SSURJET

Sir Syed University Research Journal of Engineering & Technology
2022, Vol. 12, No. 1,
https://doi.org/10.33317/ssurj.421

Empirical Analysis of Vulnerabilities in Blockchain-based Smart Contracts

Kashif Mehboob Khan[1], and Ansha Zahid[2]

[1]*Department of Software Engineering, NED University of Engineering and Technology, Karachi. Pakistan*
[2]*Department of Computer Science and Information Technology, NED University of Engineering and Technology, Karachi. Pakistan*

Correspondence Author: Kashif Mehboob Khan (kashifmehboob@neduet.edu.pk)

Received January 10, 2022; Revised March 16, 2022; Accepted April 21, 2022

Abstract

With the evolution of technology, blockchain a swiftly impending phenomenon i.e., "decentralized computing" is observed. The emergence of Smart Contracts (SC) has resulted in advancements in the application of blockchain technology. The Ethereum network's computing capabilities and functionalities are founded on the basis of SC. A smart contract is a self-executing agreement between buyer and seller with the terms of the settlement between them, written directly as lines of code, existing across a distributed decentralized blockchain network. It is a decentralized software that runs on a blockchain autonomously, consistently, and publicly. Conversely, due to the complex semantics of fundamental domain-specific languages and their testability, constructing reliable and secure SC can be extremely difficult. SC might contain some vulnerabilities. Security vulnerabilities can originate from financial tribulations; there are a number of notorious events that specify blockchain SC could comprise numerous code-security vulnerabilities. Security and privacy of blockchain-based SC are very important, we must first identify their vulnerabilities before implementing them widely. Therefore, the purpose of this paper is to conduct a comprehensive experimental evaluation of two current security testing tools: Remix solidity static analysis plugin and Solium which are used for static analysis of SC. We have conducted an empirical analysis of SC for finding tangible and factual evidence, controlled by the scientific approach. The methodology's first step is to gather all of the Ethereum SC and store them in a repository. The next step is to use the Remix solidity static analysis plugin and Solium to perform vulnerability assessments. The last step is to analyze the result of both tools and evaluate them on the basis of accuracy and effectiveness. The goal of this empirical analysis is to evaluate the two FOSS tools: Remix solidity static analysis plugin and Solium on the basis of accuracy and effectiveness. Some research questions were considered to reach the stated goal: What automated tools and frameworks are proposed in supporting the state-of-the-art empirical approach to SC vulnerability detection? How accurate are security analysis tools? And which tool has more accuracy rate? How effectively security analysis tools are detecting vulnerabilities in SC? And which is the most effective security analysis tool? We investigated the effectiveness and accuracy of security code analysis tools on Ethereum by testing them on a random sample of vulnerable contracts. The results indicate that the tools have significant discrepancies when it comes to certain security characteristics. In terms of effectiveness and accuracy, the Remix plugin outperformed and beat the other tool.

Index Terms: *Blockchain, Smart Contracts, Ethereum, Static Code Analysis, Security.*

I. INTRODUCTION

There has been a surge of interest in blockchain technology and cryptocurrency from both the academic community and the industry. Essentially, blockchain technology is a decentralized public ledger that relies on encryption to securely host apps, send digital currency, and store data on the network. It is no secret that Ethereum is one of the most popular blockchain systems, based on the existing cryptocurrency market capitalization. During one of the panel discussions, Vitalik Buterin [1], the core Ethereum founder, described Ethereum as a general-purpose blockchain. This means that the Ethereum network is sufficient to facilitate algorithms written in general-purpose programming languages. A range of applications may be created from basic wallets to complex financial systems, energy-trading platforms, and even new and unique cryptocurrency systems. As an alternative to developing a distinct blockchain for each use case or application, smart contracts may serve multipurpose domains of applications [2].

