REMIND: A Framework for the Resilient Design of Automotive Systems

Citation for the original published paper (version of record):
Rosenstatter, T., Strandberg, K., Jolak, R. et al (2020)
REMIND: A Framework for the Resilient Design of Automotive Systems
2020 IEEE Secure Development (SecDev): 81-95
http://dx.doi.org/10.1109/SecDev45635.2020.00028

REMIND: A Framework for the Resilient Design of Automotive Systems

Thomas Rosenstatter*, Kim Strandberg*†, Rodi Jolak‡ and Riccardo Scandariato‡, Tomas Olovsson*
*Chalmers University of Technology, Sweden, {firstname.lastname}@chalmers.se
†Volvo Car Corporation, Sweden
‡Chalmers | Gothenburg University, Sweden, {firstname.lastname}@cse.gu.se

***Abstract*—In the past years, great effort has been spent on enhancing the security and safety of vehicular systems. Current advances in information and communication technology have increased the complexity of these systems and lead to extended functionalities towards self-driving and more connectivity. Unfortunately, these advances open the door for diverse and newly emerging attacks that hamper the security and, thus, the safety of vehicular systems. In this paper, we contribute to supporting the design of resilient automotive systems. We review and analyze scientific literature on resilience techniques, fault tolerance, and dependability. As a result, we present the REMIND resilience framework providing techniques for attack detection, mitigation, recovery, and resilience endurance. Moreover, we provide guidelines on how the REMIND framework can be used against common security threats and attacks and further discuss the trade-offs when applying these guidelines.**

***Index Terms*—cyber-physical systems, resilience techniques, security, vehicular systems, automotive systems.**

I. Introduction

In the past years great effort has been spent in publishing guidelines and standards for security frameworks specific to their domains and in identifying security principles. Examples range from the NIST guideline for cybersecurity in smart grids [1], the cybersecurity guideline for ships [2], cybersecurity guidelines for the automotive domain [3]–[5] and the upcoming ISO/SAE standard for cybersecurity engineering for road vehicles, namely ISO 21434 [6].

Resilience is the next step towards reliable, dependable and secure vehicular systems. Vehicles need to be able to mitigate faults, errors, attacks and intrusions that would ultimately result in failures in order to withstand safety and security threats from their environment. We define automotive resilience as the *"property of a system with the ability to maintain its intended operation in a dependable and secure way, possibly with degraded functionality, in the presence of faults and attacks."* This definition is inspired by Laprie's definition [7] and the definition of network resilience by Sterbenz et al. [8]; however, the chosen definition highlights that faults or changes, e.g., functional and environmental (see [7]), can also be originated by an attacker whose aim is to disrupt the system.

Resilience can be obtained in many different ways and on different levels, i. e., hardware, software or (sub)-system level. Today's internal architecture of vehicles is quite complex and can be distributed over more than hundred so-called Electronic Control Units (ECUs). However, we are currently in a transition towards a more centralized architecture where functions will be concentrated on much fewer and more powerful ECUs [9]. These central ECUs are connected to sensors, actuators, external communication media and to some extent to smaller legacy subsystems. Such a centralized architecture enables vehicle OEMs not only to perform more resource intensive operations needed for autonomous driving, but also allows to introduce new designs and technologies needed to secure and protect these highly connected and autonomous vehicles. Virtualization is seen as one key technology enabling the isolation of vehicle functions from each other along with the possibility to dynamically assign hardware resources. Introducing resilience to such a centralized automotive system requires the deployment of techniques and principles in all layers and components of the system, ranging from the vehicle itself, the connected IT infrastructure, road infrastructure and the communication to other vehicles.

Motivation. The increasing complexity towards autonomous driving combined with the interconnectedness of vehicles, e. g., vehicle-to-vehicle and vehicle-to-infrastructure communication, and the continuous development of functions require vehicles to react and adapt to changes and attacks independently. The automotive domain is distinct from other domains as it is a safety and real-time critical system operated by millions of individuals each day. Furthermore, security and safety techniques need to be aligned and extended with resilience techniques in order to strengthen vehicles' capabilities to withstand impending threats.

Contributions. This paper provides a framework to design resilient automotive systems. First, we systematically identify relevant automotive resilience techniques proposed in the literature with the goal to provide a full picture of available tools and techniques. We also organize these techniques into a taxonomy, which comprises the categories of Detection, Mitigation, Recovery, and Endurance (REMIND). These categories represent high-level strategies that can help designers understand the *purpose* of each technique. Further, it can be beneficial to combine techniques from different strategies to achieve multiple layers of security. The selection of the right technique for the task at hand is further supported by associating the resilience techniques to the classes of *automotive assets* they are appropriate for. Additionally, we elaborate on the *trade-offs* (i.e., pros and cons) that are associated with each of the techniques, e.g., with respect to performance and other qualities. In summary, we provide a multi-dimensional decision support framework (built in a bottom-up fashion from the analysis of the literature) that can lead designers to the informed and optimal selection of a suitable set of resilience techniques to be implemented in an automotive system.

TABLE I
PUBLICATIONS THAT PROVIDE AN OVERVIEW OR COLLECTION OF RELEVANT TECHNIQUES.

Discipline	Existing Work	Domain
Resilience	Chang2015 [10]	Fog Computing
	Hukerikar2017 [11]	High Performance Computing
	NIST 800-160v2 [12]	Systems Engineering
	Ratasich2019 [13]	Cyber-Physical Systems
	Sterbenz2010 [8], [14]	Networks
Security	Segovia2019 [15]	SCADA systems
Dependability	Bakhshi2019 [16]	Fog Computing
Fault Tolerance	Egwutuoha2013 [17]	High Performance Computing
	Kumari2018 [18]	Cloud Computing
	Mukwevho2018 [19]	Cloud Computing
	Slåtten2013 [20]	Software Engineering
	Wanner2012 [21]	Vehicle Controller

II. METHODOLOGY

By means of a systematic literature survey, we identify research papers that discuss techniques that are suitable to provide automotive resilience. We consider existing work related to resilience, fault tolerance and dependability. We also analyze the papers describing each technique to understand (i) the assets that can benefit from the technique, (ii) the risks that are mentioned as being mitigated by the techniques, and (iii) any pros/cons associated with the use of such technique.

We identified relevant research papers by searching the Scopus database[1]. A search string was intended to find relevant publications that carried out a review of suitable techniques. Therefore, we formulated the search string to *include* survey or literature review, and relevant topics, such as resilience, survivability, attack recovery, error handling or fault tolerance, as well as the keywords software, system or network. We *excluded* the keywords FPGA, memory, wireless, SDN and hardware to limit the search result to publications focusing on system architecture, software design or physical networks. Furthermore, we considered only publications written in English and published after 2010 in the areas of computer science and engineering. We manually screened the 200 most relevant publications returned by Scopus and found eight additional research publications, which were added to our result set. Ultimately, we retained and analyzed 12 publications which are shown in Table I.

III. ATTACK MODEL AND ASSETS

The four *strategies* in the REMIND framework are, as shown in Figure 1, further refined in *patterns* and *techniques*. A collection of these techniques specific for automotive systems is described in Section IV and has been identified based on existing research in other domains and areas (see Table I). We additionally describe the trade-offs of these techniques in Appendix A and point to relevant publications in Appendix B. In the remainder of this section we describe the assets, security threats and attacks of automotive systems.

We consider four asset types, namely *Hardware*, *Software*, *Network/Communication* and *Data Storage*. The attacker aims to compromise these assets via various attack vectors, whereas the defender, i.e. the vehicle, aims to cope with these attacks via resilience techniques. We consider skilled attackers as well as novice hackers (e. g., script kiddies) and further give examples from an asset, threat and attacker perspective.

Hardware. Can be broken down to *ECUs*, *Sensors* and *Actuators*. An *ECU* can vary in complexity depending on its objective, from a specific limited task to a multitude of tasks. The former can relate to the processing of a sensor signal and the latter an infotainment-system with lots of applications. *Sensors* can give information about speed, temperature and obstacle distance and identification where the *Actuators* turn input from these sensors (via an ECU) into actions, such as braking, steering and engine control.

Attack example. Tampering with existing hardware or installing malicious hardware into the vehicle can act as mediators to gain complete vehicle control. Input signals from sensors may be manipulated to cause an unwanted behavior.

