GFDG: A Genetic Fuzzing Method for the Controller Area Network Protocol

Miguel Stey *, Murad Hachani †, Philipp Fuxen †, Julian Graf †, Rudolf Hackenberg †
Department of Computer Science and Mathematics
Ostbayerische Technische Hochschule Regensburg
Regensburg, Deutschland
* e-mail: miguel1.stey@st.oth-regensburg.de,
† e-mail: {murad.hachani | philipp.fuxen | julian.graf | rudolf.hackenberg}@oth-regensburg.de

***Abstract*—Ensuring the security of modern automotive systems is critical due to their increasing complexity and reliance on interconnected Electronic Control Units. The Controller Area Network still serves as a key communication protocol within these systems, making it a primary target for security testing. Traditional fuzz testing approaches for Controller Area Networks often rely on random or brute-force message generation, not leveraging the system's feedback to improve the generation process. This paper introduces the Genetic Fuzz Data Generator, a fuzzing method that leverages Genetic Algorithms and side-channel analysis to enhance Controller Area Network security testing. The Genetic Fuzz Data Generator dynamically refines its fuzzing strategy by evaluating system responses through side-channel data, such as processing unit temperatures and power supply variations. By structuring Controller Area Network messages as genetic individuals and applying evolutionary principles—including selection, crossover, and mutation—the Genetic Fuzz Data Generator systematically identifies active Controller Area Network IDs and generates targeted fuzz messages. Experimental validation was conducted on a real automotive electronic control unit within a controlled laboratory setup. The first results demonstrated the approach's effectiveness, revealing system anomalies, including a Denial of Service vulnerability that disrupted functions of the investigated Electronic Control Unit. The findings highlight the potential of feedback-driven fuzzing for improving the efficiency of black-box security testing in Controller Area Network-based systems. Future research could further optimize fitness functions or explore additional side-channel metrics.**

***Keywords*-Automotive Security; Controller Area Network; Fuzz Testing; Genetic Algorithm; Side-Channel Analysis.**

I. Introduction

Modern connected vehicle systems rely on increasingly complex software running on Electronic Control Units (ECUs) that manage critical functions. Ensuring the security of these systems is essential, particularly as the attack surface expands with enhanced connectivity, integration of multiple networked components, and the growing reliance on cloud-based infrastructures for remote diagnostics, software updates, and real-time data processing. Building upon our previous work [1], where we developed a side-channel monitoring setup for fuzz testing automotive systems, we have identified key limitations in traditional random fuzzing approaches—specifically, the challenge of efficiently detecting active Controller Area Network (CAN) IDs. This insight motivated the development of the Genetic Fuzz Data Generator (GFDG), a method that leverages genetic algorithms and side-channel feedback to systematically generate targeted fuzz messages for the CAN protocol.

In conventional fuzz testing, generating a large volume of random messages often results in a low probability of triggering a response from the target system. Our prior research demonstrated that side-channel data could significantly enhance anomaly detection; however, the approach lacked the adaptive capability to focus on active CAN IDs. The GFDG addresses this gap by structuring CAN messages as genetic individuals—each represented by an 11-bit identifier and a payload—and refining them through evolutionary operations, such as selection, crossover, mutation, and migration. Preliminary results suggest that this feedback-driven process can enhance testing efficiency and contribute to identifying subtle vulnerabilities, though further investigation is needed to quantify its full impact.

Given that this paper focuses on the innovative integration of genetic algorithms with side-channel analysis to dynamically target and refine CAN fuzzing, we frame our investigation around the following research questions:

- **RQ1:** How does the integration of genetic algorithms with side-channel feedback improve the identification of active CAN IDs?
- **RQ2:** What are the impacts of evolutionary operations (selection, crossover, mutation, and migration) on the performance and adaptability of the fuzzing process?

This paper is organized as follows. Section II reviews related work, including an analysis of existing fuzzing methodologies and the limitations observed in our 2023 study. Section III describes the underlying concepts and the theoretical foundation of genetic algorithms in the context of fuzz testing. Section IV details the architecture and implementation of the GFDG. Section V presents experimental evaluations conducted on a real automotive ECU, and Section VI discusses the results, highlighting both improvements and remaining challenges. Finally, Section VII concludes with directions for future research.