A. Motivation

In recent years, Smart Contracts (SC) have become more and more popular and are considered to be the next generation of automation based on agreements between

parties in a blockchain system. The idea of a 'Smart Contract' was first envisioned at the source code level. In blockchain programming (including the development of smart contracts) and also in other computing paradigms, code-based testing is still preferred and considered effective because it determines a decent standard of dependence on the most comprehensive artifact of the development process prior to actually deploying it to the hosting environment. Furthermore, static security analysis before the deployment of SC appears to be an ideal solution, due to the unique structure of blockchain-based SC. Based on these considerations, this paper concentrates largely on the automated static SC security mechanisms on the Ethereum blockchain by concentrating its emphasis on Solidity. SC are independent agreements implemented using software, and their implementation ensures compliance with calculation and measurement conditions. SC's major goal is to encourage the replacement of traditional trusted third parties (authorities, entities, or organizations) with bits of code operating on a decentralized and immutable system. This new approach to SC applications offers up hundreds of new possibilities. IoT security and forensics are one of the most potential topics which have attracted a lot of attention from academics as well as the corporate world [3-5]. In reality, blockchain's implementation and use have far outstripped its initial goal as the foundation of the world's first decentralized cryptocurrency. Other sectors have realized the advantages of a trustless, decentralized ledger with historical immutability and are trying to apply the basic ideas to existing business operations. Because of the blockchain's unique features, its implementation in any industry is an appealing notion. The main challenge in blockchain-enabled IoT security is controlling and knowing "who" will connect to the network across a huge number of objects (e.g. Sensors and devices) without violating data privacy [6]. Blockchain-based decentralized smart contracts appear to be an important solution [7] and [8], particularly when dealing with security vulnerabilities in far dispersed IoT nodes [9]. Blockchain technology is critical to achieving the security framework anticipated by SC, and it looks to offer tremendous potential for future IoT progress. However, blockchain technology is believed to be safe and secure by design, its integrated applications which are SC in dynamic settings (such as the Internet of Things) may present vulnerabilities in real-world scenarios. Indeed, such smart contract applications that govern nodes and transactions are only segments of code created by human developers. Furthermore, because of their unique nature, errors or defects might have huge financial consequences, therefore security is critical. Too far, smart contracts have been damaged and harmed by unfortunate occurrences and assaults (for example, a reentrancy fault in the split DAO function resulted in a $40 million loss in June 2016, and $32 million was stolen by attackers owing to a flaw in the code in November 2017). These high-profile cases indicate that developers (even experienced ones) may leave security issues and flaws in smart contracts, creating major vulnerabilities for attackers to exploit [2]. Because of the sheer breadth and pace of IoT settings, this would be expanded much further. As a result of the complex semantics of the underlying domain-specific languages and their testability, developing trustworthy and safe SC may be exceedingly challenging [10-21].

B. Related Work

This section of the paper consists of related work which includes different vulnerability analysis tools, approaches, surveys, and experiments in this area. There are a number of security analysis tools available for detecting SC vulnerabilities. Different types of technical methods were used in these tools for the implementation of security analysis on SC.

Empirical analysis of Free and Open Source Software (FOSS) tools is conducted by Reza M. Parizi and associates for security testing and to achieve this objective it is examined in terms of vulnerability detection and how effective and accurate these automated smart contract security testing tools are. The four FOSS tools namely 'Oyente', 'Mythril', 'Security', and 'SmartCheck' were empirically analyzed based on their vulnerability detection effectiveness and accuracy. Out of four automated security testing tools, SmartCheck is statistically more effective having a 95 % significance level but in terms of accuracy, Mythril tool showed the highest accuracy score with issuing the lowest number of false alarms among peer tools, though it had less effectiveness than SmartCheck, however, Oyente missed a large number of threats and it was considered to be the least effective tool out of all four, Security is the tool which showed a stable performance throughout testing It had a large number of false positives but is still catching more threats that Oyente was missing. So it can be concluded that SmartCheck is the most effective security testing tool for smart contracts developed in Solidity on the Ethereum blockchain but it is less accurate than Mythril [22].

Haijun Wang and fellow researchers proposed a new tool for detecting vulnerabilities known as 'VOLTRON' which will detect irregular transactions. The main purpose of smart contracts is to manage the transfer of assets and perform bookkeeping. There are two invariants in smart contracts for transactions, Balance invariant and Transaction invariant. The approach behind this tool was to use proposed balance and transaction invariants to detect vulnerabilities. According to the balance invariant, after a transaction, if the bookkeeping balance is not updated correctly it indicates that some irregular event has occurred. It is the requirement of balance invariant that the difference between contract balance and the sum of all participants' bookkeeping balances remain constant. The transaction invariant requires that the amount deducted from a contract's bookkeeping balance is always deposited into the recipient's account. The challenges in the proposition of invariants include identification of bookkeeping variables, handling of non-currency assets, and verification of invariants. Four vulnerabilities were selected on which the VULTRON approach was tested; these vulnerabilities are Reentrancy, Exception Disorder, Gasless Send, and Integer Overflow/Underflow. At last, it was concluded that the approach presented in this paper detected all these vulnerabilities [23].

