

Modelling and analysing cyber-resilience of an autonomous vehicle with zonal architecture with SCORE+

Hanna Klaudel

0000-0003-0790-0004

Université Paris-Saclay, Univ Evry, IBISC, 91020, Evry, France

Email: hanna.klaudel@univ-evry.fr

Witold Klaudel

0000-0002-7039-0994

SafeTech Cybernetics, Palaiseau, France,

Email: witold.klaudel@outlook.fr

Abstract—We present a large-scale case study of cyber resilience in a highly automated vehicle based on a zonal architecture, using our original methodology, called SCORE+, designed to detect cyberattacks in distributed information systems. This methodology enables the construction of a formal (automata based) model of cyberattack propagation by exploiting the system's software and hardware architecture, along with a set of cyberattack propagation patterns. This model can be analysed using model-checking techniques, allowing the evaluation of the effectiveness of system defence mechanisms.

I. INTRODUCTION

TODAY'S safety-critical systems must be resilient and often need to comply with legal requirements that include a formalized process for risk analysis. Resilience is understood here as the ability of a system to operate under adverse conditions or stress, even if in a degraded mode, while maintaining essential operational capabilities, and to recover to a nominal operational mode.

In this paper, we use an improved version of SCORE [12], [11], called SCORE+, to develop a large-scale case study of a high automation (Level 4 [1]) autonomous vehicle utilizing zonal architecture, in which we focus on modeling and analyzing cyber-resilience, with particular emphasis on the relocation of critical functions in response to cyberattacks. A cyberattack is defined here as a sequence of (potentially branching) unitary actions performed by an *attacker* to compromise certain components of the system. This sequence begins with one or more entry points and culminates in the loss of integrity of some system components, ultimately causing the damage targeted by the attacker.

II. RELATED WORK AND CONTRIBUTION

Technological advances continuously expand the cyberattack surface of distributed information systems and bring sophisticated implementations of resilience understood as active defence, capable to detect attacks and react to them. Using formal methods to mathematically verify that the system will behave as expected even under attack scenarios can be useful for high assurance in safety-critical systems where attack could have catastrophic consequences. In most formal approaches, Attack Trees [22], [18], or their variants such as forests [25]

and graphs [6], [20], [15], [24], are used to model how attacks might unfold. Later, they were extended to include defence mechanisms to form Attack-Defence Trees [3]. The edges of an attack tree represent unitary actions that enable the attacker to transition the system into a compromised state. Attack trees also support quantitative analysis, allowing the evaluation of factors such as costs, probabilities, or likelihoods, either globally or for specific sub-attacks [5]. This quantitative framework enables integration with cybersecurity measures [4], [18], which can then be optimized to achieve an acceptable level of risk.

The creation of an attack tree is typically assigned to security experts and remains a manual process. However, due to the tedious and error-prone nature of manually constructing attack trees, various methods have been proposed for automating this process [26], [25], [21], [16], [8], [19], [15]. These approaches use a set of *atomic actions* that an attacker can perform to compromise the system and drive it into an insecure state. This set of atomic actions includes both known vulnerabilities present in the system and zero-day (unknown) flaws that the expert may choose to account for. The system being analyzed is assumed to be secure, except for vulnerabilities explicitly introduced by atomic actions, which follow specific propagation rules defined by experts.

In contrast to the approaches mentioned above, SCORE does not rely on a predefined set of known vulnerabilities. Instead, it assumes that each system component can be compromised at a cost to the attacker, which varies depending on the cybersecurity protections in place. This allows for the discovery of potential attacks based on the strength of the cybersecurity measures chosen for each component. This high level of abstraction results from the motivation behind the method: estimation of general resilience based on architectural traits and not an identification of concrete vulnerabilities/exploits as understood e.g. within the Common Vulnerabilities and Exposures (CVE) database [23]. This is a substantial difference to attack graph-based vulnerability/exploit search tools [26], [14], [9] often connected to popular vulnerability databases [10].