II. Related work

Fuzz testing has emerged as a critical technique for identifying vulnerabilities in embedded systems, where conventional random-input approaches often fall short due to limited I/O capabilities, constrained resources, and heterogeneous architectures [2]. These inherent challenges have motivated the development of feedback-driven methodologies that are specifically tailored for embedded environments.

A notable advancement in this area is demonstrated by the Firm-AFL framework, which adapts coverage-guided fuzzing techniques to the constraints of embedded firmware. Firm-AFL

shows that by integrating runtime feedback into the fuzzing loop, one can significantly enhance vulnerability detection even in resource-limited settings. This insight underlines the necessity of adapting traditional fuzzing techniques to the particularities of embedded systems [3].

In parallel, side-channel-assisted fuzzing has recently emerged as a promising approach. Sperl and Böttinger propose a method that leverages power consumption measurements as a feedback mechanism, inferring aspects of the target's control flow from power traces. Such side-channel feedback can mitigate the "black-box" limitations inherent in conventional fuzz testing of embedded devices [4].

In the automotive domain, securing the CAN is of paramount importance given its central role in vehicle communications. In "Fuzzing CAN Packets into Automobiles" [5], Lee et al. demonstrate that automotive systems are vulnerable even when attackers inject fuzzed CAN packets without in-depth system knowledge. Their experiments, which involve sniffing CAN traffic and subsequently fuzzing packet fields via wireless channels, reveal that random fuzzing can induce abnormal vehicle behavior. These findings highlight the inherent insecurity of the CAN bus and motivate the need for more systematic, feedback-guided fuzzing strategies in automotive networks.

Building on these insights, our work—the GFDG—applies the Genetic Algorithms (GAs) and side-channel analysis to enhance fuzzing techniques for CAN-based systems. By representing CAN messages as genetic individuals and refining test inputs through evolutionary operations (selection, crossover, and mutation), GFDG leverages runtime and side-channel feedback to focus fuzzing efforts on active CAN IDs. This hybrid approach not only echoes the advantages demonstrated by FIRM-AFL in adapting fuzzing to embedded firmware but also extends the paradigm by integrating the side-channel feedback techniques proposed by Lee et al. [5] and the empirical findings from "Fuzzing CAN Packets into Automobiles".

III. Background

This section provides an overview of key concepts relevant to this work. The CAN protocol is widely used for ECU communication in automotive systems, making it a critical target for security testing. Fuzz testing helps uncover vulnerabilities by generating test inputs and analyzing system responses, but traditional methods struggle with identifying active CAN IDs. GAs offer a potential solution by optimizing test case generation through system feedback.

A. CAN protocol

CAN is a broadcast-based protocol [6] used extensively in the automotive sector to connect ECUs. In the CAN protocol, different types of frames are defined, each serving a specific purpose. The Standard Frame, for example, is used for the regular exchange of messages between ECUs [7]. Typical applications of this frame include transmitting sensor data or sending commands to other ECUs. The Standard Frame consists of several fields that facilitate the transmission process, including the acknowledgment field and the checksum field [7]. For fuzz testing purposes, the most relevant component of the frame is the message it contains. A CAN message is defined by its identifier (ID) and a data field of up to 8 bytes [6]. The primary function of the CAN ID is to define the context of the data, while its secondary role involves enabling message filtering by the nodes on the bus. Since each node broadcasts messages onto the bus, the CAN ID allows each node to determine whether the message is intended for it or should be ignored [8]. In the context of fuzz testing, this means that an ECU will process only those fuzz messages with IDs that it is configured to recognize.

B. Fuzz Testing

Fuzz testing is an automated method for testing the security of information systems. It involves automatically generating input data for the system under test and monitoring its responses [9]. A fuzzer typically consists of two core components: a Fuzz Data Generator (FDG), responsible for producing new fuzz messages, and an Anomaly Monitor, which analyzes the system's reactions to identify potential vulnerabilities [6]. In the context of fuzz testing via CAN, the primary challenges are the large space of possible messages and the fact that an ECU only responds to a subset of CAN IDs. In a black-box fuzzing scenario, the specific active IDs of an ECU are unknown to the tester. Therefore, methods capable of identifying these active IDs are more efficient than non-feedback-based fuzzing approaches. The literature on CAN fuzz testing presents various methods, but most rely on randomly generating messages [10][11] or employing brute-force techniques [12][13][14]. These approaches do not utilize system feedback to identify active IDs, resulting in less efficient fuzzing processes. Consequently, there is a need for more advanced methods that incorporate system feedback to enhance the efficiency of CAN fuzz testing.