A new model named ContractWard was proposed by Wei Wang et al. for the detection of six different types of vulnerabilities of smart contracts. This model was based on

extracted static characteristics in order to secure the contract layer on Ethereum and for the purification of decentralized applications. In this paper, 3 supervised ensemble classification algorithms, namely, 'XGBoost', 'AdaBoost' and 'RF', and two simple classification algorithms, namely, SVM and KNN, together with two sampling methods, namely, 'SMOTETomek' and 'SMOTE' were employed to conduct comparative experiments. The effectiveness and efficiency of ContractWard were demonstrated by the experimental results. The previously available methods named Oyente and Securify have a slower vulnerability detection speed as compared to ContractWard, it is best for rapid batch detection of vulnerabilities in smart contracts with an average detection speed of four seconds per smart contract. This model worked effectively on smart contracts written in all high-level languages such as Solidity, Serpent, and LLL. In the future, for the improvement of CounterWard, designing anomaly detection models will be focused to detect novel vulnerabilities in smart contracts [24].

A static analysis framework named Slither was described by Josselin Feist and associates in their paper. The working of Slither was defined as it first converts SC written in Solidity into an intermediate representation called SlithIR, which uses Static Single Assignment (SSA) form and reduced instruction set to ease implementation of analyses while preserving semantic information that would be lost in transforming Solidity to bytecode. This framework will provide automated detection of vulnerabilities, automated detection of code optimization opportunities, improvement of the user's understanding of the contracts, and assistance with code review. The capabilities of Slither were evaluated on real-world contracts. The bug detection capability of Slither is faster than other tools, it detects 20 different types of vulnerabilities of smart contracts. In terms of speed, robustness, the balance of detection, and false positives, Slither's bug detection is fast, accurate, and outperforms other static analysis tools. These tools were compared with Slither using a large dataset of 1000 smart contracts and the result was manually reviewed [25].

Lei Pan et al. categorized the SC security analysis methods into three types static analysis, dynamic analysis, and formal verification methods. Static code analysis is the method of debugging source code by automatically examining it before the execution of a program. The static analysis methods used in the paper by the authors, include OYENTE, ZEUS, GASPER, Vandal, Ethir, and Securify. Dynamic code analysis is a method that debugs the source code of a smart contract while executing it during run time. The dynamic analysis methods used in the paper include MAIAN and Graph Construction. Formal verification methods use mathematical formal methods or theorem provers to prove the properties of SC. The formal verification analysis conducted to validate and prove vulnerabilities in SC includes F* Framework, Formalization using Isabelle/HOL, and FEher interpreter using Coq. These three security analysis methods were then compared in terms of their accuracy, performance, and coverage of finding vulnerabilities. It was concluded by Lei Pan that static and dynamic analysis detects only defined vulnerabilities; however, formal verification methods use mathematical theorems and formal methods validate SC properties with proofs [26].

Kalra et al. proposed the ZEUS framework for verifying the validity and fairness of smart contracts. To swiftly verify contracts for safety, ZEUS combines both abstract interpretation and symbolic model checking, as well as the power of constrained horn clauses. The authors created a ZEUS prototype for Ethereum and Fabric13 blockchain platforms, which they tested with smart contracts. 94.6 percent of contracts (worth more than USD 0.5 billion) were found to be securely vulnerable, according to the analysis [27].

According to our research, not a single research paper has used solium and remix plugin (as it been included recently in remix ide) and the vulnerabilities detected in our research were also different than previously detected vulnerabilities that's why we considered our research unique and better than others.