Software. Can be *in transit*, *at rest* or *running*. *In transit* can relate to software provisioning systems, such as over-the-air or workshop updates and the latter two to software installed or running in ECUs.

Attack example. Software vulnerabilities might be exploited, e. g., via a privilege escalation attack which enables ECUs to be re-programmed with additional functionalities, such as adding remote access to the system.

Network/Communication. Can be broken down to *internal* and *external communication*. Examples for internal communication are CAN, FlexRay, LIN, MOST and Automotive Ethernet and for external communication Wi-Fi, Bluetooth, and V2X as well as external interfaces such as OBD-II, debug ports (e. g., JTAG) and CD player.

Attack example. The attacker can try to inject malicious data, through a device connected to an in-vehicle bus affecting the internal communication. Furthermore, modification of V2X data from other vehicles as well as malicious roadside units (e. g., vehicle positioning or traffic condition data) could affect system functions.

Data Storage. Can potentially be sensitive data, such as cryptographic keys, forensics logs, system information (e. g., from software libraries, OS and applications) and reports about the vehicle and the driver.

Attack example. The attacker can exploit secret keys used for sensitive diagnostics to disable firewalls. Logs and report data might be manipulated or removed to hide forensic evidence of the crime. Furthermore, information about the system can reveal vulnerabilities which might be exploited.

Attackers typically exploit the above-mentioned assets in any order to achieve their goal, e. g., uploading malicious software to the vehicle by first compromising the cryptographic keys to get access to the memory and consequently upload a modified firmware containing malicious code. This can give elevated privileges and extended functionality which could cause inconsistencies or disruption of the system.

More examples of assets and related security threats and attacks can be found in Table II.

[1] https://www.scopus.com/

TABLE II
AUTOMOTIVE ASSETS AND RELATED SECURITY THREATS AND ATTACKS

Asset	Asset Examples	Security Threat	Attack Examples
Hardware	ECU (hardware) Sensors Actuators	Disruption or direct intervention. Availability and Integrity.	*Fault Injection:* fuzzing, DoS, microprobing, malicious hardware as well as environmental injections (e. g., voltage and temperature) can disrupt or disable components or system resources. *Information Leakage:* side channel parameters, such as timing information or power consumption (e. g., differential power analysis) to extract secret keys.
Software	ECU (software) Libraries OS Virtualization	Manipulation of software, measurements or control signals. Availability and Integrity.	*Malware/Manipulated software:* indirectly affecting storage through alteration, deletion or blocking data, or indirect affecting the communication by read, manipulate or replay of messages, hence causing disruption and deviations from normal system operation.
Network/ Communication	CAN LIN MOST FlexRay Automotive Ethernet Mobile Network Wi-Fi Bluetooth OBD-II CD player	Communication failure or protocol vulnerabilities. Confidentiality, Integrity, Availability and Privacy.	*Fabrication/Jamming attack:* introducing fake traffic, e. g., sending high priority messages, to block legitimate low priority messages. *Masquerading/Spoofing attack:* masquerading as a legitimate node, e. g., by suspending the authentic ECU and send fabricated messages which seems to origin from the same. *Collision:* spoofing a message to induce a bit error/collision and then potentially spoof additional messages which get accepted. *Eavesdropping/hijacking:* intercept to read, block, manipulate or replay messages. *Suspension/DoS attack:* disable an ECU, such as inducing programming mode causing an ECU to not transmit or relay messages, potentially causing other ECUs to malfunction.
Data Storage	User Data Logs/Reports/Events Checkpoints Backups Forensics data Cryptographic material	Malicious handling of data storage. Confidentiality, Integrity, Availability and Privacy.	*Unauthorized read:* acquire sensitive data, such as privacy related user data e. g., previous locations or driving behavior. *Manipulation*: malicious alteration of data, e. g., replacing the software validation key enables potential alteration of memory data. *Removal:* data deletion of sensitive information, such as forensics data. *Reverse engineering:* extraction and analysis of firmware to deduce design features, vulnerabilities or secret keys.

IV. REMIND AUTOMOTIVE RESILIENCE FRAMEWORK

We have developed the REMIND framework shown in Figure 1 to provide system designers and developers with a categorization of suitable resilience mechanisms including the identification of the assets they protect. The structure of the layers is chosen similarly to the work in Hukerikar et al. [11], where the bottom layer is divided into *strategies* and the mid layer is split into *patterns* that provide more details about the way the strategies can be realized. We refer to relevant solutions for automotive systems in the top layer and further link to the survey papers and reviews that identify specific *techniques* for their domain in the description listed below.

The four REMIND strategies for providing resilience for vehicular systems are:

- **Detection.** Faults, attacks and other anomalies need to be detected by the system in order to take reactive measures to avoid a failure.
- **Mitigation.** Once an anomaly is detected and located, mitigation techniques need to be triggered to keep the system operational. These techniques may result in a non-optimal system state.
- **Recovery.** Transitioning back to the desired, i. e., optimum state, is the aim of recovery.
- **Endurance.** The focus is set on lasting resilience in contrast to recovery & mitigation strategies which aim at taking immediate measures.

The remaining part of this section details the strategies and describes the patterns and corresponding techniques.

A. Detection

The monitoring and detection capabilities of a system can be limited due to various factors, such as computational resources, energy consumption, and the complexity of functions and network architecture. The move to a more centralized architecture, however, paves the way for more extensive monitoring.

1) **SPECIFICATION-BASED DETECTION***:* Malicious or abnormal behavior is detected using a specification that describes the behavior of signals or communication patterns. Domain knowledge is needed to create the specifications.
– *Signature-based Detection [13].* Signatures are constructed to describe known attack behavior. By design, these techniques suffer from detecting new attacks and zero-day vulnerabilities. However, they typically achieve a low false positive rate [24].
– *Runtime Verification [13], [20].* A monitor observes the system at runtime to verify the correctness of the execution. Formal specification languages, e. g., Signal Temporal Logic [58], have been developed to describe the normal system behavior which is matched against a trace during execution.
– *Falsification-based Analysis [13].* It extends STL by including a quantitative semantics allowing the return of real values rather than Boolean values.
– *Verification of Safety-Properties [13].* The formal verification of safety properties has become increasingly complex due to the added functionality in modern vehicles. Exhaustive verification techniques, as listed and argued by Ratasich et al. [13], are currently limited to small scale models.
– *Specification-based Anomaly Detection.* Normal behavior, according to a set of rules, is defined using this technique. An alert is sent when a violation of these rules is detected [24].

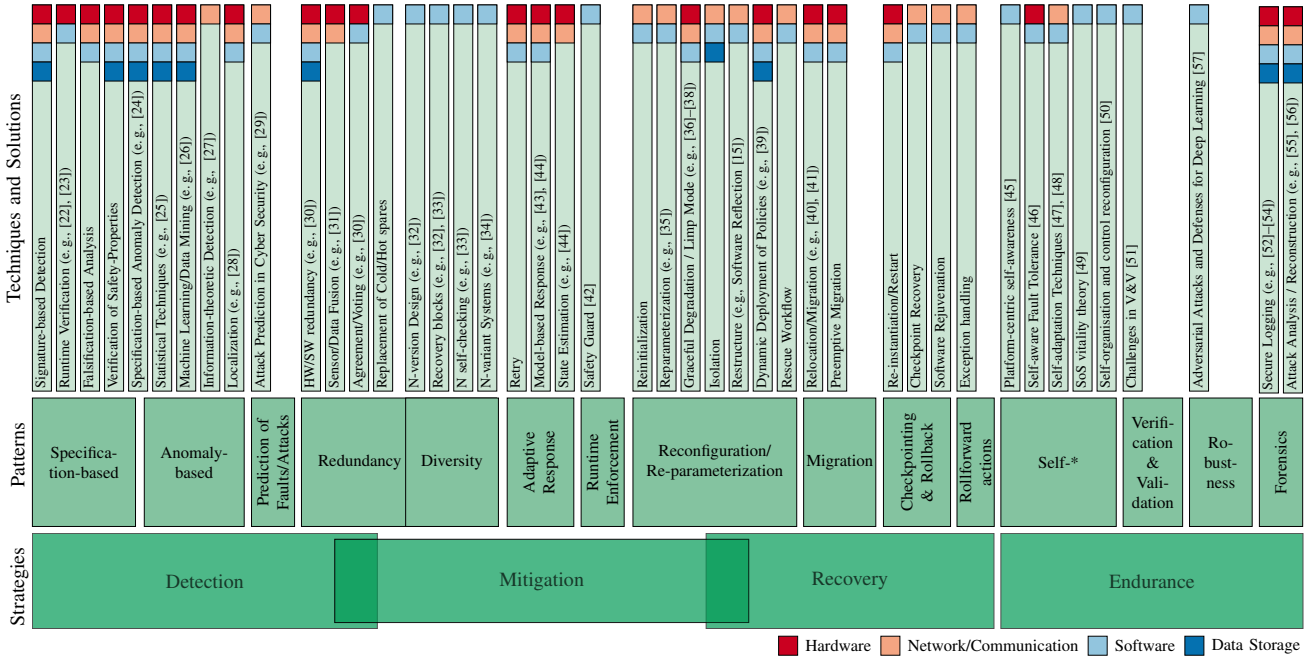

Fig. 1. REMIND resilience techniques and solutions including a mapping to the assets for each technique. The overlap of the *Mitigation* strategy highlights that some patterns also contribute to *Detection* respectively *Recovery*.

