	10	`Front Radar` → `CAN` → `MDC`
`Rear Radar`	10	`Rear Radar` → `CAN` → `MDC`
`LiDAR`	27	`LiDAR` → `SW4`
`Bluetooth`	21	`Bluetooth` → `USB` → `CDC`
`T-Box`	21	`T-Box` → `SW3`

The last architecture element described in the Prefix column may be a suitable location to instantiate a countermeasure and consequently address the attack paths. From the Prefix column, we can also notice that the gateway `VIU 3` is a common location in the attack paths from both `Front Right Camera` and `GPS`. Similarly, the connection between `CAN` and `MDC` is a common location in the attack paths from `Front Radar` and `Rear Radar`. This gives us a hint that security countermeasures could be placed in front of `VIU 3`, and between `CAN` and `MDC` to address such attack paths (specifically, 62 attack paths).

Firewalls are, e.g., recommended (Cheng et al., 2019) as means to protect vehicle architectures against such attacks. They may be deployed in front of the last architecture elements in the Prefix column to filter network traffic and prevent malicious intrusion. For the network interfaces (i.e., `T-Box` and `Bluetooth`), one could also implement a mutual authentication mechanism (e.g., mTLS) to ensure that only authenticated messages are accepted.

Safety architecture patterns, such as Heterogeneous Duplex pattern (Armoush, 2010), may also be deployed as a second-layer of defense. Consider, e.g., the V2X traffic light overwrite attack carried by an outsider attacker. This attack violates the integrity of `traffic light` topic through `T-Box`. A possible countermeasure is to include a checker in the `traffic light` component to consider inputs from both `v2x proxy` and cameras (i.e., heterogeneous inputs) – the `traffic light` component emits an alert to the driver or transition the system to a safe state if the inputs do not match.

The MITM attacks (e.g., the route injection attack) carried by an insider attacker exploit SOA communication vulnerabilities to violate the integrity of topics. Digital signatures are a well-known countermeasure for ensuring authenticity and integrity between servers (e.g., publishers) and clients (e.g., subscribers). To address MITM attacks, one can implement digital signatures in the Apollo system, where each publisher originator signs its message, and each subscribe of the message verifies the signature of the message. Fast-DDS provides a cryptographic plugin for message authentication codes computation and verification. The use of digital

certificates to address MITM attacks in Apollo was inspired by (Hong et al., 2020) that proposed a countermeasure using digital signatures for mitigating publisher-subscriber overprivilege issues in Apollo.

All 94 attack paths (insider intruder) are, in principle, addressed upon implementing digital signatures. The decision of using digital signatures causes, however, a performance penalty at the execution time of software components. The performance penalty can then be analyzed according to several points of view, including security, safety and financial. However, since all of the identified attack paths are safety critical, countermeasures shall be implemented to ensure vehicle safety.

6 RELATED WORK

Due to the lack of space, we moved some parts of the related work to (Dantas et al., 2022).

Attacks Against Vehicle SOA. The following work has inspired us to formalize the intruder model for vehicle SOA. A recent systematization of knowledge article (Shen et al., 2022) gives an overview of the state-of-the-art of the literature. In "Drift with the devil" (Shen et al., 2020) it is shown that an intruder may manipulate location information by spoofing GPS radio signals. LiDAR sensor signals may be spoofed to remove obstacles on the road (Hau et al., 2021). Camera signals may also be spoofed to manipulate video frames given that the camera traffic is transmitted in plain text (Jha et al., 2020). An attack may exploit vulnerabilities in a Bluetooth stack weakness to lock the brakes of the vehicle (Chowdhury et al., 2020). In the work by (Zelle et al., 2021), the authors investigate possible security issues in the service discovery mechanism of vehicle SOA, in particular SOA using the SOME/IP protocol, enabling the attacker to carry out MITM attacks between publishers and subscribers. In the work on AVGuardian (Hong et al., 2020), the authors investigated possible publisher/subscriber overprivilege instances in Apollo.