Table 1: Popular Tools and their Characteristics in Existing Research

Tool	Analysis Basis	Analysis Type	Description
ZEUS	Source code (.sol)	Static	ZEUS [27] can certify the fairness of smart contracts by verifying their validity. ZEUS verifies the safe programming practices of susceptible smart contracts by combining an abstract interpreter with a symbolic model checker.
Oyente	Source code (.sol)	Static	OYENTE [28] is a static analysis tool that finds security vulnerabilities in smart contracts. The inputs are a smart con- tract's bytecode and Ethereum's current global state. CFG Builder, Explorer, Core Analysis, and Validator are the four modules of OYENTE
Smart Check	Source code (.sol)	Static	SmartCheck [29] was developed by the SmartDec Security Team, it is an automated static code analyzer. It analyses Solidity source code and examines smart contracts for security flaws and poor practices automatically. It parses smart contract code into an abstract syntax tree, converts it to XML, and uses XPath to look for vulnerability patterns.
Gasper	Byte Code	Static	GASPER is a static analysis tool and was developed by Chen et al. [30] in order to discover smart contracts with inefficient gas use.
Vandal	Byte Code	Static	Vandal [31] is a framework for analyzing Ethereum SC for security flaws. To transform EVM bytecode to semantic logic relations, an analysis pipeline is employed.

C. Contribution

In this research work, we attempt to make the following contributions to investigating and highlighting potential SC vulnerabilities:

i. We have performed empirical analysis to evaluate the two FOSS tools: Remix solidity static analysis plugin [32] and solium [33] on the basis of accuracy and effectiveness.
ii. A methodology for detecting smart contracts vulnerabilities has been proposed with our own designed algorithm that has been applied to show the effectiveness of our proposed approach
iii. A comprehensive comparative analysis of automated tools and frameworks is proposed in supporting the state-of-the-art empirical approach to smart contracts vulnerability detection.

D. Organization

The paper has been organized as follows:
Section II contains the state-of-the-art techniques and approaches which have been implemented in various tools to determine the vulnerabilities and security loopholes in smart contracts. Section III discusses in detail, about the problem to be focused on and its proposed methodology. Section IV and Section V highlight the experimentation and results while Section VI concludes the paper.

II. STATE OF THE ART

There have been few state-of-the-art tools and approaches which have been applied for detecting vulnerabilities in smart contracts. Out of four automated security testing tools, as mentioned in [22], SmartCheck is statistically more effective having a 95 % significance level but in terms of accuracy, Mythril tool showed the highest accuracy score with issuing the lowest number of false alarms among peer tools, though it had less effectiveness than SmartCheck. However, Oyente missed a large number of threats and it was considered to be the least effective tool out of all four, Securify is the tool that showed a stable performance throughout testing It had a large number of false positives but is still catching more threats that Oyente was missing. So it can be concluded that SmartCheck is the most effective security testing tool for smart contracts developed in Solidity on Ethereum blockchain but it is less accurate than Mythril [22]. VULTRON approach was also proved to be very effective against vulnerabilities such as Reentrancy, Exception Disorder, Gasless send, and Integer overflow/underflow. Upon testing, this approach was able to detect all these vulnerabilities [23]. The capabilities of Slither tool were evaluated on real-world contracts. The bug detection capability of Slither is faster than other tools, it detects 20 different types of vulnerabilities of smart contracts [25].

III. PROBLEM STATEMENT AND ITS PROPOSED SOLUTION

A. Problem Description

In this paper, our focus is to address the dilemma of smart contracts' vulnerabilities and the effectiveness of their accuracy by taking into account its number of functions and line of code as a measure of its function for empirical analysis against the (increasing) number of line of codes in SC.