2) **ANOMALY-BASED DETECTION***:* Anomaly- or behavior-based detection techniques are based on comparing behavior with a model of normal behavior. Alerts are raised when a deviation is detected [59].

– *Statistical Techniques [13].* A statistical model describing the system or a specific process is designed in order to detect anomalies. Events are considered anomalies when the probability of their occurrence is below a certain threshold according to the model.

– *Machine Learning/Data Mining [13].* These techniques typically do not require domain knowledge. A model, such as Bayesian networks, neural networks and support vector machines, learns through training data how to classify observations in normal and abnormal classes.

– *Information-theoretic Detection [13].* The entropy of information can be used to detect anomalies, as a change of the entropy above a certain threshold may be caused by an attack, e. g., masquerading attack [13], [27].

– *Localization.* Finding the source of the attack may be required to take appropriate actions. Network-based Intrusion Detection Systems (IDSs) can be used to limit the location to a specific subnet, however, solutions identifying the particular ECU are needed (e. g., [28]).

3) **PREDICTION OF FAULTS AND ATTACKS***:* First, the system needs to identify the presence of an attacker. The next actions are *attack projection* and *attack intention recognition* which aim at identifying the next steps and the ultimate goal of the attacker. *Attack or intrusion prediction* can be used to foresee when and where an attack will take place [29].

Adversaries mounting simpler attacks on a single vehicle, such as DoS attacks on the CAN bus, may be difficult to predict as the attack consists of fewer steps. However, large-scale attacks requiring the attacker to go through several stages may be predicted by this technique.

4) **REDUNDANCY***:* Redundancy is twofold, as it can support both detection and mitigation. It is important to highlight that purely redundant systems suffer from the same design faults and vulnerabilities. Thus, diversity is combined with redundancy to overcome this issue.

– *HW/SW Redundancy [11]–[13], [15], [17]–[20].* Redundancy combined with a voter allows to mask system failures. The voter compares the results of a number of independently executed software and/or hardware modules and selects, for instance, the majority [30]. Repeating the computation n times on the same hardware can be used to detect random faults.

– *Sensor/Data Fusion [13].* Data from different origins may be fused to compensate inaccuracies or temporary sensor failures. Sensor fusion, e. g., extended Kalman filter [60] and particle filter [31], can be used to describe the non-linear relationship between sensors. For example, the motion of a vehicle can be described with measurements from the wheel speed sensor, GPS location and data received from other vehicles.

– *Agreement/Voting [11], [13], [17], [20].* Redundant components are required for this technique. Voting can be realized in two ways, i. e., exact voting and inexact voting, where the latter allows a variation of the result within a certain range [30].

– *N-version Design [11]–[13], [17], [19].* N versions of a software with the same requirements are developed by N independent teams resulting in a diverse set of functionally equivalent software components that fulfill the same specification. These versions are executed concurrently and a voter decides based on the majority or calculates, for instance, the

median or average of the results [32].
– *Recovery Blocks [11], [19], [20].* Similar to n-version design, n versions of a software component exist; however, only one version is executed at a time. After the active version is executed, a common acceptance test decides whether the result is accepted. In case the result is rejected, the subsequent version is executed and evaluated [32], [33].
– *N self-checking [17].* This technique is a combination of n-version design and recovery blocks. It requires at least two diverse versions with their own acceptance test. When the active component fails its acceptance test, the subsequent component takes over [33].
– *N-variant Systems.* Multi-variant execution automatically diversifies software and monitors the output of at least two variants to detect and mitigate attacks [34].
– *Replacement of Cold/Hot Spares [13].* Concurrent and sequential execution of redundant software components is costly in terms of energy consumption and computational resources. Therefore, the introduction of cold or hot spares, such as in N self-checking, have been found to be a viable alternative [13].

B. Mitigation

After detecting an attack or anomaly, the system needs to react to reduce the impact of the attack. Some mitigation techniques may require the transition to a non-optimal state.

1) **Adaptive Response***:* We focus on techniques that adapt the response of a function or sub-system in order to maintain its intended functionality.
– *Retry [18], [19].* Performing the same computation with new measurements if the first computation resulted in an undesired system state or in an error. Retry can mitigate a replay attack.
– *Model-based Response and State Estimation [15], [21].* System models, e. g., Kalman filter for state estimation [60], [61], or parameter estimation techniques, like regression analysis, are not only a temporary solution to mitigate attacks, such as replay and masquerading attacks, they can also be used to alert the system and log important information for forensics [44].

2) **Runtime Enforcement***:* Runtime enforcement is an extension of runtime verification where the system also reacts to violations [22].

3) **Reconfiguration and Reparameterization***:* The system protects itself by adapting parameters when an attack is detected. We distinguish between reconfiguration and migration in the way that migration focuses on relocating functionality whereas reconfiguration changes system or application parameters.
– *Reinitialization [11].* Temporary faults and attacks can be addressed with this technique. However, permanent faults or reoccurring attacks cannot be mitigated by restoring the system or a function to its initial state. Reinitialization can be seen as checkpoint recovery with the checkpoint being the initial state of the system or function.
– *Reparameterization [13].* Is similar to reinitialization, however, the system configuration is dynamically adjusted to the situation. As Ratasich et al. [13] point out, reparameterization typically results in a non-optimal state.
– *Graceful Degradation / Limp Mode [13], [15].* Given the extended automated driving functions of future vehicles, it is of utmost importance to implement more sophisticated solutions that ensure the passengers safety when key components in the vehicle fail or are subject to attacks. These techniques are similar to reparameterization, but focus on safety and should be seen as a last resort. Modern vehicles already have a so-called limp mode implemented, which is triggered when the vehicle detects major technical problems [62].
– *Isolation [11], [13].* Restricting access or completely isolating system components in the presence of an error or intrusion can limit the impact on the entire system and its performance.
– *Restructure [11].* Restructuring components within a subsystem aims at providing resilience through reconfiguration of affected components. Segovia et al. [15] explore software reflection as means to mitigate attacks.
– *Dynamic Deployment of Policies [15].* Security or other policies can be applied dynamically based on the type of attack, e. g., DoS or masquerading, that is detected.
– *Rescue Workflow [18], [19].* A workflow can be used to describe tasks with their dependencies to each other. The idea behind rescue workflows is to dynamically adjust the structure of the workflow when an error or intrusion affects a specific task. Existing cloud solutions may need to be adapted for automotive systems.

C. Recovery

Recovery techniques intend to bring the system back to an optimal state.

1) **Migration***:* These techniques are mainly originating from high performance computing and cloud systems. As future automotive systems move towards a centralized architecture, virtualization and service-oriented architectures are becoming more relevant.
– *Relocation/Migration [13], [19].* Virtualization such as hypervisor and container-based solutions allow a fast migration and relocalization to other nodes in the vehicular network.
– *Preemptive Migration [18], [19].* Continuous monitoring and analysis of the system can be used to relocate software functions or services before a fault occurs.

