Manufacturing Science and Education 2025

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 68, Issue Special III, August, 2025

APPLICATION OF FUZZ TESTING FOR FUNCTIONAL VALIDATION IN AUTOMOTIVE

Aurelian POPESCU, Claudiu Vasile KIFOR

Abstract: *Over the past 20 years, fuzz testing has been rarely utilized in the automotive industry. However, with the introduction of ISO/SAE 21434, its adoption has significantly increased, becoming a standard practice in most projects to mitigate cybersecurity risks. This paper provides a brief overview of fuzz testing in automotive and shows how its use can be extended beyond security testing to support functional validation. Two use cases where fuzz testing can be effectively applied: "Boundary Value Analysis" and "State Transition Testing" are presented.*
Keywords: *fuzz testing, fuzzing, automotive, functional testing, ISO/SAE 21434, validation*

1. INTRODUCTION

Fuzz testing is an effective testing technique for identifying bus and vulnerabilities, that is becoming increasingly important in the automotive industry with the emergence of cybersecurity guidelines and standards such as J3061 [1], ISO/SAE 21434[2], UNECE R155[3]. In this paper we will explore the use of fuzz testing not only for mitigating cybersecurity risks but also for detecting functional and safety issues at the component, system, and vehicle levels.

Fuzzing is a straightforward method for automated software testing, where random data is introduced into a program or communication channel. The primary goal of fuzz testing is to push the System Under Test (SUT) into a crash or abnormal state, helping to identify potential security vulnerabilities and system instabilities.

In the IT domain, fuzzing is commonly applied at both the unit testing and integration testing levels.

The concept of fuzz testing was first introduced in 1988 by Professor Barton Miller during robustness testing of Unix programs. Over time, continuous advancements in fuzzing have led to the development of sophisticated tools and automated frameworks, making it widely adopted across various domains. Fuzzing gained significant popularity with the emergence of AFL (American Fuzzy Loop) between 2014 and 2017, created by Michal Zalewski [4].

Fuzz testing can detect software issues early in the development process and can be designed to use a wide range of inputs, including boundary values that traditional testing methods may overlook. Another testing method, Requirements-based testing, focuses on a limited set of inputs, sometimes creating a false sense of high coverage. In contrast, fuzz testing can generate and execute a vast number of test cases automatically. Ideally, it can be seamlessly integrated into the Continuous Integration/Continuous Deployment (CI/CD) pipeline [5].

2. LITERATURE REVIEW

ISO/SAE 21434. Road vehicles — Cybersecurity engineering, [2] recommends fuzz testing and penetration testing as separate methods during the integration and verification phase of projects. However, in some cases, fuzzing is incorporated as part of the penetration testing process. Anistoroaei et al. [6] demonstrate how fuzzing can be applied to scan the CAN (Controller Area Network) channel to identify messages that influence the functional

behavior of in-vehicle instrument clusters. Once the relevant CAN messages are detected, attackers can simulate various vehicle malfunctions.

While fuzzing is highly effective, it also has certain drawbacks. One major challenge is the extended runtime required to achieve reliable results and high testing confidence. Security testers often need to manually analyze extensive fuzz logs, sometimes reaching several gigabytes. To minimize this manual effort, Golam et al. [7] developed a system that automatically classifies discovered vulnerabilities. This system generates severity scores for Unified Diagnostic Services (UDS) CAN bus fuzzing results, allowing security testers to prioritize the most critical findings.

Fowler et al. [8] propose a black-box fuzzing method for CAN cybersecurity testing. Their approach consists of seven steps, requiring significant time and resources: setting up a test environment, defining security tests, selecting the test target, developing or choosing testing tools (Make-or-Buy decision), validating the tools, performing experimental tests, and refining both the methodology and tooling.