Our intruder model specifies the main attacker's capabilities needed to carry out the above attacks at the architecture level, including the capabilities of attackers to carry out (a) spoofing attacks from outside (e.g., from sensors), and (b) MITM attacks from inside (e.g., between components), thus violating the integrity of safety-critical topics. The attacks exploiting overprivileged instances can be seen as a specific case of the MITM attack.

Automate Threat Analysis. A survey on threat modeling (Xiong and Lagerström, 2019) has shown that most threat modeling work remains to be done manually. We briefly describe some of the security/threat analysis tools that provide computed-aided support in the automotive domain.

AVGuardian (Hong et al., 2020) is a static analysis tool to detect overprivilege instances in source code implementing service-oriented architectures for automotive systems. AVGuardian examines each module's source code and automatically detects publisher and subscriber overprivilege instances in the fields of topics defined by the module. AVGuardian requires the behavior specification of the system to detect overprivilege instances. LAUFEN has been implemented to identify threats and attack paths during the design of the system architecture without the behavior specification of the system. LAUFEN was able to automatically compute attack paths that may lead to the attacks detected by the AVGuardian tool.

ProVerif (Blanchet et al., 2022) and Tamarin Prover (Basin et al., 2022) are well-known automated reasoning tools to verify the security properties of systems (in particular, security protocols) with the Dolev-Yao intruder model (Dolev and Yao, 1983). These reasoning tools require the formal specification of the behavior of the system to verify its properties. A promising future work direction is to include the behavior specification in our Apollo model and use such reasoning tools to verify security properties of SOA protocols such as SOME/IP or DDS.

An attack propagation method that targets automotive safety-critical functions has been proposed by (Fockel et al., 2022). The commercial tool ThreatGet (thr, 2022) enables the identification of attack paths following ISO 21434. Microsoft SDL Threat Modeling tool (sdl, 2022) is another well-known commercial tool to compute threats. The threats are computed using STRIDE. The attack path associated to each compute threat is represented using data flow diagrams. To the best of our knowledge, these tools do not support intruder model capabilities for vehicle SOA. This article advances the state-of-the-art by proposing a machinery built upon realistic formalized intruder models for vehicle SOA.

7 CONCLUSION

This article proposed automated methods for threat analysis using a model-based engineering methodology. To this end, we have (a) modeled a faithful vehicle model of the autonomous driving functions of the Apollo framework, (b) formalized

an intruder model for vehicle SOA based on a literature review, and (c) developed LAUFEN, an SOA machinery for computing several activities of a threat analysis, including attack paths.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for valuable comments. We thank Anait Boyajyan for the assistance provided with the figures used in this article.

REFERENCES

(2018). SAE International Releases Updated Visual Chart for Its "Levels of Driving Automation" Standard for Self-Driving Vehicles.

(2019). Apollo Cyber RT. Available at https://cyber-rt.readthedocs.io/.

(2022). Microsoft SDL Threat Modeling Tool. Available at https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling.

(2022). ThreatGet - Threat Analysis and Risk Management. Available at https://www.threatget.com/.

Apollo (2021). An Open Autonomous Driving Platform. https://github.com/ApolloAuto/apollo.

Aravantinos, V., Voss, S., Teufl, S., Hölzl, F., and Schätz, B. (2015). AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems. In *ACES-MB'15*.

Armoush, A. (2010). *Design Patterns for Safety-Critical Embedded Systems*. PhD thesis, RWTH Aachen University.

Basin, D., Cremers, C., Dreier, J., Meier, S., Sasse, R., and Schmidte, B. (2022). Tamarin Prover https://tamarin-prover.github.io/.

Blanchet, B., Cheval, V., Allamigeon, X., Smyth, B., and Sylvestre, M. (2022). ProVerif https://bblanche.gitlabpages.inria.fr/proverif/.

Cheng, B. H. C., Doherty, B., Polanco, N., and Pasco, M. (2019). Security Patterns for Automotive Systems. In *MODELS'19*.