2) **Checkpointing & Rollback***:* A checkpoint or snapshot describes the system state at a specific point in time. By design, recovery does not prevent the same attacks from happening again.
– *Re-instantiation/Restart [11], [13], [17], [19].* When an intrusion is detected, the affected component can be re-instantiated or restarted to recover to a known, error and attack free, state. This technique can be combined with reparameterization to avoid the same anomaly to happen again [13].
– *Checkpoint Recovery [11], [17]–[20].* Snapshots can be created in two ways: checkpoint-based and log-based. Egwutuoha et al. [17] highlight the complexity of taking checkpoints in a distributed system, as these checkpoints need to be consistent.
– *Software Rejuvenation [11], [19].* This technique carries out periodic restarts or reinitializations of the system to maintain a known, error-free state.

3) **ROLLFORWARD ACTIONS***:* These techniques aim at bringing the system to a stable state immediately before the error or attack was detected. As in rollback, the recovery is based on using checkpoint-based or log-based recovery [11].
– *Exception Handling [11].* From a model-driven engineering view, *Rollforward* can be performed using exception handling. Slåtten et al. [20] highlight that this solution can be only applied to anticipated events.

D. Endurance

Resilience needs to be ensured over the entire lifetime of a vehicle. The preceding techniques center around providing immediate response when anomalies are detected.

1) **SELF-***:* Self-* or self-X techniques cover solutions and research directions focusing on how to introduce autonomy into the system. This pattern is especially important for future vehicles as the environment is and will change frequently, new vulnerabilities will be found, new attempts to attack vehicles and their infrastructure will be developed, and new technologies will appear. Also, considering the lifetime of cars, which is around 10–15 years, it is evident that automotive systems need to adapt to a certain extent autonomously.

2) **VERIFICATION AND VALIDATION***:* Due to the increasing functionality and interconnectedness of modern vehicles it is required to update software components via over-the-air updates in order to fix vulnerabilities and bugs or upgrade vehicle functions. This is especially challenging as each vehicle model can be further configured, resulting in a manifold of possible vehicle configurations.

3) **ROBUSTNESS***:* Artificial intelligence, especially machine learning, is a key technology for autonomous driving and decision making, as the system needs to be able to handle previously unseen situations [13].

4) **FORENSICS***:* Providing evidence of intrusions even after a crash is important for taking appropriate countermeasures.
– *Secure Logging.* Hoppe et al. [52] express the need for forensic solutions in vehicles. Non-safety-critical events, such as updates, component failures and other malfunctions, need to be logged and stored securely for a prospective analysis. The authors also discuss in great detail which information and how this information can be stored in vehicles.
– *Attack Analysis.* Nilsson and Larson [55] specify requirements for forensic analyses of the in-vehicle network. It is also important to analyze attacks disclosed by researchers, such as Checkoway et al. [63] and Miller and Valasek [64], as well as attacks logged by the vehicle manufacturers in order to take appropriate actions.

V. RELATED WORK

Making vehicles safe and secure has traditionally been the main focus in research. For instance, methods to combine safety and security [65] and how to assess an automotive system and/or derive security requirements and mechanisms have been proposed [66]–[68]. Le et al. [69] provide a survey on security and privacy in automotive systems and further provide an overview of suitable security mechanisms.

One of the first structured collections of principles for cyber resilience is the Cyber Resiliency Engineering Framework [70] by MITRE in 2011 which got further incorporated in NIST SP 800-160v2 [12]. Other work describing principles for resilience have been either concentrating on other domains, i. e., high performance computing, cyber-physical systems, or networks, or they focused particularly on dependability or fault tolerance. Table I provides an overview of relevant publications, which provide a comprehensive overview or collection of techniques, and categorizes them according to their discipline and the area they are focusing on.

The reviewed publications classify the identified techniques in different ways. Hukerikar et al. [11] divide them into strategies, i. e., fault treatment, recovery, and compensation, whereas Ratasich et al. [13] organize them according to their ability, i. e., detection and diagnosis, recovery or mitigation, and long-term dependability and security. Work focusing on fault tolerance either split the identified techniques in reactive and proactive measures [18], [19] or classify them according to their ability, e. g., error handling and recovery [17], [20].

With the developed REMIND framework, we contribute to supporting the resilience of automotive systems by: (i) identifying techniques for attack detection, mitigation, recovery, and resilience endurance; (ii) organizing the techniques into a taxonomy to guide designers when selecting resilience techniques; (iii) providing guidelines on how the REMIND framework can be used against common security threats and attacks; and (iv) discussing the trade-offs when applying the techniques that are highlighted in this framework.

In addition to the identified techniques in Figure 1, we point to implementations relevant for or specific to the automotive domain in Appendix B.

VI. CONCLUSION

The reviewed work shows the current research efforts towards making systems resilient to attacks and faults in related domains. We present a novel structure for categorizing resilience techniques in the form of the REMIND framework with the aim to lead designers in making informed decisions when choosing resilience techniques. We build upon the existing work and set the focus on the limitations of automotive systems and their challenges. The REMIND techniques have been chosen considering automotive assets and related attacks which are described in Section III and further linked to the guidelines and trade-off analysis in Appendix A.

Future work includes the validation of the REMIND framework in regard to studying its applicability in industry in more depth. Furthermore, specific solutions for the identified techniques that consider the unique properties of automotive vehicles can be explored. Especially, the role of software-defined networking and its contribution to resilience can be investigated.

Acknowledgment. This research was supported by the CyReV project (2018-05013) funded by VINNOVA, the Swedish Governmental Agency for Innovation Systems.

References

[1] NISTIR 7628 Rev 1 – Guidelines for smart grid cybersecurity. Technical report, National Institute of Standards and Technology, September 2014.

[2] The Guidelines on Cyber Security Onboard Ships. Technical report, BIMCO, CLIA, ICS, INTERCARGO, INTERMANAGER, INTERTANKO, IUMI, OCIMF and WORLD SHIPPING COUNCIL, December 2017.

[3] Cyber Security and Resilience of smart cars. Technical report, The European Union Agency for Network and Information Security (ENISA), 2016.

[4] SAE J3061: SURFACE VEHICLE RECOMMENDED PRACTICE - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems. Standard, SAE International, 2016.

[5] UNECE. TFCS-09-14 Draft Recommendation on Cyber Security of the Task Force on CyberSecurity and Over-the-air issues of UNECE WP.29 IWG ITS/AD, 2017.

[6] ISO/SAE 21434 Road Vehicles – Cybersecurity Engineering. Standard, International Organization for Standardization (ISO), 2020.

[7] Jean-Claude Laprie. From dependability to resilience. In *38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, pages G8–G9, 2008.

[8] James PG Sterbenz, David Hutchison, Egemen K. Çetinkaya, Abdul Jabbar, Justin P. Rohrer, Marcus Schöller, and Paul Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks*, 54(8):1245 – 1265, 2010. Resilient and Survivable networks.

[9] Ödgärd Andersson. The Car - A Computer on Wheels. URL: https://www.icse2018.org/getImage/orig/The+Car+%E2%80%93+computer+on+wheels.pdf, May 2018. visited on 2020-05-21.

[10] Victor Chang, Muthu Ramachandran, Yulin Yao, Yen-Hung Kuo, and Chung-Sheng Li. A resiliency framework for an enterprise cloud. *International Journal of Information Management*, 36(1):155 – 166, 2016.

[11] Saurabh Hukerikar and Christian Engelmann. Resilience design patterns: A structured approach to resilience at extreme scale. *arXiv preprint arXiv:1708.07422*, 2017.

[12] Ron Ross, Victoria Pillitteri, Richard Graubart, Deborah Bodeau, and Rosalie McQuaid. Developing cyber resilient systems:: a systems security engineering approach. Technical Report NIST SP 800-160v2, National Institute of Standards and Technology, Gaithersburg, MD, November 2019.

[13] Denise Ratasich, Faiq Khalid, Florian Geissler, Radu Grosu, Muhammad Shafique, and Ezio Bartocci. A Roadmap Toward the Resilient Internet of Things for Cyber-Physical Systems. *IEEE Access*, 7:13260–13283, 2019.

[14] James PG Sterbenz, David Hutchison, Egemen K. Çetinkaya, Abdul Jabbar, Justin P Rohrer, Marcus Schöller, and Paul Smith. Redundancy, diversity, and connectivity to achieve multilevel network resilience, survivability, and disruption tolerance invited paper. *Telecommunication Systems*, 56(1):17–31, 2014.

