2022-01-0117 Published 29 Mar 2022

Integrating Fuzz Testing into a CI Pipeline for Automotive Systems

Dennis Kengo Oka Synopsys

Nico Vinzenz ZF Friedrichshafen AG

Citation: Oka, D. and Vinzenz, N., "Integrating Fuzz Testing into a CI Pipeline for Automotive Systems," SAE Technical Paper 2022-01-0117, 2022, doi:10.4271/2022-01-0117.

Received: 21 Jan 2022 Revised: 21 Jan 2022 Accepted: 17 Jan 2022

Abstract

With the rapid development of connected and autonomous vehicles, more sophisticated automotive systems running large portions of software and implementing a variety of communication interfaces are being developed. The ever-expanding codebase increases the risk for software vulnerabilities, while at the same time the large number of communication interfaces make the systems more susceptible to be targeted by attackers. As such, it is of utmost importance for automotive organizations to identify potential vulnerabilities early and continuously in the development lifecycle in an automated manner. In this paper, we suggest a practical approach for integrating fuzz testing into a Continuous Integration (CI) pipeline for automotive systems. As a first step, we have performed a Threat Analysis and Risk Assessment (TARA) of a general E/E architecture to identify high-risk interfaces and functions. Next, we discuss the strategies for continuous fuzz testing and the technical requirements for integrating fuzz testing into a CI pipeline. Here it is imperative that organizations update their test strategies, covering how often to test, when to test, what to test, how to detect exceptions and how to handle the test results. The technical requirements further describe what is required in a fuzz testing environment to fulfill these strategies. Finally, we have prepared an appropriate test environment for integrating fuzz testing into a target system's CI pipeline. The fuzz testing tool is executed in an automated and continuous manner as part of the development process. Technical details about the implementation are presented and discussed. As a result, by integrating fuzz testing into a CI pipeline, it contributes to the overall DevSecOps toolchain, allowing automotive organizations to perform more comprehensive and systematic fuzz testing for detecting potential vulnerabilities early and continuously throughout development.

Introduction

To keep up pace with the development of advanced automotive systems, the automotive industry is evolving into a software-first industry. The continued development of ADAS, AV, EV, and connected vehicles as well as providing new functionality including OTA updates and backend services, cybersecurity is becoming a more important topic in the automotive industry. This is evident by various standardization activities and regulation on cybersecurity. For example, Automotive SPICE for Cybersecurity [1] was released in February 2021, the final version of final version ISO/SAE 21434 "Cybersecurity for road vehicles" [2] was released in August 2021, and TR-68:3 "Autonomous vehicles - Cybersecurity principles and assessment framework" [3] was released in September 2021. These different documents provide guidance to automotive organizations on how to consider and implement cybersecurity both on an organizational and project level. In particular, *fuzz testing* is mentioned as a recommended test method. Fuzz testing is a test technique where malformed or out-of-specification input are provided to the target system, which is then observed for any unexpected behavior. As such, fuzz testing is an efficient approach for detecting potential unknown vulnerabilities.

Since fuzz testing is a negative testing approach with virtually unlimited combinations of malformed input that can be provided to the target system, it is imperative to consider how to test as much as possible in an automated fashion. Thus, in this paper, we provide a practical suggestion on how to integrate fuzz testing into a Continuous Integration (CI) pipeline for automotive systems, which would contribute to the overall DevSecOps toolchain. It is important to elaborate on how to achieve a large coverage considering both quantitative and qualitative measures. Quantitative fuzz testing describes how much fuzz testing is performed; thus, it is a measurement of the code coverage and test coverage achieved during the fuzz testing. Qualitative fuzz testing is used as a measure to determine how "good" the fuzz testing cases are and how well exceptions on the target system can be detected [4]. More consistent fuzz testing during development also leads to improved robustness.

The main goal is to find potential issues that could lead to cybersecurity vulnerabilities in the target system earlier in the development lifecycle. Finding potential vulnerabilities earlier allows automotive organizations to fix these issues earlier which reduces the cost involved. The strategic steps for organizations to perform fuzz testing systematically as part of the process are described in more detail in [4].

lacking the necessary integration within the process. Embracing fuzz testing from the beginning as an inherent part of the CI pipeline and engaging with its findings early on from the development start offers a multitude of benefits. First of all, a much longer testing period until product release can be achieved which means more test cases can be covered. Second, the test environment can be iteratively adapted and tweaked during the whole development phase until a high code and test state coverage is achieved. The CI setup described within this paper provides a baseline for fostering these activities.

The other issue we identified is that fuzz testing is overall seen as a manual activity. This mindset does not only lead to fuzz testing being conducted late in the process as described above, but also has further shortcomings. Employees must be allocated and trained to manually put together the test environment, setup and finally run the fuzz tests. This is not only inefficient but also cumbersome for the employee and therefore prone to human error. Having these steps all automatized within the CI allows for a consistent and reliable fuzz test procedure. Human error which may occur within the setup is more easily reproducible and therefore detectable, leading to an overall better quality of fuzz tests. Due to the automatization the quantity of test cases is strongly increased as well, resulting in better code and test coverage.

Conclusion

In this paper we present an implementation of a fuzz testing tool integrated into the CI pipeline of a target automotive system. Fuzz testing is performed periodically and consistently during the development process to achieve greater test coverage. We present technical details from our implementation and discuss the benefits of integrating fuzz testing into the CI pipeline. In order to establish fuzz testing in the CI pipeline we also discuss fuzz testing strategies including how often to test, when to test, what to test, how to detect exceptions, as well as how to handle the test results. Specifically, we also provide a set of technical requirements for integrating fuzz testing into the CI pipeline.