Greybox fuzz testing is generally used when code coverage is available or the system architecture is known, making it effective for identifying new issues. Mutation testing (also called mutation analysis) serves as an alternative to code coverage by assessing how well test cases detect artificially injected faults in the System Under Test (SUT). Vikram et al. [9] introduce and evaluate Mu2, a Java-based tool that integrates mutation analysis into the greybox fuzzing loop to generate test input arrays with high mutation scores.

Lee et al. [10] propose another mutation testing approach that has proven effective for C and C++ software. In this method, fuzz testing generates random test inputs designed to explore different software branches and program states, increasing the likelihood of discovering vulnerabilities.

Kim et al. . [11] introduce a Structure-Aware CAN Fuzzing protocol, a greybox fuzzing method that identifies the structure of CAN messages in Phase 1 and then systematically generates fuzzing inputs in Phase 2 to detect vulnerabilities in the SUT (see Figure 1). This approach reduces testing time and accelerates issue discovery within CAN DBC (database CAN) files. However, the analyzed system does not include feedback fuzzing, a feature commonly found in advanced software fuzzing techniques. The final test results and reports are generated after the Monitoring phase (Phase 3) is completed.

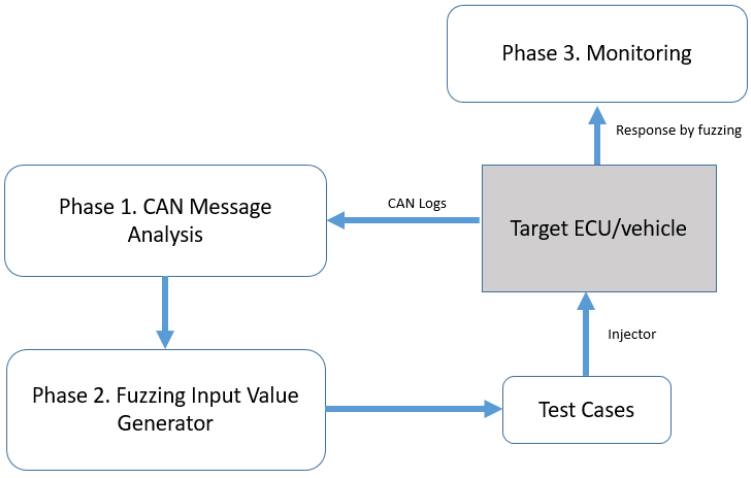

Fig. 1. Structure-aware CAN Fuzzing protocol [11]

Other greybox fuzzing tools for automotive applications, such as Vulfuzz and VulFuzz++ [10][11], have been developed and analyzed for improved efficiency [12-13]. These tools

combine fuzz testing capabilities with prioritization of untested software branches.

Celik et al. [14] compare black-box fuzzing tools for testing the Unified Diagnostic Services (UDS) implementation over CAN, specifically evaluating Caring Caribou [15], an open-source software, and Defensics [16], a commercial solution. Additionally, they analyze eight other open-source UDS implementations available on GitHub.

VITROBENCH [17] is a test framework designed for commercial off-the-shelf (COTS) ECUs, providing full packet control over CAN, CAN-FD, and LIN. Although primarily developed for cybersecurity testing—capable of simulating interception, frame injection, and sniffing attacks—VITROBENCH can also be utilized as a fuzz testing tool for functional testing.

Experiences with protocol-based fuzz testing have revealed two major drawbacks [18]. When CAN messages include a checksum signal, fuzzing often generates a high volume of invalid test cases, which receiving ECUs ignore, reducing testing efficiency. However, a machine learning-based fuzz testing approach trained on original CAN bus messages could enhance testing performance by improving message validity and effectiveness.

Oka et al. [19] suggest that fuzz testing should not be limited to system and acceptance testing, as recommended by ISO/SAE 21434, but should also be integrated into earlier stages of software development, such as Unit Testing and Integration Testing.

3. PROPOSED APPROACH

In this paper we try to show that fuzz testing method can be used for functional validation at integration and system test level. The method could be seen like a ”stress” testing. The usage of fuzzing tools can be extended also from cybersecurity test to functional and safety tests. Because the fuzz testing is an automated testing method, the functional validation have to be in this case semi or fully automated.