C. Genetic Algorithm

The GA is a search algorithm that mimics the principle of evolution from biology to find optimal solutions to problems. GAs can be considered a family of algorithms that utilize the same foundational structure but differ in specific strategies or parameters [15]. To solve a given problem, the first step involves defining an individual, which represents a potential solution. Each individual consists of certain genes that encode potential solutions for the problem. A group of individuals forms a population, and each iteration of the algorithm corresponds to a generation of individuals [15]. The GA begins by generating an initial population that represents Generation 0. The algorithm then iteratively proceeds through several steps for each generation. The core step is the evaluation of individuals based on their fitness, which is a numerical value indicating how well an individual solves the problem. Analogous to survival probability in natural evolution, the fitness score determines the likelihood of an individual contributing to the next generation of solutions [15]. After evaluating the fitness of all individuals, a selection process occurs, where certain individuals are chosen based on their fitness to participate in crossover. Crossover is

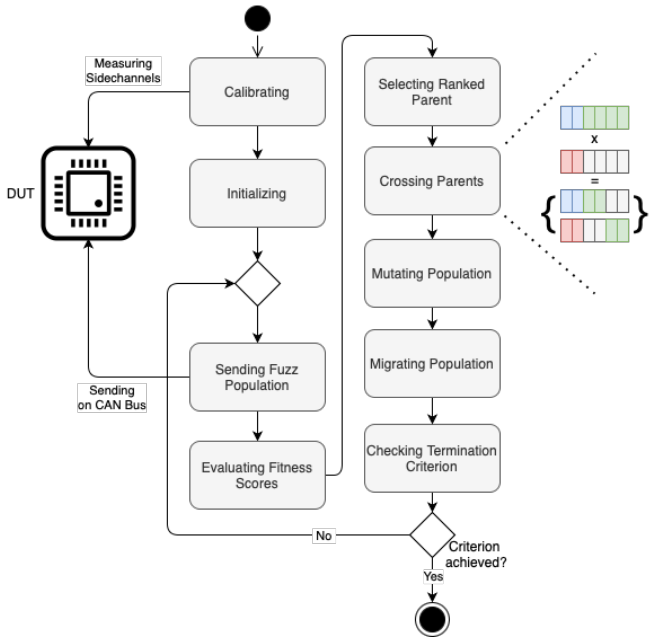

Figure 1. The Process of GFDG.

a genetic operation that combines pairs of selected individuals to generate new offspring, effectively recombining genes to explore new solution spaces [16]. Additionally, during the creation of offspring, mutations can occur. The mutation is typically implemented through bit-flips of genes based on a predefined mutation chance. After generating a new population, the algorithm repeats the cycle of evaluation, selection, crossover, and mutation until a termination criterion is met.

IV. The Genetic Fuzz Data Generator

This section describes the concept for the GFDG that is based on the structure of the GA and uses side-channel information for the evaluation. For this method, additional changes to the GA were taken into account for special requirements of the application to CAN fuzzing. For the GA, it is required to define an individual for the specific problem that the algorithm is applied to. The definition of a CAN Individual in the context of fuzz testing is provided in the following Subsection IV-A. In Subsection IV-B, the developed algorithm for the GFDG is described, including the reasoning in chosen strategies for each operation and all alterations to the standard GA structure. The modified algorithm for the CAN protocol is illustrated in Figure 1 and is further elaborated upon in the subsequent sections.