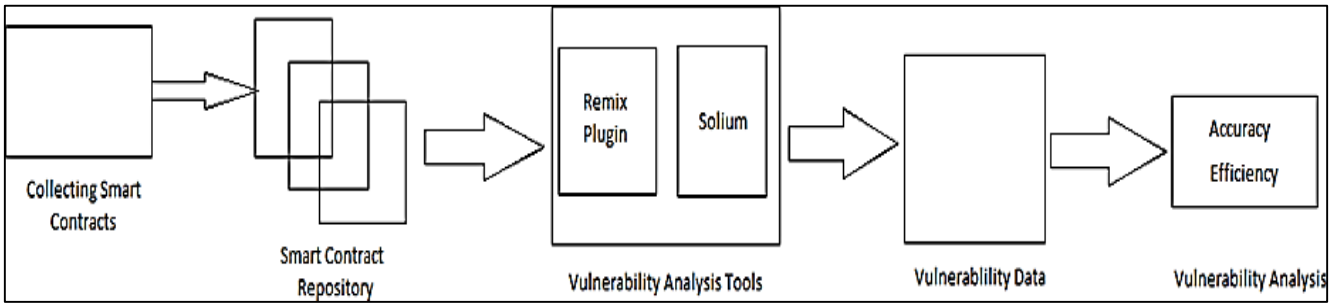

Figure 1: System Diagram

B. Solution Framework/Proposed Methodology

The research work in this paper attempts to increase the efficiency of the vulnerability detection scheme to increase the accuracy level by proposing a methodology. This methodology includes various steps from the collection of smart contracts to vulnerability analysis and incorporates our proposed algorithm for the analysis of the vulnerability.
The system diagram for our proposed work is shown in Figure 1. The initial step is to gather all of the Ethereum SC and store them in a repository. The second stage is to use the Remix solidity static analysis plugin and Solium to perform vulnerability assessments. The last step is to analyze the result of both tools and evaluate them on the basis of accuracy and effectiveness.

```
1: procedure ANALYSIS (Contract, StaticAnalysisTool)
2: BufferedReader ← contract
3: while BufferedReader ≠ EndofFolder do
4: StaticAnalysisTool ← BufferedReader[contract]
5:Vulnerability←AnalysisResult(tool, contract)
6:contract← contract + 1
7: return detected vulnerability
```

Figure 2: Algorithm 1 Smart Contract Vulnerability Analysis

IV. EXPERIMENTATION

We carried out the experiment for detecting vulnerabilities in smart contracts using the proposed methodology as shown in Figure 1.

A. Collecting Smart Contracts

Ethereum's blockchain is an extremely useful resource for collecting SC. Consequently, because they are kept in bytecode format, analyzing them for vulnerabilities would be difficult. We used EtherScan.io, a website that provides information about Ethereum's blockchain data and smart contracts, in order to acquire smart contract source code for our project. Using this website, developers may publish their source codes and the website verifies that the source code that corresponds to the bytecodes is put on the blockchain so that anybody who wants to engage with a smart contract can examine its source code, and logic, and trust it more easily. However, EtherScan.io does not have a collection of each and every smart contract implemented and deployed on Ethereum's blockchain, it does include a substantial collection of smart contracts in the form of source codes to be studied and analyzed. In addition, we obtained a few contracts from etherscan, as well as a few from other sources, such as github repositories.

Table 1: Smart Contracts used in the Experiment

Contracts	No. of Functions	LOC	Source
Reentrancy.sol	5	42	https://bit.ly/3oLlGDc
KingOfThe	6	170	https://bit.ly/3lqg13a
Reentrancy.sol	5	42	https://bit.ly/3oLlGDc
HoneyPot.sol	4	24	https://bit.ly/3FuzNm9
Auction.sol	3	53	https://bit.ly/3oNREi3
Roulette.sol	1	15	https://bit.ly/3oNbhql
TimedCrowd	2	21	https://bit.ly/3oOtGn0
HYIP.sol	3	20	https://bit.ly/3oKYq8j
EtherGame.sol	3	58	https://bit.ly/2YCNzCw
EtherStore.sol	2	20	https://bit.ly/3ArMWZE
EthTxOrder	2	32	https://bit.ly/3oT4ITv
GuessThe	3	21	https://bit.ly/2YHngv9
GuessThe	3	27	https://bit.ly/3lrwsfF
KingOfThe	6	170	https://bit.ly/3lqg13a
NEW_YEARS_	7	70	https://bit.ly/3ArNsH4
OpenAddress Lottery.sol	7	97	https://bit.ly/3AqoUhl
PredictThe	4	36	https://bit.ly/3lpBFEQ
Private_	5	63	https://bit.ly/301jEnV
Race	2	47	https://bit.ly/3alP2zu
Rubixi.sol	17	136	https://bit.ly/2WVUnua
TimeLock.sol	3	21	https://bit.ly/301jEnV
TxOrigin	4	16	https://bit.ly/3Fw5j3g
TxOrigin	3	11	https://bit.ly/3Fw5j3g
MyContract.sol	2	15	https://bit.ly/2YzhDP8
Return	2	16	https://bit.ly/3FwWAOf
Origin.sol	1	32	https://bit.ly/3FwWAOf
ETPlanV3.sol	23	340	https://bit.ly/3FwWAOf

B. Smart Contract Vulnerability Analysis Tools

Two analysis tools – Remix [32] and solium [33] - examined the source code of the smart contracts. The source code of the smart contracts was written in solidity language:

1) Remix Solidity Static Analysis Plugin:

Remix IDE is an online web-based and desktop program, it is free and open source. Rapid development cycles are encouraged, and a large number of plugins with intuitive GUIs are available in this tool. When it comes to contract development, Remix is the go-to tool. When smart contracts are built and compiled, the Solidity Static Analysis plugin performs static analysis upon those contracts. Among other things, it checks for security flaws, improper development methods, and bad coding practices.