The propagation model in SCORE is derived from both

the software and hardware architecture descriptions of the analyzed system and from a set of generic, unitary attack propagation rules linked to classes of software components and communication protocols. This leads to a formal model of a network of automata, where each automaton represents a software component, encoding its knowledge of shared secrets (whether stolen or not) and its operational status (such as functional, unavailable, having corrupted data, or infected with malware). The attacker's presence is reflected by a non-functional status of certain automata, indicating compromised states. Each automaton evolves based on a predefined set of attack propagation rules. Since the model consists of a network of automata, it is compatible with off-the-shelf model checkers, which can be used to explore temporal logic properties that describe the progression of an attack from its initial state, including the attacker's point of entry. See also [17] for more information on SCORE+.

III. CASE STUDY

Our case study concerns a new type of autonomous vehicle architecture, known as zonal architecture, which incorporates various resilience strategies. Unlike the classical architectures discussed in [12], the zonal architecture features geographical division, network redundancy, and a reduced number of computational units, all while maintaining sufficient computing resources to enable critical function relocation.

As represented in Fig. 1 and 2, the use case architecture is composed of two main parts: the zone oriented **Vehicle** part comprising

- two Central Zones (Primary and Secondary), and
- four Peripheral vehicle Zones (front right, front left, back right and back left).

and a simplified representation of the **External World** comprising

- Internet with a content provider and cellular network,
- Card and Phone used for accessing the vehicle,
- Dealership with the capability of vehicle diagnostics and software update,
- OEM (car maker) with the central management of vehicle software and navigation map delivery,
- GNSS (Global Navigation Satellite System) responsible of the vehicle geographical positioning.

The four peripheral zones are responsible for managing sensors and actuators associated with the vehicle's geographical zone. These include systems like wipers, lights, air conditioning, doors, windows, and distributed perception sensors such as cameras and radars.

Each Central Zone contains two computing units: one dedicated to high-level functionalities, including perception, decision-making, cabin management, and multimedia, and the other focused on real-time control.

- The Primary Central Zone (PCZ) specialises in controlling the vehicle's trajectory and speed, utilizing perception sensors and actuators such as brakes, steering, and the engine.

- The Secondary Central Zone (SCZ) on the other hand, is mainly responsible for cabin management and human interaction.

A notable feature of this architecture is the ability for PCZ and SCZ to seamlessly take over each other's critical functions in the event of a failure. This is made possible by the flexible network connectivity and the duplication of perception sensors, ensuring system reliability and fault tolerance.

Both, the **Vehicle** and the **External World**, have their hardware and software architectures, as shown in Fig. 1 and 2, respectively.

Regarding hardware architecture, all perception sensors are connected via Ethernet to the high-level computing units in PCZ and SCZ. Similarly, all low-level sensors (e.g. brakes, steering, cabin sensors) are connected through CAN to the real-time computing units in PCZ and SCZ.

Concerning the software architecture, we distinguish the following categories of components:

- kernels, depicted with a dashed border,
- components with root privileges, depicted in light pink with a brown border,
- user space components, depicted in light blue with a dark border.

Each software component is associated with its corresponding hardware component, which is highlighted against a dark background at the bottom.

Special attention is paid to the software components responsible for diagnostics and updates, as they have root privileges and are present on each hardware component. These components are identified by names in the form xxxDU.

Interactions are represented by links, and in some cases, by pairs of color-coded labels, particularly when tracing lines becomes too complex. However, there is no semantic difference between these two representations.

A hardware component can host no more than one kernel. Although explicit interactions with kernels are not shown, each kernel implicitly manages all system interactions with the software components hosted on the same hardware component. Hardware components without a kernel can only host software components with root privileges.

Potential attacker entry points to the system are also indicated in the architecture definition. They appear in orange in Fig. 1 with a Trojan icon. We consider the following entry points for attackers:

- IntHacker is located on the Internet;
- DealHacker is located in the dealership;
- OemHacker is located in the intranet of the carmaker (OEM);
- GnssHack can attack the GNSS radio transmission;
- BtHacker can attack the Bluetooth connectivity of the vehicle;
- CardHack can attack via the radio link used by the vehicle's access card;
- InCarHack is located on the Ethernet bus of the vehicle.

Each of the above hackers has a dedicated software component in Fig. 2 which is a kernel surrounded by a red box.

