2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW) | 979-8-3503-1956-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISSREW60843.2023.00041

On the Efficacy of Smart Contract Analysis Tools

Silvia Bonomi
Sapienza University of Rome
bonomi@diag.uniroma1.it

Stefano Cappai
Sapienza University of Rome
cappai.1844363@studenti.uniroma1.it

Emilio Coppa
Sapienza University of Rome
coppa@diag.uniroma1.it

***Abstract*—Distributed Ledger Technologies are an emerging reality opening the way to new application design paradigms like smart contracts-based distributed applications. If on one side they are creating new markets and opportunities, on the other they are exposing users to new security issues deriving from the scarce maturity in terms of security practices in their design and development. This paper raises a warning about the efficacy of a state-of-the-art software testing tool, namely Mythril, by challenging it with real smart contracts extracted from the Code4arena competitions and comparing its performance with security audits released during the contests. The paper highlights possible root causes of inefficiency, opening the way toward more scalable and efficient smart contract testing tools.**

***Index Terms*—Smart contracts, vulnerability detection, symbolic execution, software testing, blockchains.**

I. Introduction

Distributed Ledger Technologies and Decentralized Applications , implemented through *smart contracts*, are becoming essential components in a large variety of modern solutions in many different application domains ranging from finance to agriculture. A Distributed Ledger is essentially a replicated storage system accessible by its customers that allows performing transactions i.e., atomic updates to the state of the ledger. A smart contract is a software program that is intended to automatically execute, control, or document events and actions according to the terms of a contract or an agreement (i.e., the business logic behind the transactions recorded in the ledger). A smart contract is *deployed* on the Distributed Ledger by executing a particular transaction on the ledger itself that enables its execution. Smart contracts can store arbitrary states and execute arbitrary computations. Clients may interact with a smart contract by invoking its operations which are then translated into transactions recorded in the ledger and that may generate the invocation of other smart contracts and might result in changing the state of the ledger (e.g., sending coins from one smart contract to another).

Given the pivotal role of smart contracts in several modern technologies, a growing concern involves their security. Indeed, in the context of Decentralized Finance (DeFi), vulnerabilities discovered in real-world smart contracts have already shown to lead to significant economic damages, with hundreds of millions of funds stolen from armless victims [1]. To make the situation even worse, smart contracts cannot be easily updated, as typically done in traditional software systems, due to the immutability property of the underlying Distributed Ledger (i.e., the capability of the ledger to preserve the integrity and durability of the recorded transaction), requiring non-trivial patching procedures.

Due to these concerns, the research community and the industry have started in the last decade to investigate software testing methods and security tools that can possibly identify bugs and vulnerabilities[1] in smart contracts before these software components are deployed on blockchains. As for traditional software, several techniques could be considered.

Formal approaches can provide the strongest guarantees but require in most cases to redesign the blockchain technologies and how the smart contracts are written by the developers, making their adoption often impractical.

Lightweight static analyses [3] have seen large adoption in the blockchain community as they can efficiently provide insights into the possible flaws of smart contracts. To favor scalability, these approaches perform a *local* analysis, possibly missing flaws involving *global* behaviors and producing invalid alerts, requiring manual validations.

In recent years, the community has started to consider more heavyweight analyses, aiming at better accuracy and efficacy in terms of vulnerability discovery. One significant candidate is symbolic execution [4], which evaluates the program's behaviors by considering *symbolic* inputs, whose values are not fixed a-priori but are determined based on the program execution, trying to explore all the possible execution paths. While powerful, this technique hardly scales in the presence of complex software. Among the large set of tools, Mythril [5] emerges as one of the most mature frameworks for blockchains that are compatible with Ethereum Virtual Machine. Compared to lightweight static analyses, Mythril tends to generate fewer alerts, due to its limited scalability, but can demonstrate the actual feasibility of the identified attacks.

Our contributions. In this paper, we report on the preliminary results of an experimental investigation on the efficacy of Mythril and Code4rena bots race participants on real-world smart contracts. In particular, we consider the smart contracts that were part of four competitive contests from Code4Arena.com, a platform where developers can obtain security audits from the community in exchange for monetary rewards. We have compared the results from Mythril with vulnerabilities reported by the bots and humans during the contests. Our preliminary evaluations show that Mythril is unable to find several security flaws that were detected by

[1]We follow the intuition from [2]: *if an issue leads to a planned scenario not running, it is a bug*; *if an issue leads to an unplanned scenario running, it is a vulnerability*. In this paper, we mainly focus on vulnerabilities.

979-8-3503-1956-9/23/$31.00 ©2023 IEEE
DOI 10.1109/ISSREW60843.2023.00041

Contest	# Smart Contracts	SLOC	Vulnerabilities found by			Vulnerabilities found by both		Max (Avg) Analysis Time
			Bots	Humans	Mythril	Bots ∧ Mythril	Humans ∧ Mythril	
Asymmetry	4	460	0	20	0	0	0	7872 (2055) secs
Llama	11	2096	2	5	0	0	0	11991 (2635) secs
Juicebox	1	160	3	3	0	0	0	2578 (2578) secs
Stader	22	4334	1	15	2	0	2	39518 (5174) secs

TABLE I
Summary of the preliminary experimental evaluation.

humans, suggesting that several enhancements to the original approach may be needed to improve its efficacy.