Fuzz testing can be conducted using black-box, grey-box, or white-box approaches. Black-box fuzz testing is primarily employed for security testing at the system or vehicle level, while grey-box fuzz testing is also applicable to cybersecurity assessments. We propose extending the grey-box approach to functional testing at the system level (using testbenches or Hardware-in-the-Loop setups) as well as at the vehicle level. White-box fuzz testing is most suitable for Unit Testing, particularly when the code or model (e.g., a Simulink model) is accessible.

Depending on the test level, different parameters can be subjected to fuzzing, including software/system parameters, internal ECU variables, external analog/digital ECU inputs, communication bus signals (LIN, CAN, CAN-FD, Ethernet, MOST, FlexRay), and diagnostic requests (primarily via the UDS protocol).

A typical fuzz testing framework consists of three key components (see Figure 2): the fuzz engine, which generates test inputs for SUT, the injector that ensures that the generated fuzz data is correctly delivered to the SUT in the appropriate format (e.g., CAN, LIN, Ethernet messages, or analog / digital signals), and the monitor that observes the SUT for any unexpected reactions or behaviors.

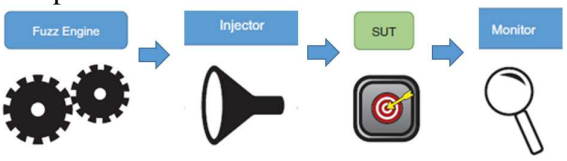

Fig. 2. Fuzz testing. (Openclipart. Creative Commons Zero 1.0 License)

Figure 3 illustrates a possible integration of a fuzz tester within a functional system test framework. The fuzz engine must be embedded within the test case generation environment, while the injector should be integrated into the HIL I/O interface. The fuzz monitor will operate alongside the HIL monitor component.

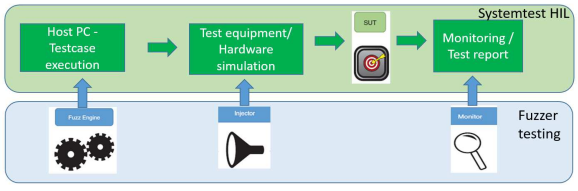

Fig. 3. Integration of Fuzz testing in Systemtest HIL

In most cases, the fuzz monitor requires additional signals or properties of the system under test (SUT) to be observed. In embedded systems, key characteristics monitored by the fuzz tester typically include power consumption, error codes, and timeouts.

For complex projects requiring high testing efficiency, we propose a structure-aware fuzzing system with runtime feedback. As shown in Figure 4, part of the test case generation process (including certain test parameters) occurs dynamically during test execution. If a slight modification of an input parameter triggers a noticeable reaction in the system under test (SUT), the fuzz engine will generate additional test cases focusing on the surrounding value range of that parameter.

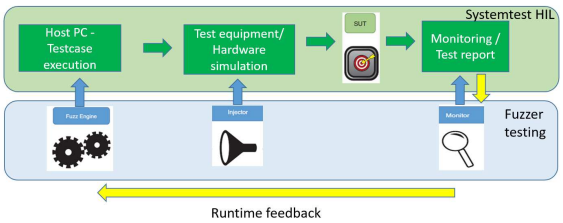

Fig. 4. Fuzz testing with feedback loop

Oka et al. [19] identified two key challenges in applying fuzz testing to the automotive domain.

First, during fuzzing, the SUT may not reach a stable and testable state. Some required inputs may be missing or invalid, causing the SUT to enter a failsafe mode and disregard the fuzzed inputs. Second, there may be limitations in monitoring capabilities, where certain unexpected behaviors of the SUT go undetected.

4. IMPLEMENTATION

We will introduce two theoretical use cases demonstrating the application of fuzz testing for specification-based testing [20].