TABLE I
FALLBACK ACTIONS RAISED BY ATTACK TENTATIVE ON SOFTWARE COMPONENTS.

Software components	Status targeted by the attacker	Fallback action
CommUnitK	$\mathcal{M}$	
CommDU, ProxyInt, ProxyMap, ProxyDU	$\mathcal{B}, \mathcal{M}$	stopExt
MCentrK, PrimCtrlK	$\mathcal{N}, \mathcal{M}$	
Perception, Decision, TrajCtrl	$\mathcal{N}, \mathcal{M}, \mathcal{B}$	
PrimCtrlDU	$\mathcal{M}, \mathcal{B}$	stopPrim
ScndCentrK, ScndCtrlK, CarDU	$\mathcal{N}, \mathcal{M}$	
Navig, BodyMngt	$\mathcal{N}, \mathcal{M}, \mathcal{B}$	
Browser, Multimedia, ScndCentrDU	$\mathcal{M}, \mathcal{B}$	stopSecond

TABLE II
FOUND PATHS AND MINIMAL COSTS OF ATTACKS.

	Attack path (software component/status:unitary cost)	cost
1_A	IntHacker/M $\rightarrow$ ProxyMap/B:130 $\rightarrow$ MapSrv/B:10 $\rightarrow$ Decision/B:70 $\rightarrow$ TrajCtrl/B:5 $\rightarrow$ Steering/B:5	220
2_A	BtHacker/M $\rightarrow$ Multimedia/B:15 $\rightarrow$ ScndCentrK/M:60 $\rightarrow$ Navig/B:10 $\rightarrow$ Decision/B:20 $\rightarrow$ TrajCtrl/B:5 $\rightarrow$ Steering/B:5	115
3_A	InVehicleHack/M $\rightarrow$ Perception/B:10 $\rightarrow$ Decision/B:5 $\rightarrow$ TrajCtrl/B:5 $\rightarrow$ Steering/B:5	25
1_B	IntHacker/M $\rightarrow$ ProxyMap/B:130 $\rightarrow$ MapSrv/B:10 $\rightarrow$ Decision/B:100 $\rightarrow$ TrajCtrl/B:35 $\rightarrow$ Steering/B:5	280
2_B	BtHacker/M $\rightarrow$ ScndCentrK/M:130 $\rightarrow$ Decision/B:80 $\rightarrow$ TrajCtrl/B:35 $\rightarrow$ Steering/B:5	250
3_B	InVehicleHack/M $\rightarrow$ TrajCtrl/B:75 $\rightarrow$ Steering/B:5	80

feared events which, in this context, are associated with vehicle functionalities and represent potential harmful breaches. Each feared event is assigned a severity level and an acceptable level of risk. Together, these factors allow us to estimate the maximal accepted likelihood of attacks that could trigger each feared event. We assume here that the list of feared events is already known along with the related acceptable likelihood, represented by its inverse, *i.e.*, the acceptable cost, which is for example 120 for steering sabotage. We also assume that the unitary costs are already defined for all protocols and all types of software components used in our architectures, taken from libraries provided by experts. Intuitively, the costs may be divided in categories, for example weak (under 10), intermediate (between 10 and 50) and strong (above 50).

From that, our method allows identifying potentially harmful attacks that could trigger the feared events mentioned earlier, assessing their costs to either confirm that the risk level remains acceptable or suggest modifications to enhance protections. The analysis presented below is intended to illustrate the method. It is, of course, partial and covers only a small subset of attacks, which should be considered in an exhaustive industrial cybersecurity risk assessment.

For our analysis we consider two versions *A* and *B* of monitoring and fallback actions defined in the system.

A - the monitoring is only implemented on the software components placed on the communication unit CommUnit and the fallback disconnects all of them if compromised, and

B - in addition to what is present in *A*, the monitoring is activated on all components responsible for vehicle

movement control (such as Decision or Trajectory Control) and the fallback relocates the essential functions to the part which is not compromised.

For both versions, we assume the system initially operates in nominal mode and focus on attacks targeting vehicle movement, specifically those capable of compromising the **Steering** software component. For each version, we consider three initial configurations, each representing a different entry position of the attacker.