A. Defining a CAN-Individual

In the GA, each individual in the population represents a potential solution composed of various genes. In the context of fuzzing over the CAN protocol, an individual corresponds to a CAN message. This message must be recognized as a valid CAN message by the receiver of the target ECU. If the message violates protocol rules, the receiver node detects this, resulting in an error frame being sent back to the fuzzer without further processing of the message. Consequently, not all fields of the CAN frame are suitable as mutable genes for the individual. Therefore, the GFDG uses only the message portion of the CAN frame as the genes of an individual. From a broader perspective, an individual in the GFDG is a CAN message consisting of an 11-bit ID and a 64-bit data field. Other fields, such as data length and checksum, are recalculated correctly for transmission and are neither mutated nor inherited. The GA further conceptualizes individuals as collections of chromosomes composed of genes. Here, the CAN ID is treated as a chromosome consisting of 11 genes, each representing one bit of the CAN ID. This chromosome is mutation-resistant and cannot be split during crossover because the CAN ID determines whether the control unit processes the message. One goal of the GFDG is to identify accepting CAN IDs and test different payload data for these IDs. Therefore, IDs are not mutated randomly. Instead, the algorithm strategically generates more or fewer individuals with specific IDs. The second chromosome of a CAN individual represents the payload data, which can contain 0 to 64 bits. Since the payload's interpretation depends on the CAN ID and therefore is not predictable in a black-box scenario, each bit of the payload is treated as an individual gene. In summary, from the perspective of the GA, an individual in the GFDG consists of a non-mutable and non-crossable chromosome representing an 11-bit number and a standard chromosome with up to 64 genes.

B. GFDG Algorithm

The GFDG applies GAs to refine fuzz testing for CAN-based systems. This section details the algorithm's key steps, including initialization, message evaluation, selection, crossover, mutation, and migration. By leveraging system feedback, the GFDG aims to optimize test case generation and improve fuzzing efficiency.

1) Initialization: At the start of the fuzzing process, an initial generation must be created. This initial population is particularly important for the GFDG algorithm as it largely determines the IDs of the first generations. Since IDs do not change when creating new generations and only a few new IDs are introduced through migration, the initial population significantly impacts the performance of a run. In many cases, the standard strategy for GA is to initialize the first generation randomly. However, in this application, random initialization has a low probability of generating IDs in the first generation that trigger a response from the system. If no active IDs are generated, the run's performance depends on how many active IDs are introduced through migration, which constitutes only a small portion of each generation. To address this limitation, a different approach was developed. This method leverages side-channel feedback through the fitness function to actively select the initial population. In the first step, a multitude of CAN messages to the required amount are randomly generated and sent as fuzzing inputs to the target system. As in the normal

fuzzing process of GFDG, side-channel data is collected, and a fitness value is calculated for each message. The messages with the highest fitness values are then selected to form the first generation. By prioritizing messages with high fitness scores, this method aims to increase the likelihood of including active CAN IDs in the initial population, which can influence the performance of the GFDG algorithm.

2) Sending and Measuring: After each generation, the main steps of the fuzzing process are performed for each individual. First, the individual is sent to the system under test. Then, the side-channel data of the ECU is measured to evaluate the individual. In the test environment used to evaluate this method, the available side-channels included the temperature of the ECU's three processing units—namely, the Central Processing Unit (CPU), Graphics Processing Unit (GPU), and Automotive Microcontroller (AMC)—as well as the power supply of the associated display of the tested Infotainment System.

3) Evaluation: After an individual is sent to the system under test and the side-channel measurements are collected, these data are used to evaluate the individual. The evaluation is performed similarly to traditional GA, using a fitness function. In the context of optimization problems, the fitness value indicates how well an individual solves the problem. In this application of fuzz testing, the fitness function measures if and to what extent the system reacts to a sent fuzz message. It is important to note that the specific fitness function heavily depends on the system being tested, the chosen side-channels, and their behavior. Therefore, a single, fixed fitness function is not presented here. Instead, the process of identifying an appropriate fitness function for the investigated system is described. When applying this method to other CAN-based systems, it is necessary to analyze the available side-channels to determine which changes indicate a system reaction. This analysis informs the formulation of a suitable fitness function for the specific system.
This method was tested on an ECU controlling the infotainment system of a car. As previously mentioned, the selected side-channels in this setup were the temperatures of the ECU's processing units (CPU, GPU, and AMC) and the power supply of the connected display. The rationale for these side-channels is the assumption that if a fuzz message is processed by an otherwise isolated ECU, the processing units calculating should cause temperature increases. This is especially relevant when the infotainment system displays information, as any change on the display triggered by a fuzz message would require the GPU to render new images, leading to a measurable increase in its temperature. Similarly, the power supply of the display was monitored because potential system reactions, such as turning off the display or adjusting its brightness, would cause detectable changes in power, consumption. Therefore, the fitness function was designed to yield higher scores when the temperature of one or more processing units increased after sending a message or changes in power supply were measured.

