Automating fuzz test generation to improve the security of the Controller Area Network

Daniel S. Fowler, Jeremy Bryans, and Siraj Shaikh

Coventry University, Priory Street, Coventry, CV1 5FB, UK
{fowlerd3,jeremy.bryans,siraj.shaikh}@coventry.ac.uk

Abstract. Engineers design functional behaviour, but testing for cyber-security is difficult because it requires examining the system beyond the functional design. Fuzz testing (multiple calls to systems interfaces over a wide value space) has been successfully used to reveal vulnerabilities, yet it has seen little use in the automotive domain. An automated analysis of the in-vehicle network specification can be used to generate fuzz tests. These unexpected system inputs will reveal unconsidered operational cases to reveal security flaws prior to manufacture.

1 Motivation

Automotive cyber-security is important because hacking a vehicle is a safety issue [8]. Although engineers are primarily concerned with correctly implementing the operational requirements of a vehicle, they do so with safety in mind. The testing regime of a vehicle is rigorous and extensive, with the ISO 26262 standard addressing the safety critical electrical and electronics components. However, the advancement of technology and the advent of the connected and autonomous vehicle (CAV) have added extra Non-Functional Requirements (NFRs) [4]. The cyber-security NFR can affect safety. The automotive industry has begun to face this issue, publishing SAE J3061 and is in the process of updating ISO 26262 [10].

Yet the testing challenge remains due the complexity of the modern vehicle and the number and variety of possible cyber attacks. The modern vehicle is highly dependent upon networked computers, referred to as Electronic Control Units (ECUs). A typical mid-range 2017 executive car has 29 ECUs controlling a wide range of subsystems [2]. There are several protocols available to interconnect ECUs, the Controller Area Network (CAN) being the most popular (22 of the 29 ECUs in the example executive car have a CAN connection). CAN is usually exposed via the easily accessible On-board Diagnostics port (OBD). Some of the issues related to vehicle hacking revolve around manipulation of the messages flowing on the CAN data bus. A compromised ECU or man-in-the-middle (MITM) attack (for example an aftermarket device attached to a vehicle's OBD connector) can spoof the messages transmitted, affecting normal operation and threatening vehicle and passenger safety.

The in-vehicle network design (ECU nodes, ECU data interfaces and inter-ECU data flows) is defined in a network database file, known as the DBC file.

When an adversary injects messages on to the CAN bus it can violate this DBC design. The vehicle system must be resilient to such misuse. To ensure this resilience the CAN communications must be tested beyond what is defined in the DBC file.

2 Why fuzz testing?

How do tests get generated for CAN if the design does not contain adequate cyber-security information? Current designs focus on function, despite methods being available that allow attacks against systems to be described (such as: misuse cases; annotations; model stereotypes; and Aspect-oriented Modelling (AOM) [9]). However, such modelling methods must not detrimentally impact the existing engineering processes due to teams facing time and resource constraints.

It remains to be seen if the design process adapts to integrate cyber-attack considerations into the ECU designs. However, the DBC file can be used to improve vulnerability detection. Fuzz testing (multiple calls to systems interfaces over a wide value space) is considered a useful hacker's tool [6] to reveal system weaknesses. Yet fuzzing sees little use in the automotive industry [1]. The capturing of the in-vehicle network design into the DBC file presents an opportunity to automatically perform pre-production fuzz testing in the same environment as functional testing, using simulations, and hardware-in-the-loop (HIL) or software-in-the-loop (SIL) systems [5].

Several fuzzer packages exist that can be adapted to automotive use. The open source Peach Fuzzer is now commercialised and that company offers services to the automotive industry. However, studies of the effectiveness of automotive fuzzing prior to series production are required.

3 ECU Fuzz Testing

Authoring test cases to cover all permutations is an error prone task and typically fuzz tests are automatically generated. We propose that the existing design process (represented by the left part of Figure 1, blocks numbered 1 to 5) is expanded with an automated analysis of the DBC file to generate the fuzz tests (represented by the blocks numbered 6 and 7 in Figure 1). The fuzzing is intended to reveal system weaknesses in the System Under Test (SUT), or for a single ECU the Device Under Test (DUT). For the following process description the step number matches the block number in Figure 1.

1. Normal vehicle system design process where models and tools are used to design sensor inputs, ECUs, and the communication between them.
2. The outputs from the design process include:
   (a) Test cases for the known operational design.
   (b) The code that executes on the ECUs.

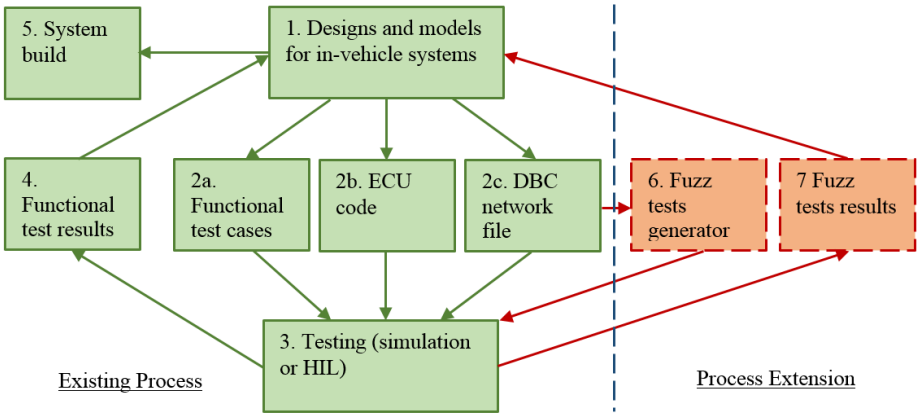

Fig. 1. Fuzzing tests from in-vehicle network specifications

(c) The DBC communications database file which defines the network properties.