The first use case, "Boundary Value Analysis with Hysteresis" highlights three potential functional requirements that can be tested in a HIL environment, while the second one explores a simplified "State Transition" scenario for the AUTOSAR CAN network management function [21].

4.1 Boundary value analysis with hysteresis

Consider the example of a Tire Pressure Monitoring System (TPMS) where the "under-pressure warning" function needs to be tested.

Implementation A: A function that relies on a single variable - pressure.

If the tire pressure drops below $P_{warningL1}$(80% of the reference pressure P_{ref}), an under-pressure warning is triggered. This warning remains active until the pressure rises above $P_{thresholdL1}$ (90% of P_{ref}). It is particularly interesting to analyze how the system responds when the pressure fluctuates between $P_{warningL1}$ and $P_{thresholdL1}$.

Implementation B: A function with two variables (pressure and time)

If the tire pressure remains below $P_{warningL1}$ (80% of the reference pressure P_{ref}) for more than $t_{warningL1}$ (6 minutes), an under-pressure warning is triggered. This warning stays active until the pressure exceeds $P_{thresholdL1}$ (90% of P_{ref}) for at least $t_{thresholdL1}$ (2 minutes).

Implementation C: Function with a Three-Variable System (Pressure, Time, and Temperature-Compensated Reference Pressure)

If the tire pressure drops below $P_{warningL1}$ (80% of the temperature-compensated reference pressure, P_{Tref}) for more than $t_{warningL1}$ (6 minutes), an underpressure warning is triggered. This warning remains active until the actual pressure exceeds $P_{thresholdL1}$ (90% of P_{Tref}) for more than $t_{thresholdL1}$ (2 minutes). The temperature-compensated reference pressure (P_{Tref}) varies linearly with a **k** coefficient based on the reference pressure.

For Implementation A, the fuzzer will generate “pressure” values within the range of ($P_{warningL1}$ – 1) Bar to Pref.

For Implementation B, the fuzzer will modify both pressure and the time duration of pressure decrease/increase (i.e., the rate of inflation or deflation).

For Implementation C, in addition to the modifications in Implementation B, the fuzzer will also include tire temperature as an input. In this case, the fuzzer settings must be carefully selected to prevent unrealistic scenarios, such as an increase in temperature while tire pressure decreases.

Fuzz testing for all three requirements can be extended to include scenarios where the TPMS

temporarily loses communication with tire sensors (radio signal disruption) or receives invalid sensor values for a few seconds.

To enhance testing quality, a mutation/analysis-guided fuzzing loop can be integrated into the fuzzer. The effectiveness of this approach was evaluated using a Java framework [9] and, with some modifications, can also be applied in the automotive domain.

4.2 State Transition Testing

In order to explore how a fuzzer can enhance state transition testing**,** a simplified CAN Network Management (NM) state transition diagram (see Figure 5)**,** derived from the AUTOSAR specification was used [21]. Depending on the traffic on the CAN Bus**,** the ECU transitions between different NM states**.** The fuzzer will inject dummy CAN messages at random cycle times to evaluate the system's response.

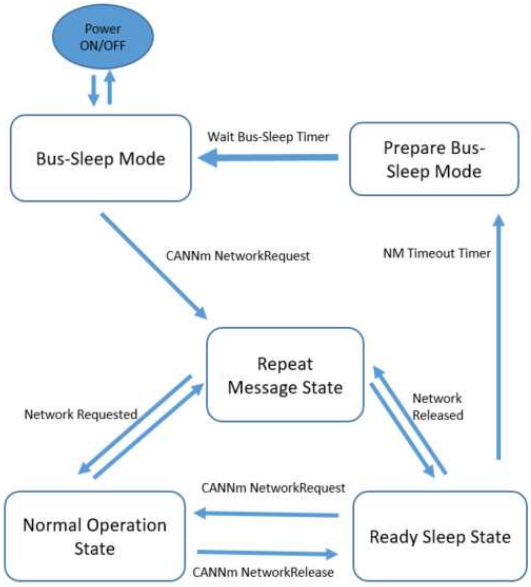

Fig. 5. CAN Network Management (CanNM) – AUTOSAR BSW [21]