4) Selection: After each individual of the current generation has been evaluated, parents for the next generation are selected using Ranked Selection. This method assigns each individual a fixed selection probability based on its rank within the generation according to its fitness score [17]. Ranked Selection was chosen because it ensures that the best-performing individuals have a significantly higher chance of being selected compared to others. This approach is based on the assumption that the fitness function accurately reflects the system's reactions. Consequently, GFDG can prioritize generating more fuzz messages from individuals that triggered the strongest reactions, increasing the likelihood of uncovering system vulnerabilities.

5) Crossover and Mutation: After the parent individuals for the next generation are selected, they are used to create the majority of the next generation through crossover and mutation. This process follows the standard GA approach, using single-point crossover for the payload genes of a message. The resulting offspring are then mutated using a bit-wise mutation strategy, where each bit of the payload is checked for mutation probability, and bit-flips are performed accordingly. As previously mentioned, the CAN ID is not affected by mutation, and its bits are not split by crossover. Instead, the CAN ID is linked to the first gene of the payload. Consequently, a child individual inherits the ID from the parent from which it received its first payload bit. Therefore, this design additionally ensures that the distribution of CAN IDs among the offspring mirrors the distribution present in the parent population.

6) Migration: As previously explained, altering the CAN IDs of parents is avoided because it would most likely result in CAN IDs that the ECU does not respond to. However, this creates a challenge for the GFDG approach, as it restricts each run to generating messages only with the IDs present in the initial generation. The likelihood of the first generation containing at least one active CAN ID is low, especially with small population sizes. This could lead to multiple runs of the GFDG without producing a single CAN message that triggers a response from the system under test. To overcome this limitation, the GFDG introduces an additional step to the standard GA to introduce new IDs into the population without losing the progress made in previous steps. This step is called Migration. During migration, a small portion of the next generation is created using new random individuals. This approach ensures a steady flow of new IDs and payload genes within a single run of the GFDG. Consequently, the algorithm can explore more CAN IDs in one run, increasing overall efficiency.

7) Termination Criteria: Every GA requires a termination criterion, which is checked after each generation. The common approach is to set a maximum number of iterations, which the GFDG also employs. Tests of the GFDG evaluated different iteration limits, identifying 50 to 100 generations as the optimal range. A lower limit prevents the algorithm from fully leveraging its evolutionary process, while a higher limit leads to a rapid increase in duplicate messages. Notably, this effect correlates with mutation probability—higher mutation rates can reduce the likelihood of duplicate messages within a single run, potentially allowing for a greater number of generations.

Algorithm 1 GFDG Algorithm

Input: POP_SIZE, MAX_ITER, CROSSOVER_RATE, MUTATION_RATE, MIGRATION_COUNT

$population \leftarrow initialize\_population()$
$generation \leftarrow 0$
while $generation < MAX\_ITER$ **do**
  for each $individual$ in $population$ **do**
    $individual.fitness \leftarrow send\_and\_evaluate(individual)$
  end for
  if $termination\_criteria\_met(population)$ **then**
    break
  end if
  $mating\_pool \leftarrow ranked\_selection(population)$
  $new\_population \leftarrow \{\}$
  while $|new\_population| < POP\_SIZE - MGRATION\_COUNT$ **do**
    $(parent1, parent2) \leftarrow pick\_two(mating\_pool)$
    $(child1, child2) \leftarrow crossover\_and\_mutate(parent1, parent2, CROSSOVER\_RATE, MUTATION\_RATE)$
    $new\_population \leftarrow new\_population \cup \{child1, child2\}$
  end while
  for $i \leftarrow 1$ **to** $MIGRATION\_COUNT$ **do**
    $new\_population \leftarrow new\_population \cup generate\_CAN\_message()$
  end for
  $population \leftarrow new\_population$
  $generation \leftarrow generation + 1$
end while

V. Experiments

The previously described GFDG algorithm was implemented and tested on an ECU in a laboratory setup to evaluate its performance. The tested system is an automotive ECU used in certain 2020 vehicle models to control various cockpit functions. Testing focused on the ECU's dedicated CAN channel for controlling the Infotainment System. Temperature sensors and an oscilloscope were used to monitor the previously mentioned side-channel signals. Before fuzz testing, multiple baseline measurements of the ECU were conducted to establish reference data representing its normal operating state. This baseline was then used for comparison with measurements taken during fuzz testing. For evaluation, multiple fuzz testing runs with the GFDG were performed under different ECU states, including while Navigation or Radio functions were active and while a mobile device was connected to the Infotainment System via Bluetooth. Each test run was performed with a population size of 16, a migration count of 4, and a maximum iteration count of 50. After each test run, the ECU was properly shut down, and a waiting period was observed to allow the processing units to cool. After each test, the GFDG log file—containing all sent CAN messages, associated fitness scores, and timestamps—was documented along with the side-channel measurements. Additionally, the display of the Infotainment System was manually monitored, and any visual effects were recorded.