As more complex automotive systems and more features for connected vehicles are being developed the risk for vulnerabilities and the likelihood for cybersecurity attacks also increase. Therefore, it is of utmost importance for automotive organizations to identify potential vulnerabilities early and continuously in the development lifecycle in an automated manner. Fuzz testing is an efficient test method for detecting unknown vulnerabilities. As such, performing fuzz testing in an automated manner consistently throughout development in a CI pipeline allows automotive organizations to detect and fix issues earlier in the development process. As the automotive industry continues to improve its cybersecurity posture through the implementation of cybersecurity standards such as ISO/SAE 21434, fuzz testing will also play an integral role in the overall DevSecOps toolchain in helping automotive organizations develop more secure systems.

Outlook

For automotive organizations to successfully be able to integrate fuzz testing into the CI pipeline requires several considerations. First, organizations need to review and update their existing processes to also incorporate fuzz testing as part of the validation strategy. Next, organizations must review and define fuzz testing strategies and determine the technical requirements to establish an appropriate fuzz testing environment as presented in this paper. We plan to continue this work by running fuzz testing of multiple real software-heavy automotive targets in parallel in the CI pipeline. Further, as part of future work we are also exploring how to perform fuzz testing on software-only targets, for example, in virtualized or emulated environments in combination with SIL (software-in-the-loop) systems. Moreover, we are also exploring how to process the results from fuzz testing into evidence reports and store in management systems in an automated manner.

References

1. VDA, "Automotive SPICE® for Cybersecurity," 2021.
2. ISO/SAE, "ISO/SAE 21434: 2021 - Road Vehicles — Cybersecurity Engineering," 2021.
3. Manufacturing Standards Committee, "TR 68 - 3 : 2021 Autonomous Vehicles - Part 3: Cybersecurity Principles and Assessment Framework," 2021.
4. Vinzenz, N. and Oka, D., "Integrating Fuzz Testing into the Cybersecurity Validation Strategy," SAE Technical Paper 2021-01-0139, 2021, https://doi.org/10.4271/2021-01-0139.
5. Knudsen, J. and Varpiola, M., *Fuzz Testing Maturity Model*, (Synopsys Whitepaper, May 2017).
6. Oka, D.K., *Building Secure Cars: Assuring the Automotive Software Development Lifecycle*, (Wiley, 2021).
7. Oka, D.K., Fujikura, T., and Kurachi, R., "Shift Left: Fuzzing Earlier in the Automotive Software Development Lifecycle using HIL Systems," in in *Escar Europe*, Brussels, Belgium, 2018.
8. Kuipers, R. and Oka D.K., "Improving Fuzz Testing of Infotainment Systems and Telematics Units Using Agent Instrumentation", in in *Escar USA*, Ypsilanti, MI, USA, 2019.
9. Atlassian, "Jira Software," https://www.atlassian.com/software/jira (accessed 11 Oct 2021).
10. Synopsys, "Code Dx: Faster and More Scalable AppSec Through Automation," https://codedx.com (accessed 11 Oct 2021).
11. Intland Software, "codebeamer: Simplify Complex Product and Software Engineering at Scale," https://intland.com/codebeamer/ (accessed 11 Oct 2021).
12. Oka D.K., Makila T., and Kuipers R., "Integrating Application Security Testing Tools into ALM Tools in the Automotive Industry," in 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Sofia, Bulgaria, 2019.

13. libFuzzer, “libFuzzer - a Library for Coverage-Guided Fuzz Testing,” URL: https://llvm.org/docs/LibFuzzer.html.
14. Jenkins, “Jenkinsfile,” URL: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/.
15. AddressSanitizer, “AddressSanitizer (aka ASan)” URL: https://clang.llvm.org/docs/AddressSanitizer.html.
16. Valgrind, “Valgrind,” URL: https://www.valgrind.org/.
17. Synopsys, “Defensics,” URL: https://www.synopsys.com/software-integrity/security-testing/fuzz-testing.html.
18. GitLab, “GitLab,” URL: https://about.gitlab.com/.
19. Jenkins, “Jenkins,” URL: https://www.jenkins.io/.
20. Dadam, S.R., Zhu, D., Kumar, V., Ravi, V. and, Palukuru V.S.S., “Onboard Cybersecurity Diagnostic System for Connected Vehicles,” SAE Technical Paper 2021-01-1249, 2021, doi:10.4271/2021-01-1249.

Contact Information

Dennis Kengo Oka, Ph.D.
+81 3-6746-3600
dennis.kengo.oka@synopsys.com

Nico Vinzenz, M.Sc.
+49 7541 77-962500
nico.vinzenz@zf.com

Definitions/Abbreviations

ALM	Application Lifecycle Management
CAL	Cybersecurity Assurance Level
CD	Continuous Deployment
CEP	Cybersecurity Engineering Process
CI	Continuous Integration
E/E architecture	Electrical/Electronic architecture
ECU	Electronic Control Unit
FTMM	Fuzz Testing Maturity Model
HD maps	High-Definition maps
HiL	Hardware-in-the-Loop
HPC	High Performance Controller
LiDAR	Light Detection and Ranging
SiL	Software-in-the-Loop
SUT	System/Software-under-Test
TARA	Threat Analysis and Risk Assessment
TCU	Telematic Control Unit
V2X	Vehicle-to-Everything
VCS	Version Control System
VCU	Vehicle Control Unit

ISSN 0148-7191