CAPL code:

```
msTimer  DummyCANmessage_mT;
on timer DummyCANmessage_mT {
    Output(DummyCANmessage);//        put
message on CAN
    Time_random                        =
(Time_expired+WaitBusSleepTime)*random(0;
1.1)
    setTimer(time_random)
    }
```

For functional testing, fuzzing the Network Management (NM) implementation of an ECU can be particularly useful when the ECU operates on two or more CAN networks, where certain messages need to be gated between them. Since each ECU has a unique overrun time, which can also be influenced by active functions before the Ignition_OFF event, covering all possible scenarios with predefined test cases can be challenging for a tester.

By implementing fuzz testing, previously untested scenarios can be executed, increasing test coverage. Additionally, integrating fuzz testing into regression testing provides an opportunity to uncover new bugs over time.

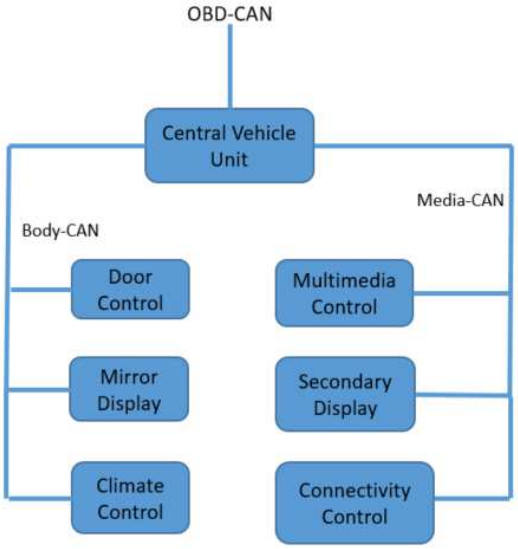

Fig. 6. Vehicle Subsystem CAN architecture example

Figure 6 illustrates a vehicle's subsystem CAN architecture**,** where all ECUs in the network have Network Management (NM) functionality implemented. To assess the stability of NM at the system level, a fuzzer can be configured to trigger an "OpenDoor" event at varying time intervals by manipulating a digital DoorContact signal. During a complete sleep state of the subsystem CANs, opening the vehicle door will wake up the DoorControl ECU**,** which in turn activates the Body-CAN and Media-CAN to notify other ECUs of the door state change. As a result:

- The MirrorDisplay ECU turns on to provide the driver with a vehicle view.
- The Secondary Display activates to show a pictogram with all door states.

In this process, the DoorControl ECU transitions from NM_BusSleepMode to NM_NormalOperation (NM_NO) state, briefly passing through RepeatMessage**.** In NM_NO**,** it transmits CAN messages on the Body-CAN**,** causing all ECUs to enter the NM_NO state. Simultaneously, the CentralVehicleUnit forwards the DoorState message to the Media-CAN**,** ensuring that the Secondary Display wakes up to show the door status or, upon request, different camera views.

Once the door contact is closed**,** and after a predefined timeout, the DoorControl ECU no longer needs to maintain CAN communication and transitions through NM_ReadySleep → NM_PrepareBusSleep → NM_BusSleepMode. The other ECUs follow the same NM sequence.

Scenario to Consider: What happens to CAN communication if the driver activates the Cabin Ventilation function (from the Climate Control ECU) or the Radio function (from the Multimedia Control ECU) while the DoorControl ECU is in NM_ReadySleep or NM_PrepareBusSleep Mode**?**

5. RESULTS

We present two functional testing use cases where the fuzzing method can enhance test depth.

5.1. Boundary Value Analysis with Hysteresis

-Validating Function A is straightforward and does not require fuzz testing.

