2018 25th Asia-Pacific Software Engineering Conference (APSEC)

Fuse: An Architecture for Smart Contract Fuzz Testing Service †

W.K. Chan
Department of Computer Science
City University of Hong Kong
Hong Kong
wkchan@cityu.edu.hk

Bo Jiang
School of Computer Science and Engineering
Beihang University
Beijing, China
jiangbo@buaa.edu.cn

***Abstract*—In this paper, we report our project Fuse, which is a fuzz testing service. It presents the Fuse architecture and discusses the progress and technical issues to be addressed to fuzz-test smart contracts and support fuzz-testing of Dapps.**

***Keywords*—blockchain, fuzz testing, Dapps, architecture, security vulnerability, smart contract, Ethereum**

I. Introduction

Testing is vital to software development projects. Ethereum smart contracts [3] are a kind of program running on Ethereum Virtual Machine (EVM). They are deployed on top of the blockchain technology stack, making each smart contract in production immutable in the long run and smart contracts' transactions in each valid block at each blockchain node to be executed locally.

A decentralized application (Dapp) contains a set of smart contracts and other components. Like any kind of applications, a smart contract in a Dapp may contain bugs or require enhancement such as handling missed cases with or without changes in functional signatures of the functions in smart contracts. Popular de facto approaches to updating a version of a smart contract to a newer version are to *either* de-construct that smart contract and deploy a new one *or* incorporate an address diversion facility such as a proxy pattern implementation through the system call deletecall() to forward each called function at the present contract address to a new contract address. A development project of a Dapp may thus include not only many rounds of evolution of its smart contracts but also transactions at the production blockchain network to update various such addresses. The combinations of deletecall() and changes in functional signature and the values concatenated as a byte array may further increase the security vulnerability potentials of the corresponding Dapps.

In this paper, we present the progress of building a new kind of testing service called ***Fuse*** to support the fuzz testing of smart contracts and explain how it can support fuzz testing of Dapps.

Section II outlines the architecture of Fuse, and discusses the challenges. Section III concludes this paper.

II. Fuse

A. Architecture

Figure 1 depicts the overall architecture of *Fuse*. We use the following scenario to illustrate how Fuse works.

† This research is supported in part by NSFC (project no. 61772056), the Research Fund of the MIIT of China (project no. MJ-Y-2012-07), the GRF of Research Grants Council (project no. 11214116, 11200015, and 11201114), and the State Key Laboratory of Virtual Reality Technology and Systems.

Scenario: A developer has a smart contract usually ready for testing and unavailable to the public. S/he first submits an encrypted binary of the smart contract to be tested or the URL of a deployed smart contract (denoted as smart contract A in the figure) via a *web portal* to Fuse. Then, the *Smart Contract Acquisition* module decrypts the received submission to extract contract A, and passes contract A to the *Smart Contract Preparation* module. The Smart Contract Preparation module extracts all the function signatures of contract A and passes the functional signatures to the *Fuzz Test Generation* module, and deploys contract A to a testnet within Fuse. The Fuzz Test Generation module then generates fuzz test cases to test the smart contract A together with a set of other smart contracts already deployed on the same testnet. In the course of execution of these smart contracts, the *Execution Profiling* module profiles their control flow information, the interactions between smart contracts' transactions and messages and program states, and passes these information to the *Vulnerability Detection* module as well as the Fuzz Test Generation module. The Vulnerability Detection module checks whether some blockchain accounts or messages or transactions result in violation of the predefined test oracles. It marks the execution points for those detected vulnerability (i.e., a violation of any test oracle defined in Fuse).

The detected vulnerability and the execution scenario leading to the vulnerability are then passed to the *Test Scenario Encoding* module. This module encode the test scenario in a static form, which is further passed to the *Test Report Generation* module. The Test Report Generation module then combines the fuzz testing profile received from the Smart Contract Preparation module and the test scenario from the Test Scenario Encoding module to generate an encrypted test report for contract A. The encrypted test report will send via a web portal back to the developer.

The developer decrypts the test report using a computer in his/her space and watches the test scenario through the *Test Scenario Visualization* module (e.g., in the user' web browser). The module is designed to be executable without network connectivity, and thus users may use a standalone machine without networking connectivity to go through the visualization of the test report.

B. Progress

We [1] have made research progress for the *Fuzz Test Generation* module, the *Execution Profiling* module, and *Vulnerability Detection* module as follows. We have defined the first set of test oracle definitions with respect to seven classes of vulnerability patterns but realized as an offline analysis. To facilitate these offline analyses, we have profiled the caller-callee relations, input and output values of the called functions as well as the executed