3. Typical testing environment for designs (simulations or hardware-in-the-loop test rigs).
4. Test results feedback into the designs.
5. Once the design is signed-off it moves to manufacture.
6. Additionally, we propose that the DBC file analysis generates test cases to execute against the SUT or DUT. Going beyond the pure random tests of the black-box fuzz testing, the DBC file is a template for grey-box testing. The fuzzing test cases will be created by modifying slightly the CAN message components (id, data length, payload and message frequency), giving them values in the region space close to their definition. For example a payload may have a DBC signal definition for an engine temperature, e.g. `SG_ Comf_EngTemp : 16|8@1- (1,32) [-50|150]`. The fuzzer can vary the component elements (starting bit, bit length, endianness, sign, conversion factor and offset, and value), then monitor resultant system behaviour (one of the challenges of the research). This will lead, for example, to tests that explore unrealistic or unhandled component values, or rapid value changes in unrealistic time periods, to reveal vulnerabilities (caused by flaws in the code handling the defined communications). Those vulnerabilities may reveal unexpected data (confidentiality), affect system values (integrity) or prevent certain operations (availability).
7. The generated fuzzing test cases execute as a compromised ECU or a MITM illicit node on the system simulation or HIL rig. If the system has been designed correctly it will respond in a safe manner. This can include illegal states not being accepted, error codes registered, triggering of a Malfunction Indicator Lamp (MIL), or ECUs entering a limp home state. If an unexpected system response occurs then the result is used to inform the functional specifications or design models [3].

Fuzz testing has been successful in finding vulnerabilities in other domains [7]. With vehicles now connected cyber-physical systems automotive engineering needs to apply similar techniques to reduce vulnerabilities. The research is prototyping the methodology outlined above. Validation of the methodology can provided a useful tool extendable to other sectors, since CAN is used in industrial, medical and other transport domains.

References

[1] Harald Altinger, Franz Wotawa, and Markus Schurius. "Testing methods used in the automotive industry: results from a survey". In: *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing - JAMAICA 2014*. San Jose, California: ACM, 2014, pp. 1–6. DOI: 10.1145/2631890.2631891.

[2] BMW. *BMW Technical Training - F30 General Vehicle Electronics*. Tech. rep. Munich: BMW Group, 2012, p. 78.

[3] Madeline Cheah et al. "Combining Third Party Components Securely in Automotive Systems". In: *Information Security Theory and Practice: 10th IFIP WG 11.2 International Conference, WISTP 2016, Heraklion, Crete, Greece, September 26–27, 2016, Proceedings*. Ed. by Sara Foresti and Javier Lopez. Cham: Springer International Publishing, 2016, pp. 262–269. ISBN: 978-3-319-45931-8. DOI: 10.1007/978-3-319-45931-8_18.

[4] Riccardo Coppola, Maurizio Morisio, and Politecnico Torino. "Connected Car : Technologies , Issues , Future Trends". In: *ACM Computing Surveys* 49.3 (2016), pp. 1–36. ISSN: 15577341. DOI: 10.1145/2971482.

[5] D. S. Fowler et al. "Towards a Testbed for Automotive Cybersecurity". In: *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. Mar. 2017, pp. 540–541. DOI: 10.1109/ICST.2017.62.

[6] Craig Smith. *The Car Hacker's Handbook : A Guide for the Penetration Tester*. No Starch Press, 2016. ISBN: 9781593277031.

[7] Julian Thomé, Alessandra Gorla, and Andreas Zeller. "Search-based Security Testing of Web Applications". In: *Proceedings of the 7th International Workshop on Search-Based Software Testing*. SBST 2014. Hyderabad, India: ACM, 2014, pp. 5–14. ISBN: 978-1-4503-2852-4. DOI: 10.1145/2593833.2593835.

[8] Chris Valasek and Charlie Miller. "Remote Exploitation of an Unaltered Passenger Vehicle". In: *Black Hat USA* 2015 (2015), pp. 1–91.

[9] Armin Wasicek, Patricia Derler, and Edward A Lee. "Aspect-oriented Modeling of Attacks in Automotive Cyber-Physical Systems". In: *Proceedings of the 51st Annual Design Automation Conference*. DAC '14. New York, NY, USA: ACM, 2014, 21:1–21:6. ISBN: 978-1-4503-2730-5. DOI: 10.1145/2593069.2593095.

[10] Paul Wooderson and David Ward. "Cybersecurity Testing and Validation". In: *SAE Technical Paper*. SAE International, 2017. DOI: 10.4271/2017-01-1655.