Chowdhury, A., Karmakar, G. C., Kamruzzaman, J., Jolfaei, A., and Das, R. (2020). Attacks on Self-Driving Cars and Their Countermeasures: A Survey. *IEEE Access*.

Dantas, Y. G., Barner, S., Ke, P., Nigam, V., and Schoepp, U. (2022). Technical Report: Automating Vehicle SOA Threat Analysis using a Model-Based Methodology. Technical report.

Dantas, Y. G. and Nigam, V. (2022a). Automating Safety and Security Co-Design through Semantically-Rich Architectural Patterns. *ACM Trans. Cyber Phys. Syst.*

Dantas, Y. G. and Nigam, V. (2022b). https://github.com/ygdantas/LAUFEN.

Dolev, D. and Yao, A. C. (1983). On the security of public key protocols. *IEEE Trans. Inf. Theory*, 29(2):198–207.

Fockel, M., Schubert, D., Trentinaglia, R., Schulz, H., and Kirmair, W. (2022). Semi-automatic integrated safety and security analysis for automotive systems. In *MODELSWARD'22*.

Förster, D., Loderhose, C., Bruckschlögl, T., and Wiemer, F. (2019). Safety goals in vehicle security analyses: a method to assess malicious attacks with safety impact. In *the 17th escar Europe - Embedded Security in Cars*.

fortiss GmbH (2022). AutoFOCUS 2.21.

Hau, Z., Co, K. T., Demetriou, S., and Lupu, E. C. (2021). Object removal attacks on lidar-based 3d object detectors. *CoRR*, abs/2102.03722.

Hong, D. K., Kloosterman, J., Jin, Y., Cao, Y., Chen, Q. A., Mahlke, S. A., and Mao, Z. M. (2020). AVGuardian: Detecting and Mitigating Publish-Subscribe Overprivilege for Autonomous Vehicle Systems. In *EuroS&P'20*.

ISO26262 (2018). ISO 26262, road vehicles — functional safety — part 6: Product development: software level.

ISO/SAE 21434 (2020). Road vehicles - cybersecurity engineering.

Jha, S., Cui, S., Banerjee, S. S., Cyriac, J., Tsai, T., Kalbarczyk, Z., and Iyer, R. K. (2020). ML-Driven Malware that Targets AV Safety. In *DSN 2020*.

Komissarov, R. and Wool, A. (2021). Spoofing attacks against vehicular FMCW radar. In *ASHES@CCS'21*.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7.

Leveson, N. G. and Thomas, J. P. (2018). *STPA Handbook*.

Macher, G., Sporer, H., Berlach, R., Armengaud, E., and Kreiner, C. (2015). SAHARA: A Security-aware Hazard and Risk Analysis Method. In *DATE'15*.

Sabaliauskaite, G., Liew, L. S., and Cui, J. C.

Shen, J., Wang, N., Wan, Z., Luo, Y., Sato, T., Hu, Z., Zhang, X., Guo, S., Zhong, Z., Li, K., Zhao, Z., Qiao, C., and Chen, Q. A. (2022). Sok: On the semantic AI security in autonomous driving. *CoRR*, abs/2203.05314.

Shen, J., Won, J. Y., Chen, Z., and Chen, Q. A. (2020). Drift with Devil: Security of Multi-Sensor Fusion based Localization in High-Level Autonomous Driving under GPS Spoofing. In *USENIX'20*.

Shostack, A. (2014). *Threat Modeling: Designing for Security*. Wiley.

SOTIF, I. . (2021). Safety of the Intended Functionality.

UN (2021). UN Regulation No. 155 - Cyber security and cyber security management system.

WIRED (2015). Hackers remotely kill a jeep on the highway-with me in it. Available at https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

Xiong, W. and Lagerström, R. (2019). Threat modeling - A systematic literature review. *Comput. Secur.*, 84:53–69.

Zelle, D., Lauser, T., Kern, D., and Krauß, C. (2021). Analyzing and Securing SOME/IP Automotive Services with Formal and Practical Methods. In *ARES'21*.