[15] Mariana Segovia, Ana Rosa Cavalli, Nora Cuppens, and Joaquin Garcia-Alfaro. A study on mitigation techniques for scada-driven cyber-physical systems (position paper). In Nur Zincir-Heywood, Guillaume Bonfante, Mourad Debbabi, and Joaquin Garcia-Alfaro, editors, *Foundations and Practice of Security*, pages 257–264, Cham, 2019. Springer International Publishing.

[16] Zeinab Bakhshi, Guillermo Rodriguez-Navas, and Hans Hansson. Dependable Fog Computing: A Systematic Literature Review. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 395–403, 2019.

[17] Ifeanyi P Egwutuoha, David Levy, Bran Selic, and Shiping Chen. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *The Journal of Supercomputing*, 65(3):1302–1326, 2013.

[18] Priti Kumari and Parmeet Kaur. A survey of fault tolerance in cloud computing. *Journal of King Saud University - Computer and Information Sciences*, 2018.

[19] Mukosi A. Mukwevho and Turgay Celik. Toward a smart cloud: A review of fault-tolerance methods in cloud systems. *IEEE Transactions on Services Computing*, pages 1–1, 2018.

[20] Vidar Slåtten, Peter Herrmann, and Frank Alexander Kraemer. Chapter 4 - model-driven engineering of reliable fault-tolerant systems—a state-of-the-art survey. In Atif Memon, editor, *Advances in Computers*, volume 91 of *Advances in Computers*, pages 119 – 205. Elsevier, 2013.

[21] Daniel Wanner, Annika Trigell, Lars Drugge, and Jenny Jerrelind. Survey on fault-tolerant vehicle design. *World Electric Vehicle Journal*, 5(2):598–609, Jun 2012.

[22] Ezio Bartocci and Yliès Falcone. *Lectures on Runtime Verification: Introductory and Advanced Topics*, volume 10457. Springer, Cham, 2018.

[23] Donal Heffernan, Ciaran Macnamee, and Padraig Fogarty. Runtime verification monitoring for automotive embedded systems using the ISO 26262 functional safety standard as a guide for the definition of the monitored properties. *IET Software*, 8(5):193–203, 2014.

[24] Michael Müter, André Groll, and Felix C. Freiling. A structured approach to anomaly detection for in-vehicle networks. In *2010 Sixth International Conference on Information Assurance and Security*, pages 92–98, 2010.

[25] Nasser Nowdehi, Wissam Aoudi, Magnus Almgren, and Tomas Olovsson. CASAD: CAN-Aware Stealthy-Attack Detection for In-Vehicle Networks. *arXiv preprint arXiv:1909.08407*, 2019.

[26] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. Canet: An unsupervised intrusion detection system for high dimensional can bus data. *IEEE Access*, 8:58194–58205, 2020.

[27] Michael Müter and Naim Asaj. Entropy-based anomaly detection for in-vehicle networks. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 1110–1115, 2011.

[28] Kyong-Tak Cho and Kang G. Shin. Viden: Attacker identification on in-vehicle networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1109–1123, New York, NY, USA, 2017. Association for Computing Machinery.

[29] Martin Husák, Jana Komárková, Elias Bou-Harb, and Pavel Čeleda. Survey of attack projection, prediction, and forecasting in cyber security. *IEEE Communications Surveys Tutorials*, 21(1):640–660, 2019.

[30] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[31] Fredrik Gustafsson. Particle filter theory and practice with positioning applications. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):53–82, 2010.

[32] Liming Chen and Algirdas Avizienis. N-version programming: A fault-tolerance approach to reliability of software operation. In *Proc. 8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8)*, volume 1, pages 3–9, 1978.

[33] J-C Laprie, Jean Arlat, Christian Beounes, and Karama Kanoun. Definition and analysis of hardware-and software-fault-tolerant architectures. *Computer*, 23(7):39–51, 1990.

[34] Benjamin Cox, David Evans, Adrian Filipi, Jonathan Rowanhill, Wei Hu, Jack Davidson, John Knight, Anh Nguyen-Tuong, and Jason Hiser. N-Variant Systems: A Secretless Framework for Security through Diversity. In *15th USENIX Security Symposium*, pages 105–120, 2006.

[35] Andrea Höller, Tobias Rauter, Johannes Iber, and Christian Kreiner. Towards dynamic software diversity for resilient redundant embedded systems. In Alessandro Fantechi and Patrizio Pelliccione, editors, *Software Engineering for Resilient Systems*, pages 16–30, Cham, 2015. Springer International Publishing.

[36] Tsvika Dagan, Yuval Montvelisky, Mirco Marchetti, Dario Stabili, Michele Colajanni, and Avishai Wool. Vehicle safe-mode, concept to practice limp-mode in the service of cybersecurity. *SAE Int. J. Transp. Cyber. & Privacy 2 (2)*, feb 2020.

[37] Tasuku Ishigooka, Satoshi Otsuka, Kazuyoshi Serizawa, Ryo Tsuchiya, and Fumio Narisawa. Graceful degradation design process for autonomous driving system. In Alexander Romanovsky, Elena Troubitsyna, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 19–34, Cham, 2019. Springer International Publishing.

[38] Andreas Reschka, Gerrit Bagschik, Simon Ulbrich, Marcus Nolte, and Markus Maurer. Ability and skill graphs for system modeling, online monitoring, and decision support for vehicle guidance systems. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 933–939, 2015.

[39] Jose Rubio-Hernan, Rishikesh Sahay, Luca De Cicco, and Joaquin Garcia-Alfaro. Cyber-physical architecture assisted by programmable networking. *Internet Technology Letters*, 1(4):e44, 2018.

[40] Zhe Jiang, Neil C. Audsley, and Pan Dong. BlueVisor: A Scalable Real-Time Hardware Hypervisor for Many-Core Embedded Systems. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 75–84, 2018.

[41] Pekka Alho and Jouni Mattila. Service-oriented approach to fault tolerance in cpss. *Journal of Systems and Software*, 105:1 – 17, 2015.

[42] Meng Wu, Haibo Zeng, Chao Wang, and Huafeng Yu. INVITED: Safety guard: Runtime enforcement for safety-critical cyber-physical systems. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017.

[43] Luis F. Cómbita, Jairo Giraldo, Alvaro A. Cárdenas, and Nicanor Quijano. Response and reconfiguration of cyber-physical control systems: A survey. In *2015 IEEE 2nd Colombian Conference on Automatic Control (CCAC)*, pages 1–6, 2015.

[44] Youmin Zhang and Jin Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2):229 – 252, 2008.

[45] Mischa Möstl, Johannes Schlatow, Rolf Ernst, Nikil Dutt, Ahmed Nassar, Amir Rahmani, Fadi J. Kurdahi, Thomas Wild, Armin Sadighi, and Andreas Herkersdorf. Platform-Centric Self-Awareness as a Key Enabler for Controlling Changes in CPS. *Proceedings of the IEEE*, 106(9):1543–1567, 2018.

[46] Tatiana D. Nya, Stephan C. Stilkerich, and Christian Siemers. Self-aware and self-expressive driven fault tolerance for embedded systems. In *2014 IEEE Symposium on Intelligent Embedded Systems (IES)*, pages 27–33, Dec 2014.

[47] Sherali Zeadally, Teodora Sanislav, and George D. Mois. Self-Adaptation Techniques in Cyber-Physical Systems (CPSs). *IEEE Access*, 7:171126–171139, 2019.

[48] Danny Weyns. *Software Engineering of Self-adaptive Systems*, pages 399–443. Springer International Publishing, Cham, 2019.

[49] Hongjun Zhang, Baiqiao Huang, Peng Zhang, and Hongbin Ju. A New SoS Engineering Philosophy - Vitality Theory. In *2019 14th Annual Conference System of Systems Engineering (SoSE)*, pages 19–24, May 2019.

[50] George Vachtsevanos, Benjamin Lee, Sehwan Oh, and Michael Balchanos. Resilient design and operation of cyber physical systems with emphasis on unmanned autonomous systems. *Journal of Intelligent & Robotic Systems*, 91(1):59–83, 2018.

[51] Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw et al. *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*, pages 1–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[52] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures. *Reliability Engineering & System Safety*, 96(1):11 – 25, 2011. Special Issue on Safecomp 2008.