VI. Results

After the experiments, the initial analysis compared side-channel measurements with and without fuzzing to identify anomalies. Without fuzz testing, the temperature of all three processing units remained stable within a 0.1 to 0.2°C range across all tests, and the display's power supply remained consistent. During fuzz testing with the GFDG, multiple anomalies were observed. These included temperature increases of 1 to 2°C in one or more processing units in a short period, which were reproducible by resending the same fuzz messages. Notably, temperature spikes were most pronounced when fuzz messages triggered visual changes on the display, confirming the hypothesis that rendering new images requires GPU processing, leading to increased temperature. Additionally, certain CAN messages generated by the GFDG caused a temporary decrease in display brightness, resulting in a measurable voltage drop of at least 50 percent. Some fuzz messages also led to short-term temperature spikes in the AMC. However, no visible changes in the user interface were observed for these cases, making it unclear what system behavior they triggered. Overall, multiple CAN messages were identified that caused observable changes in the system under test. Several messages with different CAN IDs triggered notifications on the display, including warning and error messages. Analyzing the GFDG log file confirmed that all fuzz messages with a measurable system impact were correctly identified by the fitness function. Furthermore, multiple child messages—generated from the same parent ID but with modified payloads—elicited distinct system reactions. A notable case involved a message generated through the migration step, which triggered a "Goodbye" message on the display. The GFDG detected this message as active due to a significant GPU temperature increase and selected it as a parent for further genetic operations. The resulting three child messages with the same ID exhibited different behaviors: two had no visible effect and were confirmed inactive through side-channel measurements, while the third caused the system to reboot the Bluetooth function and display an error message. Further testing revealed that sending this message twice in quick succession caused multiple infotainment functionalities to crash. Specifically, the Media, Radio, and Telephone menus became inaccessible through the user interface. Although the Bluetooth settings menu remained available, users could no longer modify any settings, turn Bluetooth on/off, or connect new devices to the vehicle's Infotainment System. This behavior was classified as a Denial of Service (DoS) vulnerability.

VII. Conclusion and Future Work

This research introduced the GFDG, a fuzzing approach for the CAN protocol that leverages GAs with the goal of enhancing testing efficiency. Unlike conventional CAN fuzzing

methods, the GFDG incorporates system feedback through side-channel analysis, enabling the identification of active CAN IDs and the generation of targeted fuzz messages. By structuring CAN messages as genetic individuals and applying evolutionary operations, such as selection, crossover, and mutation, the GFDG dynamically refines its test cases to increase the likelihood of triggering system responses and uncovering vulnerabilities. Experimental results suggest that this approach is capable of identifying system anomalies within an automotive ECU, though further analysis is needed to fully assess its effectiveness. Notable findings included measurable increases in processing unit temperatures and power supply variations, confirming the correlation between system reactions and observable changes in side-channels. Furthermore, the GFDG successfully uncovered a DoS vulnerability capable of disrupting multiple Infotainment functions, highlighting its potential for real-world security assessments. However, this method has certain limitations. The performance of the GFDG heavily depends on the accuracy of the defined fitness function, which, in turn, relies on the availability and quality of side-channel data for recognizing system reactions. Additionally, parameters, such as mutation probability, population size, and migration proportion influence its overall effectiveness. These aspects require further research to refine this method.

Despite these limitations, the experimental results indicate that integrating genetic algorithms with side-channel feedback can enhance the identification of active CAN IDs (RQ1) and can contribute to the efficiency and accuracy of vulnerability detection in automotive ECUs. Additionally, the application of evolutionary operations—selection, crossover, mutation, and migration—appeared to help refine fuzz messages, potentially enhancing the adaptability of the fuzzing process (RQ2), though further analysis is required to quantify this effect.