-However, for Function B and Function C, the number of possible input combinations increases significantly. Using fuzz testing for these functions helps reduce the risk of software implementation bugs.

5.2. State Transition Testing

-Applying fuzz testing to validate NM implementation in a single ECU may not justify the effort.

-However, when testing a System Under Test (SUT) with multiple ECUs on the same CAN bus, the number of possible test cases increases exponentially. In such cases, fuzz testing enhances confidence in the SUT's implementation. If unexpected behavior is detected, the fuzzer engine saves the input that triggered the issue, creating a reproducible test case for regression testing.

6. CONCLUSION

In these two use cases, we demonstrate how the fuzz testing method can be applied to identify functional bugs. The primary objective of this testing approach is not to detect vulnerabilities but to ensure that the system behaves as expected under normal operating conditions.

During functional testing, testers are aware of the expected ECU states, and fuzzing from these states increases the likelihood of reaching the relevant application code sections. For example, certain system functions are only activated when the engine is running, and this status is communicated via CAN signals. If, during fuzz testing, the engine speed signal is unavailable or invalid, the application software will ignore all related inputs.

When using fuzz testing for functional validation, determining the necessary duration of testing for a system or software function can be challenging. The extent of fuzzing depends on project resources and risk assessment, making it a key area for future research.

Fuzz testing continues to evolve, particularly in terms of testing efficiency [22]. As its application expands to functional and safety testing, the maturity of this method in the automotive industry will also accelerate.

7. REFERENCES

[1] SAE, J3061 - *Cybersecurity Guidebook for Cyber-Physical Vehicle Systems*, SAE Int. J. Connect. Autom. Veh. (2016). https://www.sae.org/standards/content/j3061_201601/. 2016

[2] ISO, *ISO/SAE 21434:2021 Road vehicles — Cybersecurity engineering*, ISO/TC 22/SC 32 Electr. Electron. Components Gen. Syst. Asp. (2021).

[3] UN-ECE, R155 - *Cyber security and cyber security management system*, Off. J. Eur.

Union Eur. Union. (2021). https://doi.org/ISSN 1977-0642. 2021

[4] Zalewski, M, *American fuzzy lop*, (n.d.). https://lcamtuf.coredump.cx/afl/. accessed in 2025

[5] Bhavani, R, *What Is Fuzz Testing, And How Does It Work?*, (n.d.). https://www.qatouch.com/blog/fuzz-testing/. accessed in 2025

[6] Anistoroaei, A., Groza, B., Murvay, P.-S., Gurban, H. *Security Analysis of Vehicle Instrument Clusters by Automatic Fuzzing and Image Acquisition*, in: Proc. 2022 IEEE Int. Conf. Autom. Qual. TESTING, Robot. (AQTR 2022), IEEE, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2022: pp. 13–18. https://doi.org/10.1109/AQTR55203.2022.9802024. 2022

[7] Golam, G., Kayas, Z., Pelletier; D., *AI-assisted Vulnerability Analysis And Classification Framework for UDS on CAN-bus Fuzzer*, in: 10th escar USA - The World's Leading Automotive Cyber Security Conference, 2023

[8] Fowler, D.S., Bryans, J., Cheah, M., Wooderson, P., Shaikh, S.A., *A Method for Constructing Automotive Cybersecurity Tests, a CAN Fuzz Testing Example*, in: 2019 COMPANION 19TH IEEE Int. Conf. Softw. Qual. Reliab. Secur. (QRS-C 2019), IEEE COMPUTER SOC, 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA, 2019:

[9] Vikram, V., Laybourn, I., Li, A., Nair, N., Brien, K.O., Sanna, R., Padhye, R. *Guiding Greybox Fuzzing with Mutation Testing*, Proc. 32ND ACM SIGSOFT Int. Symp. Softw. Test. Anal. ISSTA 2023. 929–941. (2023)

[10] Lee, J., Viganò, E., Cornejo, O., Pastore, F., Briand, L. IEEE, *Fuzzing for CPS Mutation Testing*, 2023 38TH IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE. 1377–1389. (2023)

[11] Kim, H., Jeong, Y., Choi, W., Lee, D.H., Jo, H.J. *Efficient ECU Analysis Technology Through Structure-Aware CAN Fuzzing*, IEEE Access. 10 (2022).