[53] Seungho Lee, Wonsuk Choi, Jo Hyo J., and Dong H. Lee. T-Box: A Forensics-Enabled Trusted Automotive Data Recording Method. *IEEE Access*, 7:49738–49755, 2019.

[54] Hafizah Mansor, Konstantinos Markantonakis, Raja Naeem Akram, Keith Mayes, and Iakovos Gurulian. Log your car: The non-invasive vehicle forensics. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 974–982, 2016.

[55] Dennis K. Nilsson and Ulf E. Larson. Conducting forensic investigations of cyber attacks on automobile in-vehicle networks. In *Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia and Workshop*, e-Forensics '08, Brussels, BEL, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[56] William Bortles, Sean McDonough, Connor Smith, and Michael Stogsdill. An introduction to the forensic acquisition of passenger vehicle infotainment and telematics systems data. Technical report, SAE Technical Paper, 2017.

[57] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial Examples: Attacks and Defenses for Deep Learning. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2805–2824, 2019.

[58] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[59] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805 – 822, 1999.

[60] Greg Welch and Gary Bishop. An Introduction to the Kalman filter. 1995.

[61] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[62] Burdi Motorworks. Mercedes Limp Home Mode. https://burdimotors.com/2017/11/30/mercedes-limp-home-mode, Accessed 2020-03-16, 2018.

[63] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.

[64] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015, 2015.

[65] Georg Macher, Eric Armengaud, Eugen Brenner, and Christian Kreiner. Threat and risk assessment methodologies in the automotive domain. *Procedia Computer Science*, 83:1288–1294, 2016.

[66] Olaf Henniger, Ludovic Apvrille, Andreas Fuchs, Yves Roudier, Alastair Ruddle, and Benjamin Weylr. Security requirements for automotive on-board networks. In *2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST)*. Institute of Electrical and Electronics Engineers (IEEE), oct 2009.

[67] Mafijul Md. Islam, Aljoscha Lautenbach, Christian Sandberg, and Tomas Olovsson. A risk assessment framework for automotive embedded systems. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security - CPSS 16*. Association for Computing Machinery (ACM), 2016.

[68] Thomas Rosenstatter and Tomas Olovsson. Towards a Standardized Mapping from Automotive Security Levels to Security Mechanisms. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1501–1507, Nov 2018.

[69] Van Huynh Le, Jerry den Hartog, and Nicola Zannone. Security and privacy for innovative automotive applications: A survey. *Computer Communications*, 132:17 – 41, 2018.

[70] Deborah Bodeau and Richard Graubart. Cyber Resiliency Engineering Framework (MITRE Technical Report MTR1-10237). *Bedford, MA: MITRE Corporation*, 2011.

[71] Benoit Baudry and Martin Monperrus. The multiple facets of software diversity: Recent developments in year 2000 and beyond. *ACM Comput. Surv.*, 48(1), September 2015.

[72] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, Cham, 2017.

[73] Christian Engelmann, Geoffroy R. Vallee, Thomas Naughton, and Stephen L. Scott. Proactive Fault Tolerance Using Preemptive Migration. In *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 252–257, 2009.

[74] Raffaele Romagnoli, Bruce H. Krogh, and Bruno Sinopoli. Design of Software Rejuvenation for CPS Security Using Invariant Sets. In *2019 American Control Conference (ACC)*, pages 3740–3745, 2019.

APPENDIX A
REMIND RESILIENCE GUIDELINES

In this section, we report in Table III resilience techniques that can be used against common security threats and attacks. We also describe the trade-offs when implementing these techniques.

TABLE III: REMIND Resilience Guidelines

Asset Hardware		**Attack** Fault Injection	
Resilience Strategy	**Resilience Technique**	**Trade-off**	
		Pros	Cons
Detection	• Statistical Techniques [13] • Machine Learning/Data Mining [13] • Localization (e. g., [28]) • Sensor/Data Fusion [13]	• Less computation is required. • No domain knowledge is needed. It handles multivariate and non-linear data. • Identifies the exclusive part causing the fault or attack. • Calculates a value of trust of the data sources derived from the normalization factor.	• Very sensitive to outliers, imprecise detection, and increased complexity when modelling non-linear data. • Requires training. Imprecise prediction: false positives and false negatives. Time penalty and resource consumption (power, processing, and storage). • Often applied offline. The precision of the localization is dependent on both, the number of observed parameters and the set frequency for probing monitored resources. • Imprecise detection: false positives and negatives. It also introduces time penalty (increase in execution time) and space penalty (increase in resource usage).
Mitigation	• Hardware Redundancy [11]–[13], [15], [17]–[20]	• Enables offsetting the effects of faults and attacks, and allows the progress of the system without loss of functionality.	• Time penalty (increase in execution time) and resource consumption (increase in required resources). Hardware costs independent of whether attacks occur. Also, the design and verification of replicas requires an effort.
Recovery	• Relocation/Migration [13], [19]	• Maintain system functionality in an operational state as it was before the fault or attack.	• May cause a degraded system, with less functionality, resources, and performance.
Endurance	• Self-aware Fault Tolerance [46]	• Enables systems to adapt their behavior when a fault or attack occurs in their environment, thus allowing a continuous operation of these systems.	• Complexity and resource consumption.
Asset Software		**Attack** Malware/Manipulated Software	
Detection	• Signature-based Detection [13] • Runtime Verification [13], [20]	• A precisely calibrated signature effectively identifies abnormal events during software execution. • Well-established and efficient technique to verify the correctness of software execution and monitor the behavior of the system.	• Does not work when designers and 3rd party suppliers (e.g., intellectual property providers) are not trusted. It cannot handle zero-day attacks and, thus, often used in combination with anomaly-based techniques leading to an increased resource consumption and time penalty. • Limited coverage. The used monitoring algorithms usually handle a single execution trace which limits the scope of the verification.

TABLE III – *Continued on next page*

TABLE III – *Continued from previous page*

Resilience Strategy	**Resilience Technique**	**Trade-off**	
		Pros	Cons
Mitigation	• Software Redundancy [11]–[13], [15], [17]–[20] • N-Version Design [11]–[13], [17], [19] • Agreement/Voting [11], [13], [17], [20] • Recovery Blocks [11], [19], [20] • N self-checking [17]	• Helps to contain and exclude malicious behavior (i.e., reduces likelihood of harm). Enable restoration in case of disruption. Enhances the availability of critical capabilities. • Helps to mitigate the impact of failures when a risk is introduced to system design or configuration. • Typically combined with redundancy. Can be used to select, for instance, the average or median of the results provided by the redundant sources. • Uses different implementations of the same design specification to provide tolerance of design faults. • Provides mitigation by creating N versions of the same software, each with its own acceptance test. The version that passes its own acceptance test is selected through an acceptance voting system.	• Resource consumption. It demands the protection of redundant resources. It can degrade over time as configurations are updated or connectivity changes. It is often applied with diversity techniques which increases complexity and leads to scalability issues. • Requires much effort for designing, implementation, testing, and validation of the N independent versions. • Attackers may exploit the voting process in order to force the system to a degraded mode. • Requires extra verification and validation effort and, thus, more resource consumption. It might be difficult to create alternative software implementations without any correlation between the various versions. • Causes an increase in required resources and execution time.
Recovery	• Preemptive Migration [18], [19] • Checkpoint Recovery [11], [17]–[20] • Software Rejuvenation [11], [19]	• Prevents failures from impacting running parallel applications by enabling the migration of running software from one virtual machine to another in real time. • Helps the system to resume its operation in a state free of the effects of the fault or attack. Frequent checkpointing reduces the amount of lost work. • Helps avoiding the costs of failures from software degradation, as periodic (graceful) restarts of the software component allow the release and re-allocation of memory, thus, operation in a clean state.	• Lack of standardized metrics for measuring and evaluating the health and interfaces between system components. • Overhead in relation to the size and frequency of created checkpoints. Creating a checkpoint, for instance, requires interrupting the normal operation of a system to record the checkpoint. Moreover, it requires storage resources to store the checkpoint. The created checkpoints might potentially contain an error or intrusion that has not been detected yet. Globally consistent checkpoints are not trivial to obtain in a distributed system, due to e. g., variation of the local clock, parallel computation and possible different system states. • Requires shutting the software down and restarting it periodically which causes the software to be unavailable for the duration of the restart. It is often a slow process requiring an extra overhead.
Endurance	• Platform-centric Self-awareness [45] • Secure Logging (e. g., [52]–[54])	• Enables systems to recognize their own state and to continuously adapt to change, evolution, system interference, environment dynamics, and uncertainty. It optimizes resilience, quality of service, and supports system dynamics and openness. It also helps to reduce uncertainties and identify inconsistencies. • Prevents modifying the logs by using e.g., chained hashes. It enables storing security-related events containing information about e.g., flash operations, external interactions, and power downtime. This information helps to reconstruct events, detect intrusions and identify problems.	• Automatically maintaining coherent specifications that capture and monitor security is a challenging task. Complexity, scalability, and difficulty in dealing with uncertainties and inaccuracies. The determination of relevant dependencies in a complex system is also challenging. • Resource consumption and time penalty. Moreover, missing authentication and lack of cryptographic means to ensure data integrity can limit the potential of the logging.