REFERENCES

[1] P. Fuxen, M. Hachani, J. Schmidt, P. Zaumseil, and R. Hackenberg, "Side channel monitoring for fuzz testing of future mobility systems", *CLOUD COMPUTING 2023*, p. 24, 2023.

[2] J. Yun, F. Rustamov, J. Kim, and Y. Shin, "Fuzzing of embedded systems: A survey", vol. 55, no. 7, 2022, ISSN: 0360-0300. DOI: 10.1145/3538644.

[3] Y. Zheng *et al.*, "FIRM-AFL: High-throughput greybox fuzzing of IoT firmware via augmented process emulation", presented at the 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 1099–1114, ISBN: 978-1-939133-06-9.

[4] P. Sperl and K. Böttinger, "Side-channel aware fuzzing", in *Computer Security–ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I 24*, Springer, 2019, pp. 259–278.

[5] H. Lee, K. Choi, K. Chung, J. Kim, and K. Yim, "Fuzzing can packets into automobiles", in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, IEEE, 2015, pp. 817–821.

[6] H. Zhang, K. Huang, J. Wang, and Z. Liu, "CAN-FT: A fuzz testing method for automotive controller area network bus", in *2021 International Conference on Computer Information Science and Artificial Intelligence (CISAI)*, Sep. 2021, pp. 225–231. DOI: 10.1109/CISAI54367.2021.00050.

[7] "Iso 11898-1:2024 road vehicles — controller area network", International Organization for Standardization, Standard, version 3, May 2024.

[8] F. Pölzlbauer, R. I. Davis, and I. Bate, "Analysis and optimization of message acceptance filter configurations for controller area network (CAN)", in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, ser. RTNS '17, New York, NY, USA: Association for Computing Machinery, Oct. 4, 2017, pp. 247–256, ISBN: 978-1-4503-5286-4. DOI: 10.1145/3139258.3139266.

[9] A. Singhal, T. Winograd, and K. A. Scarfone, "Guide to secure web services", National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-95, 2007, Edition: 0, NIST SP 800–95. DOI: 10.6028/NIST.SP.800-95.

[10] D. S. Fowler, J. Bryans, S. A. Shaikh, and P. Wooderson, "Fuzz testing for automotive cyber-security", in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, ISSN: 2325-6664, Jun. 2018, pp. 239–246. DOI: 10.1109/DSN-W.2018.00070.

[11] T. Werquin, R. Hubrechtsen, A. Thangarajan, F. Piessens, and J. T. Mühlberg, "Automated fuzzing of automotive control units", in *2019 International Workshop on Secure Internet of Things (SIOT)*, ISSN: 2690-8557, Sep. 2019, pp. 1–8. DOI: 10.1109/SIOT48044.2019.9637090.

[12] A. Anistoroaei, B. Groza, P.-Ş. Murvay, and H. Gurban, "Security analysis of vehicle instrument clusters by automatic fuzzing and image acquisition", in *2022 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, May 2022, pp. 1–6. DOI: 10.1109/AQTR55203.2022.9802024.

[13] D. S. Fowler, J. Bryans, M. Cheah, P. Wooderson, and S. A. Shaikh, "A method for constructing automotive cybersecurity tests, a CAN fuzz testing example", in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Jul. 2019, pp. 1–8. DOI: 10.1109/QRS-C.2019.00015.

[14] M. Li, Y. Wang, H. Zhang, and J. Wang, "PRFT: A fuzz testing method for tire pressure monitoring system based on protocol reverse", in *2023 2nd International Conference on Big Data, Information and Computer Network (BDICN)*, Jan. 2023, pp. 248–252. DOI: 10.1109/BDICN58493.2023.00058.

[15] S.-J. Wu and P.-T. Chow, "Steady-state genetic algorithms for discrete optimization of trusses", *Computers & Structures*, vol. 56, no. 6, pp. 979–991, Sep. 17, 1995, ISSN: 0045-7949. DOI: 10.1016/0045-7949(94)00551-D.

[16] A. Lambora, K. Gupta, and K. Chopra, "Genetic algorithm- a literature review", in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, Feb. 2019, pp. 380–384. DOI: 10.1109/COMITCon.2019.8862255.

[17] A. Shukla, H. M. Pandey, and D. Mehrotra, "Comparative review of selection techniques in genetic algorithm", in *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, Feb. 2015, pp. 515–519. DOI: 10.1109/ABLAZE.2015.7154916.