[12] Moukahal, L.J., Zulkernine, M., Soukup, I.C.. Soc, *Boosting Grey-box Fuzzing for Connected Autonomous Vehicle Systems*, 2021 21ST Int. Conf. Softw. Qual. Reliab. Secur. COMPANION (QRS-C 2021). 516–527. (2021)

[13] Moukahal, L.J., Zulkernine, M., Soukup, I.C. *Vulnerability-Oriented Fuzz Testing for Connected Autonomous Vehicle Systems*, IEEE Trans. Reliab. (2021).

[14] Celik, L., McShane, J., Scott, C., Aideyan, I., Brooks, R., Pesé, M., *Comparing Open-Source UDS Implementations Through Fuzz Testing*, 2024. https://doi.org/10.4271/2024-01-2799.

[15] Vinnova, *Caring Caribou*, (n.d.). https://github.com/CaringCaribou/caringcaribou. accessed in 2025

[16] Blackduck, *Defensics Fuzz Testing Tool*, (n.d.). https://www.blackduck.com/fuzz-testing.html. accessed in 2025

[17] Yeo, A.K.T. Garbelini, M.E., . Chattopadhyay, S., Zhou, J. *VITROBENCH: Manipulating in-vehicle networks and COTS ECUs on your bench A comprehensive test platform for automotive cybersecurity research*, Veh. Commun. 43 (2023).

[18] Li, Z., Jiang, W., Liu, X., Tan, K., Jin, X., Yang M., *GAN model using field fuzz mutation for in-vehicle CAN bus intrusion detection*, Math. Biosci. Eng. 19 6996–7018. https://doi.org/10.3934/mbe.2022330. 2022

[19] D. Kengo Oka, *Building Secure Cars*,. https://doi.org/10.1002/9781119710783. 2021

[20] ISTQB, *International Software Testing Qualification Board*, (2002). https://www.istqb.org. - accessed in 2025

[21] AUTOSAR, *Specification of CAN Network Management R23-11*, AUTOSAR. (n.d.) 103. https://www.autosar.org/fileadmin/standards/R23-11/CP/AUTOSAR_CP_SWS_CANNetworkManagement.pdf. accessed in 2025

[22] Kifor, C., Popescu, A. *Automotive Cybersecurity: A Survey on Frameworks, Standards, and Testing and Monitoring Technologies,* Sensors. 24. https://doi.org/10.*3390*/s24186139. (2024)

Aplicarea Fuzz Testing pentru validarea funcțională în industria auto

În ultimii 20 de ani, fuzz testing a fost rar utilizată în industria auto. Cu toate acestea, odată cu introducerea ISO/SAE 21434, adoptarea acestuia a crescut semnificativ, devenind o practică standard în majoritatea proiectelor din automotive, atenuând riscurile de securitate cibernetică. Această lucrare oferă o scurtă prezentare generală a testării fuzzy în automotive și arată modul în care utilizarea acestuia poate fi extinsă dincolo de testarea pentru securitate pentru a sprijini validarea funcțională. Sunt prezentate două cazuri de utilizare în care testarea fuzzy poate fi aplicată eficient: „ Boundary Value Analysis" și „State Transition Testing".

Aurelian POPESCU, Ing, PhD Candidate, Lucian Blaga University of Sibiu, Faculty of Engineering, Department of Industrial Engineering and Management, aurelian.popescu@ulbsibiu.ro
Claudiu V. KIFOR, Prof. PhD., Lucian Blaga University of Sibiu, Faculty of Engineering, Department of Industrial Engineering and Management, claudiu.kifor@ulbsibiu.ro