TABLE III – *Continued on next page*

TABLE III – *Continued from previous page*

Resilience Strategy	**Resilience Technique**	**Trade-off**	
		Pros	Cons
Asset Network/Communication		**Attack** Fabrication/Jamming	
Detection	• Specification-based Anomaly Detection (e. g., [24]) • Localization (e. g., [28]) • Verification of Safety-Properties [13]	• Helps detecting anomalies in the system's behavior by reporting the specific deviation that has been observed. • Identifies the exclusive part causing the fault or attack. • Ensures that the system does not evolve in unsafe state starting from some initial conditions.	• Needs of resources for detection and processing of collected information (e.g., costly intelligent sensors). Domain knowledge is required to specify normal behavior. Specifications need to be adapted for each specific vehicle configuration otherwise risk of high false positives or negatives. • Requires additional resources. • It is limited to small scale systems.
Mitigation	• Isolation [11], [13] • Restructure [11]	• It provides a remedy to enable the system to continue its operation by offsetting the effect of the attack. Also, it prevents loss of functionality. • Helps to mitigate incorrectness in the interactions between the components or subsystems by excluding the affected part from interacting with the rest of the system, and maintaining system functionality.	• Introduces a time penalty and an increase in required resources (e.g., replica modules that are used to compensate for isolating the affected component of the system). • May cause an operation of the system in a degraded condition which influences its performance and incurs additional time overhead to the system.
Recovery	• Relocation/Migration [13], [19] • Re-instantiation/Restart [11], [13], [17], [19]	• Maintain system functionality in an operational state as it was before the fault or attack. • Helps to restore the system to its initial state when the impact of the attack can not be handled in another manner. It guarantees that the impact of the attack is completely removed.	• May cause a degradation in the operation of the system which influences the performance and functionality thereof. • Restoring the system to its initial state causes lost data, such as privacy related data (e. g., location, speed, driving behavior) and workshop data (e. g., vehicle health, engine data and emissions). The impact of the lost data depends on the type of data and the current need for it. In addition, the re-instantiation of safety-critical functions may require the vehicle to be in standstill.
Endurance	• Self-adaptation [47], [48]	• Ensures a secure, reliable, and predictable communication between system components and between the system and its environment. Supports and maintains an acceptable level of service despite the occurrence of faults and other factors that affect normal operations. Seamlessly adapts to different network loads and reacts to security threats and other disturbances in the environment.	• Complexity and resource consumption.
Asset Network/Communication		**Attack** Masquerading/Spoofing/Collision	
Detection	• Information-theoretic Detection [13] • Falsification-based Analysis [13]	• Helps to detect anomalies by analyzing available audit logs and records (e.g., entropy measures) and comparing these records with defined normal behaviors. More records enhance the precision of the detection. • Provides an indication (i.e., a robustness degree) to what extent temporal logic properties are from satisfying or violating a specification.	• Time penalty for processing audit records. More records at disposal increases the processing time and complexity. On the other hand, a low number of records leads to an imprecise detection with more false positives and false negatives. • Imprecise detection: false positives and false negatives

TABLE III – *Continued on next page*

TABLE III – *Continued from previous page*

Resilience Strategy	Resilience Technique	Trade-off	
		Pros	Cons
Mitigation	• Rescue Workflow [18], [19] (adaptation may be necessary) • Dynamic Deployment of Policies [15]	• Enables the system to continue operation after the failure of the task until it is unable to proceed without amending the fault or attack. Already finished tasks do not need re-execution, thus saving time and resources. • Takes the dynamic and changing nature of attacks into account. Deploys different defense policies depending on the attack, for example, it can modify the executed actions while the attack is going on.	• It may lead to a decrease in the quality of service. Time penalty might be caused by re-computing and migrating the tasks which cause the problem. • Leads to performance overhead. Moreover, it always requires runtime permissions which may not be present when running normally. Complexity.
Recovery	• Checkpoint Recovery [11], [17]–[20] • Re-instantiation/Restart [11], [13], [17], [19]	• Helps the system to resume its operation in a state free of the effects of the fault or attack. Frequent checkpointing reduces the amount of lost work. • Helps to restore the system to its initial state when the impact of the attack can not be handled in another manner. It guarantees that the impact of the attack is completely removed.	• Overhead in relation to the size and frequency of created checkpoints. Creating a checkpoint, for instance, requires interrupting the normal operation of a system to record the checkpoint. Moreover, it requires storage resources to store the checkpoint. The created checkpoints might potentially contain an error or intrusion that has not been detected yet. Globally consistent checkpoints are not trivial to obtain in a distributed system, due to e. g., variation of the local clock, parallel computation and possible different system states. • Restoring the system to its initial state causes lost data, such as privacy related data (e. g., location, speed, driving behavior) and workshop data (e. g., vehicle health, engine data and emissions). The impact of the lost data depends on the type of data and the current need for it. In addition, the re-instantiation of safety-critical functions may require the vehicle to be in standstill.
Endurance	• Secure Logging (e. g., [52]–[54])	• Prevents modifying the logs by using e.g., chained hashes. It enables storing security-related events containing information about e.g., flash operations, external interactions, and power downtime. This information helps to reconstruct events, detect intrusions and identify problems.	• Resource consumption and time penalty. Moreover, it requires authentication and cryptographic means to ensure data integrity and confidentiality.
Asset Network/Communication		**Attack** Hijacking/Replay/Suspension/DoS	
Detection	• Signature-based Detection [13] • Verification of Safety-Properties [13]	• A precisely calibrated signature effectively identifies abnormal events during software execution. • Ensures that the system does not evolve in unsafe state starting from some initial conditions.	• Does not work when designers and intellectual property providers are not trusted. It cannot handle zero-day attacks and, thus, often used with Anomaly-based techniques leading to a increased resource consumption and time penalty. • It is limited to small scale systems.
Mitigation	• Reparameterization [13] • Isolation [11], [13] • Graceful Degradation [13], [15]	• Enables adaptation by switching the configuration parameters of the compromised component to another configuration. • It provides a remedy to enable the system to continue its operation by offsetting the effect of the attack. Also, it prevents loss of functionality. • Prevents a catastrophic failure of the system. It enables a system to continue functioning even after parts of the system have been compromised. It shuts down less critical functions to allocate the resources to more critical functions to maintain availability.	• Decreases the quality of service. • Introduces a time penalty and an increase in required resources (e.g., replica modules that are used to compensate for isolating the affected component of the system). • Causes a degradation in the performance of the operations and services of the system.