- 1) a hacker present on Internet;
- 2) a hacker connected by Bluetooth to multimedia, and
- 3) a hacker already present inside the vehicle, which represents the situation where other protections did not work.

In order to compute the six corresponding attack paths of minimal costs, *i.e.*, 1_A , 2_A , 3_A , 1_B , 2_B , 3_B , we set for each case the initial configuration in the propagation model expressed in UPPAAL and use the model checker. The UPPAAL formula allowing to find the minimal cost of an attack of a given target may be prohibitively long to compute due to the size of the model and the combinatorial explosion of the number of reachable configurations. To cope with this problem, the method consists in first assuming that such an attack exists and use the model checker to confirm that it is under a given estimated cost, which is generally much faster.

For example, in case 1_A of Tab. II we set the status of IntHacker to $\mathcal{M}$ in the initial configuration, which means that we assume that an Internet hacker is active. Then, with the UPPAAL formula $\inf\{\text{Steering.B}\}:\text{cost}$ we search for the minimum value of the variable `cost` along an attack path

that reaches a configuration where the component **Steering** has status $\mathcal{B}$, i.e., where the vehicle steering is compromised. However, as mentioned above, it is not efficient without constraining the cost to some estimated limit. To find such a limit, we check under different cost constraints the UPPAAL CTL formula $E\langle\langle \text{Steering}.\mathcal{B} \rangle\rangle$ saying that there is a path reaching eventually the configuration where the component **Steering** has $\mathcal{B}$ status. We start from largest estimations of `cost` and refine them by dichotomy until obtaining an acceptable computation time. Usually this stage lasts under 30m. If the found cost constraint is close to the minimum, the `inf` formula becomes efficient and we obtain a result in few minutes.

Tab. II shows the attacks identified for the six configurations mentioned above along with their calculated minimal costs and the corresponding attack paths. Notably, in case 1_A , the vehicle in version A already has strong protections against attacks originating from the Internet. Adding code relocation, as shown in 1_B , does not significantly change the situation.

In contrast, there is a notable difference in cases 2_A and 2_B , where the attack is launched directly via the Bluetooth interface, bypassing the primary protections against Internet-based attacks and where the target is closer to the essential functions of the vehicle. However, continuous monitoring and the activation of fallback actions provide a satisfactory level of protection.

Finally, cases 3_A and 3_B illustrate a scenario where the attacker has bypassed peripheral security measures. Here, relocating essential functions introduces additional security barriers, significantly enhancing the overall protection level.

Tab. II highlights the protective benefits of relocation. However, the fallback mechanism also introduces a new vulnerability: the potential for attackers to easily trigger degraded modes. For example, the attacker `IntHacker` can bypass `ProxyInt` and attempt to corrupt `Multimedia`, which triggers the relocation action on `PCZ`, resulting in a total cost of 80. This cost may be considered too low to effectively deter such an attack.

IV. CONCLUSION AND PERSPECTIVES

Compared to our previous study [12], which introduced the main concepts of our approach and illustrated them through a classical (non-zonal) vehicle architecture, the present work reports several new contributions, mainly at the specification and optimisation levels. We extended our models and supporting tools, which required a substantial redesign, extension, and optimization of our approach to the management of complex and multiple fallback mechanisms, in order to maintain an acceptable efficiency. These improvements enabled the specification and analysis of a representative modern zonal architecture, including approximately twice as many software components as the non-zonal architecture studied in [12].

Acknowledgments: During the preparation of this work, the authors used ChatGPT at the stage of editorial correction without any impact on the research results. The authors take full responsibility for the content of this publication.