II. Preliminary Experimental Study

Setup and dataset. The left side of Table I reports the four contests considered by this investigation. Code4arena has publicly released two types of reports: (a) flaws identified by contestants' bots when given a one-hour time budget and (b) vulnerabilities reported by contestants when given a 5/7-day time budget. The bots may be representative of the state-of-the-art for fully-automatic lightweight static analyses, while the reports from humans may be representative of the state-of-the-art of advanced security auditing practices. From the reports, we have ignored flaws marked with a low severity as they were often treated differently, or even skipped, by different tools, making hard a comparison. In this investigation, we compare these reports with Mythril 0.23.15 with a 24-hour time budget (for each contract) using a docker container on an Intel Xeon E5-4610v2 CPU and 256 GB of RAM.

Research Questions. Our study targets these questions:

RQ1 How does Mythril compare to bots' analysis?
RQ2 Can Mythril provide answers within the 1-hour budget?
RQ3 How does Mythril compare to humans' analysis?
RQ4 Is Mythril missing flaws due to its poor scalability?

RQ1. Table I shows that bots have reported more vulnerabilities than Mythril in 2 out of 4 contests. Moreover, in Stader, they have identified one flaw that was missed by Mythril. This result is unexpected as Mythril has a larger time budget (1 day versus 1 hour) and it should be able to find similar, or possibly more, security insights. This may point out that Mythril likely needs more fine-grained detection capabilities and scalability.

RQ2. The rightmost column of Table I reports the maximum and average running time of Mythril. In most cases, it needs more than 1 hour to terminate its analysis. When considering the only contest where it identifies two flaws, it needs up to 10 hours to provide such an answer. Hence, Mythril is not yet likely a good candidate for evaluations that must be finished within a limited time budget (e.g., during continuous testing).

RQ3. A natural question is whether Mythril, although slower than the bots, can at least find most flaws in the considered contests. When looking at Table I, we see that Mythril is missing a significant number of vulnerabilities in most contests, compared to humans.

RQ4. Given the unimpressive results from Mythril, we attempted to understand why this tool may miss flaws in the four contests. On 40 out of 49 flaws, we manually evaluated that Mythril cannot identify them even when given an infinite time budget. Indeed, the root cause of these flaws is not currently even contemplated by Mythril. For instance, 22 flaws could be categorized as the *improper implementation of the business logic*: since Mythril is agnostic to the logic, it struggles to detect them. Similarly, 10 bugs can be seen as *incorrect accounting*, e.g., wrong use of an exchange rate, and Mythril is unaware of these accounting mechanisms. Moreover, 7 vulnerabilities are due to *inconsistent state updates*, e.g., when different variables should be updated together but the code fails to do it, and Mythril is unaware of such conceptual links. Finally, one bug is an *atomicity violation*. Many of these categories are known to be hard to find but the community is developing better and better tools, encouraged by rewards. Due to the lack of space, we refer to [6] for a deeper discussion.

Interestingly, a recent work [7] has also analyzed the root causes of vulnerabilities from several real-world smart contracts: their results confirm our insights that state-of-the-art tools can miss several crucial bugs, especially due to the lack of modelization of business logic aspects.

III. Concluding Remarks

The paper provided an evaluation of the Mythril tool when fed with real smart contracts extracted from the Code4arena competition. Our findings highlight the need for further investigation and improvements in symbolic execution tools. In particular, we are currently extending the current evaluation by considering different configuration settings for the Mythril tool to better classify sources of inefficiency. In addition, we are continuing with the data collection of smart contracts and their audits to validate our claims on a larger sample of contracts.

Acknowledgment

This work is partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU and by the CINI project Blockchain per Filiera Sicura.

References

[1] "The growing rate of defi fund loss," https://twitter.com/PeckShieldAlert/status/1520620826613010432, 2022.
[2] K. Gorshkov, "Bug vs Vulnerability: Know Both Your Enemies," https://blog.smartdec.net/bug-vs-vulnerability-d6d4dc4068bd, 2018.
[3] J. Feist, G. Greico, and A. Groce, "Slither: A static analysis framework for smart contracts," ser. WETSEB '19, 2019.
[4] R. Baldoni, E. Coppa, D. C. D'Elia, C. Demetrescu, and I. Finocchi, "A Survey of Symbolic Execution Techniques," *ACM Comp. Surveys*, 2018.
[5] Consensys, "Mythril," https://github.com/Consensys/mythril, 2022.
[6] S. Bonomi, S. Cappai, and E. Coppa, "Smart contract tool analysis," https://github.com/niser93/SmartContractToolAnalysis/, 2023.
[7] Z. Zhang, B. Zhang, W. Xu, and Z. Lin, "Demystifying exploitable bugs in smart contracts," ser. ICSE '23, 2023.