TABLE III – *Continued on next page*

TABLE III – *Continued from previous page*

Resilience Strategy	**Resilience Technique**	**Trade-off**	
		Pros	Cons
Recovery	• Relocation/Migration [13], [19] • Software Rejuvenation [11], [19] • Reinitialization [11]	• Maintain system functionality in an operational state as it was before the fault or attack. • Helps avoiding the costs of failures from software degradation, as periodic (graceful) restarts of the software component allow the release and re-allocation of memory, thus, operation in a clean state. • Applied in conditions in which the mitigation is deemed impossible. Restores or pristine resets the system to its initial state.	• May cause an operation of the system in a degraded condition which influences its performance. • Requires shutting the software down and restarting it periodically which causes the software to be unavailable for the duration of the restart. It is often a slow process requiring an extra overhead. • Causes loss of work, and accordingly leads to a waste of resources.
Endurance	• Attack Analysis / Reconstruction (e. g., [55], [56])	• Helps to enhance resilience by systematically and empirically analyzing attacks as well as used technologies (potential entry point, e.g., Bluetooth and WiFi) that interact with the external environment.	• Resource consumption and analysis effort.
Asset Data Storage		**Attack** Unauthorized Read/Manipulation	
Detection	• Signature-based Detection [13] • Specification-based Anomaly Detection (e. g., [24])	• A precisely calibrated signature effectively identifies abnormal events during software execution. • Helps detecting anomalies in the system's behavior by reporting the specific deviation that has been observed.	• Does not work when designers and intellectual property providers are not trusted. It cannot handle zero-day attacks and, thus, often used with Anomaly-based techniques leading to a increased resource consumption and time penalty. • Needs of resources for detection and processing of collected information (e.g., costly intelligent sensors). Domain knowledge is required to specify normal behavior. Specifications need to be adapted for each specific vehicle configuration otherwise risk of high false positives or negatives.
Mitigation	• Redundancy [11]–[13], [15], [17]–[20] • Isolation [11], [13]	• It enables data backup and restore by replicating information and data sources. • It provides a remedy to enable the system to continue its operation by offsetting the effect of the attack. Also, it prevents loss of functionality.	• Requires extra resources for data storage. • Introduces a time penalty and an increase in required resources (e.g., replica modules that are used to compensate for isolating the affected component of the system).
Recovery	• Dynamic Deployment of Policies [15]	• Takes the dynamic and changing nature of attacks into account. Deploys different defense policies depending on the attack, for example, it can modify the executed actions while the attack is going on.	• Leads to performance overhead. Requires runtime permissions which may not be present when running normally. Complexity.
Endurance	• Secure Logging (e. g., [52]) • Attack Analysis / Reconstruction (e. g., [55], [56])	• Prevents modifying the logs by using e.g., chained hashes. It enables storing security-related events containing information about e.g., flash operations, external interactions, and power downtime. This information helps to reconstruct events, detect intrusions and identify problems. • Helps to enhance resilience by systematically and empirically analyzing attacks as well as used technologies (potential entry point, e.g., Bluetooth and WiFi) that interact with the external environment.	• Resource consumption and time penalty. Moreover, missing authentication and lack of cryptographic means to ensure data integrity can limit the potential of the logging. • resource consumption and analysis effort.

APPENDIX B
PROPOSED AUTOMOTIVE SOLUTIONS

In Table IV we provide a description of the solutions referred to in Figure 1. This overview of specific solutions should be considered as a starting point for interested readers and is by no means complete.

TABLE IV: TECHNIQUES AND SOLUTIONS RELEVANT FOR THE AUTOMOTIVE DOMAIN.

Pattern	**Technique**	**Solution**
DETECTION		
Specification-based	Runtime Verification	Heffernan et al. [23] use the automotive functional safety standard ISO 26262 as a guide to derive logical formulae. They demonstrate the feasibility of their proposed runtime verification monitor with an automotive gearbox control system as use case.
	Specification-based Anomaly Detection	Müter et al. [24] describe eight detection sensors that are applicable for the internal network of automotive systems. Six of these sensors are specification-based, e. g., the frequency of specific message types and the range of transmitted values like speed.
Anomaly-Based	Statistical Techniques	Nowdehi et al. [25] propose an IDS that learns about the automotive system by learning from samples of normal traffic without requiring a model definition.
	Machine Learning	Hanselmann et al. [26] propose CANet an unsupervised IDS for the automotive CAN bus. The anomaly score is calculated using the error between the reconstructed signal and the true signal value.
	Information-theoretic	Müter et al. [27] design an entropy-based IDSs for automotive systems with experimental results using data from a vehicle's CAN-Body network.
	Localization	Cho and Shin [28] present a scheme identifying the attacking ECU based on fingerprinting the voltage measurements on the CAN bus for each ECU. We see great opportunities in the localization of attacks when considering a centralized vehicle architecture combined with virtualisation techniques. This allows us to get detailed performance metrics of virtualized vehicle functions.
Predicting Faults and Attacks	Attack Prediction	Husák et al [29] perform a survey about current attack projection and prediction techniques in cybersecurity.
Redundancy	Diversity Techniques	Baudry and Monperrus provide in their survey [71] an overview of different software diversity techniques.
	Adaptive Software Diversity	Höller et al. [35] introduce an adaptive dynamic software diversity method. The diversification control receives error information from the decision mechanism and randomizes specific parameters during execution. Their experimental use cases demonstrate the dynamic reconfiguration of ASLR parameters, respectively, random memory gaps.
MITIGATION		
Adaptive Response	Model-based Response	Cómbita et al. provide a survey on response and reconfiguration techniques for cyber-physical control systems. Controllers or other systems that can be modelled as a control loop can be, for instance, adjusted to have another module in the feedback loop that compares the actual feedback from the control loop with a simulated/modelled response of what is expected.
Runtime Enforcement	Safety Guard	Wu et al. [42] show how so-called safety guards can be applied to safety-critical Cyber-Physical Systems (CPSs).
Reconfiguration and Reparametrisation	Graceful Degradation	Dagan et al. [36] provide an architectural design on how to extend limp modes so that they can be additionally used in a cyber security context. A safe-mode manager sends out triggering messages that cause the ECUs to transition to a limp mode when cyber-breaches are detected.
		Ishigooka et al. [37] propose a graceful degradation design process for autonomous vehicles with focus on safety.
		Reschkka et al. [38] explore how skills and ability graphs can be used for modelling, on-line monitoring and supporting decision making of driving functions.

TABLE IV – *Continued on next page*

TABLE IV – *Continued from previous page*

Pattern	Technique	Solution
	Restructure	Segovia et al. [15] set the focus of their survey on software reflection as mitigation technique for SCADA systems. Software reflection enables the system itself to examine and change its execution behaviour at runtime, which allows, for instance, the system to take actions when an attack is detected. The drawbacks currently seen in software reflection are the performance overhead, the increased execution time and the extended permissions required by software reflection.
	Dynamic Deployment of Policies	Rubio-Hernan et al. [39] propose an architecture for CPS that combines feedback control loops with programmable networking in order to mitigate attacks by re-routing traffic or applying security rules.
RECOVERY		
Migration	Relocation/Migration	Jiang et al. [40] propose a hypervisor that meets real-time requirements.
		Other relocation techniques are microservices [72]. Pekka and Mattila [41] propose a service-oriented architecture for real-time CPSs.
	Pre-emptive Migration	Engelmann et al. [73] describe a pre-emptive migration technique which uses a feedback-loop for observing health parameters to detect behaviour indicating a fault. This solution was developed for high performance computing and its applicability for the automotive domain needs to be further investigated.
Checkpointing and Rollback	Software Rejuvenation	Romangnoli et al. [74] describe a method to decide when it is safe to reload the software of a CPS.
ENDURANCE		
Self-*	Continuous Change	Möstl et al. [45] identify in their work the challenges of continuous change and evolution of CPS and propose two frameworks for self-aware systems centring around self-modelling, self-configuration and self-monitoring. The controlling concurrent change (CCC) framework is concerned with how to deal with changes in software components during the lifetime of a CPS. The authors highlight that the well-established V-model currently used is not designed for continuous change and therefore parts of the integration testing and system validation and verification need to be moved to the system itself. The proposed framework includes an automated integration process for new or updated functions that addresses safety, security, availability and real-time requirements. The structure and workflow of the proposed framework is further described using an automotive use case. The second framework concentrates on optimising performance, power consumption and resilience of CPS by using self-organisation and self-awareness techniques.
Verificaton & Validation	Challenges in V&V	De Lemos [51] discuss research challenges of verification and validation for self-adaptive systems at runtime.
Robustness	Adversarial Attacks on DNN	Yuan et al. [57] give an overview of current adversarial attack and defence techniques for deep learning.
Forensics	Secure Logging	Lee et al. [53] describe T-Box a secure logging solution for automotive systems that makes use of the trusted execution environment in ARM TrustZone.
		Mansor et al. [54] propose a framework to log vehicle data, such as diagnostic transmission codes, via the mobile phone and store it on a secure cloud storage.
	Attack Analysis / Reconstruction	Nilsson and Larson [55] discuss the requirements for conducting forensic investigations on the in-vehicle network.
		Bortles et al. [56] present which types of data may be retained from current infotainment and telematic systems.