REFERENCES

- [1] SAE levels of driving automation refined for clarity and international audience. <https://www.sae.org/blog/sae-j3016-update>. Accessed: (February 1st, 2025).
- [2] Ebios risk manager. French Cybersecurity Agency, 2024. https://www.ssi.gouv.fr/uploads/2019/11/anssi-guide-ebios_risk_manager-en-v1.0.pdf.
- [3] J. Arias, C. E. Budde, W. Penczek, L. Petrucci, T. Sidoruk, and M. Stoelinga. Hackers vs. security: attack-defence trees as asynchronous multi-agent systems. In *Formal Engineering Methods*, pages 3–19. Springer, 2020.
- [4] Z. Aslanyan and F. Nielson. Pareto efficient solutions of attack-defence trees. volume 9036, pages 95–114, 2015.
- [5] A. Bagnato, B. Kordy, P. H. Meland, and P. Schweitzer. Attribute decoration of attack–defence trees. *International Journal of Secure Software Engineering*, 2012.
- [6] M. S. Barik, A. Sengupta, and C. Mazumdar. Attack graph generation and analysis techniques. *Defence Science Journal*, 66(6):559, 2016.
- [7] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In *LNCS*, volume 3185, pages 200–236. Springer, 2004.
- [8] F. Di Maio, R. Mascherona, and E. Zio. Risk analysis of cyber-physical systems by `gst-mld`. *IEEE Systems Journal*, 14(1):1333–1340, 2019.
- [9] Mengmeng Ge, Jin B Hong, Walter Guttman, and Dong Seong Kim. A framework for automating security analysis of the internet of things. *Journal of Network and Computer Applications*, 83:12–27, 2017.
- [10] Chris Hankin, Pasquale Malacaria, et al. Attack dynamics: an automatic attack graph generation framework based on system topology, capec, cwe, and cve databases. *Computers & Security*, 123:102938, 2022.
- [11] G. Hutzler, H. Klaudel, W. Klaudel, F. Pommereau, and A. Rataj. Automatic discovery of cyberattacks. In *IEEE International Conference on Cyber Security and Resilience, CSR 2024*, pages 1–8. IEEE, 2024.
- [12] G. Hutzler, H. Klaudel, W. Klaudel, F. Pommereau, and A. Rataj. An autonomous vehicle in a connected environment: case study of cyber-resilience. In *FedCSIS*, volume 39 of *Annals of Computer Science and Information Systems*, pages 363–373, 2024.
- [13] Road vehicles, Cybersecurity engineering. Standard, International Organization for Standardization, Geneva, CH, 2021.
- [14] Sushil Jajodia, Steven Noel, and Brian O’berry. Topological analysis of network attack vulnerability. *Managing Cyber Threats: Issues, Approaches, and Challenges*, pages 247–266, 2005.
- [15] K. Kaynar. A taxonomy for attack graph generation and usage in network security. *J. of Information Security and Applications*, 29:27–56, 2016.
- [16] K. Kaynar and F. Sivrikaya. Distributed attack graph generation. *IEEE Trans. on Dependable and Secure Computing*, 13(5):519–532, 2016.
- [17] H. Klaudel and W. Klaudel. Modelling and analysing cyber-resilience of an autonomous vehicle with zonal architecture with score+ (long version). (hal-05669933), 2024.
- [18] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer. Attack–defence trees. *Journal of Logic and Computation*, 24(1):55–87, 06 2012.
- [19] R. Kumar, S. Schivo, E. Ruijters, B. Mehmet Yildiz, D. Huistra, J. Brandt, A. Rensink, and M. Stoelinga. Effective analysis of attack trees: A model-driven approach. In *FASE 2018*, pages 56–73. Springer International Publishing, 2018.
- [20] H. Singh Lallie, K. Debattista, and J. Bal. A review of attack graph and attack tree visual syntax in cyber security. *Computer Science Review*, 35:100219, 2020.
- [21] H. Li, Y. Wang, and Y. Cao. Searching forward complete attack graph generation algorithm based on hypergraph partitioning. *Procedia Computer Science*, 107:27–38, 2017.
- [22] S. Mauw and M. Oostdijk. Foundations of attack trees. In *Information Security and Cryptology-ICISC 2005*, pages 186–198. Springer, 2006.
- [23] MITRE. Common weakness enumeration, 2023. <https://cwe.mitre.org/data/index.html>.
- [24] R.W. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *IEEE Symposium on Security and Privacy*, pages 156–165, 2000.
- [25] M. Salfer, H. Schweppe, and C. Eckert. Efficient attack forest construction for automotive on-board networks. In *Information Security*, pages 442–453, Cham, 2014. Springer International Publishing.
- [26] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and Privacy*, pages 273–284, 2002.