978-1-7281-1970-0/18/$31.00 ©2018 IEEE
DOI 10.1109/APSEC.2018.00099

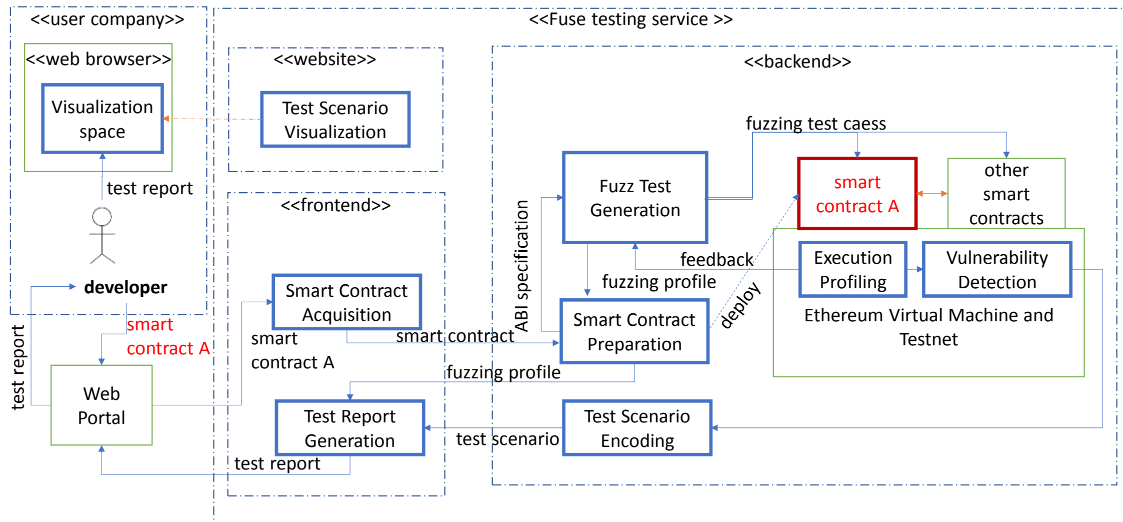

Fig. 1. Architecture of Fuse

opcode in each called function and the gas stipend allowed for that call. We have proposed a novel form of fuzz testing with constant seeding as the strategy to generate test transactions in which it fuzz-tests the whole set of deployed smart contracts and constants needed are extracted from this whole set. Our experiment results showed that the current combination of techniques can detect security vulnerability issues of these seven classes precisely with a true positive rates of 96-100%, and Table 1 shows the results [1]. We have made the tool called ContractFuzzer publicly available [2].

Table 1 Summary of Vulnerabilities Detected [1]

Vulnerability Class	# of Detected Vulnerabilities	True positive rate
Gasless Send	138	1.000
Exception Disorder	36	1.000
Reentrancy	14	1.000
Timestamp Dependency	152	0.960
Block Number Dependency	82	0.963
Freezing Ether	30	1.000
Dangerous Delegatecall	7	1.000

C. *Discussion*

To develop Fuse, we still need to address a number of technical issues. Our fuzz testing prototype [2], even though precise in practice, are still restrictive. For instance, compared to Oynete [4][5], the prototype of ContractFuzzer detected about 50% fewer true positive cases in the experiment [1]. Fuse will address this issue by expanding the definitions of test oracle and incorporating a new testing technique in generating test data. Another issue is that the offline analysis could be inefficient. We plan to study an online counterpart. However, online analysis may further slow down the execution of smart contracts, which is an issue to be addressed.

Fuse also aims to provide test scenarios for developers' reference. Our experience in path tracing [6] is that the data log for encoding the control flow of a test scenario could be large in size, prohibiting the transfer of a test scenario to a developer's site for visualization in private. Fuse aims to address the log size problem while facilitating visualization in high fidelity. One interesting observation is that the test scenario encoding can be performed offline as the number of cases that expose vulnerability issues should be in small minority. One possible option is to develop on top of the notion of HPPDAG [6] to encode such a test scenario.

A smart contract may interact with some other modules in a Dapp. A Dapp may generate wrong transactions to a blockchain for smart contract execution. This part should also be tested. We are studying how Dapp testing can be performed together with fuzz testing on smart contracts.

III. Conclusion

This paper has presented the Fuse architecture and discussed the progress and a plan for actions to realize Fuse. to fuzz-test smart contract and Dapps and assist developers for test diagnosis via test scenario visualization.

References

[1] Bo Jiang, Ye Liu, and W. K. Chan. ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (ASE 2018). ACM, New York, NY, USA, 259-269, 2018. DOI: https://doi.org/10.1145/3238147.3238177

[2] ContractFuzzer. https://github.com/gongbell/ContractFuzzer (last access on 26 Sept 2018)

[3] Etherenum. https://www.ethereum.org/ (last access on 26 Sept 2018)

[4] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making Smart Contracts Smarter. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS 2106)*, pp:254-269, Vienna, Austria, 2016. DOI: https://doi.org/10.1145/2976749.2978309

[5] Oynete. https://github.com/melonproject/oyente (last access on 26 Sept 2018)

[6] Chunbai Yang, Shangru Wu, and W. K. Chan. Hierarchical Program Paths. *ACM Transactions on Software Engineering and Methodology*, 25(3), Article 27, 44 pages, 2016. DOI: https://doi.org/10.1145/2963094