2) Solium:

Solium examines and resolves style and security problems in your Solidity code. Solium does not precisely adhere to the Solidity Style Guide. The behaviors it imposes by default are best practices for the community as a whole. Security is a key consideration when creating blockchain applications. The solidity code must be devoid of security flaws. The main objective of the development of this tool is to fix security concerns. In addition, it checks for vulnerabilities in smart contracts and ensures that the code is structured correctly.

3) Tools Analysis:

In order to analyze which tool performs best among the two we have used two assessment methods: effectiveness and accuracy. We restricted ourselves to 30 contracts due to the time and effort required to analyze the contracts using the assessed tools and check the tool's analysis results. The execution time of both solium and Remix is 25 seconds.

V. RESULTS AND DISCUSSION

The acquired data is evaluated and analyzed in regards to research questions RQ2 and RQ3. To manage both research questions we have carried out an experiment using two new security analysis tools which were not used in any previous research. In order to calculate the effectiveness and accuracy of the tools we will collect the four building-block metrics;

- **True Positive (TP)** = Total count of contracts perfectly detected as vulnerable by the tool
- **False Positive (FP)** = Total count of contracts could not be detected as vulnerable by the tool
- **True Negative (TN)** = Total count of contracts perfectly detected as non-vulnerable by the tool

- **False Negative (FN)** = Total count of contracts not be detected as non-vulnerable by the tool.

The effectiveness of a security analysis tool in terms of detection of vulnerabilities can be measured by Recall.

$$Recall = \left(\frac{TP}{TP+FN}\right) * 100 \quad (1)$$

Where;
TP is True Positive, and
FP is False Positive.

The accuracy score of a tool is measured as follows:

$$Accuracy = \left(\frac{TP+TN}{TP+TN+FP+FN}\right) \quad (2)$$

Where;
TP is True Positive,
TN is True Negative,
FP is False Positive, and
FN is False Negative.

Table 1II: Confusion Matrix of Solium

True Negative 5	False Positive 0
False Negative 5	True Positive 20

Accuracy = TP+TN/(TP+TN+FP+FN)
20+5/20+0+5+5= 0.8333*100=83.3%

Table 1V: Confusion Matrix of Remix

True Negative 5	False Positive 0
False Negative 2	True Positive 23

Accuracy = TP+TN/(TP+TN+FP+FN)
23+5/23+5+0+2= 0.9333*100=93.3%

A. Analysis of Effectiveness

The effectiveness of the tools will be measured by how many smart contract issues they were able to uncover from our data set. As shown in Figure 2, our data set comprised approximately 25 vulnerable smart contracts and the tools that we have chosen for the experiment are Remix with Solidity static analysis plugin and Solium. Firstly, we ran these 25 vulnerable contracts on Solium and got an 80 % recall rate which means that Solium is 80 % effective in terms of vulnerability detection.

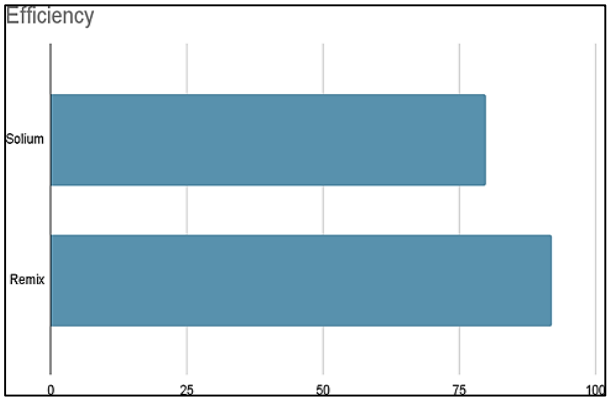

Figure 2: Comparison of the Effectiveness of the Security Analysis Tool (Remix and Solium)

Then we ran our dataset of 25 vulnerable contracts on Remix IDE and enabled a solidity static analysis plugin for the detection of vulnerabilities. On Remix we got a 92 % recall rate which indicates that the Remix plugin is 92 % effective for the detection of vulnerabilities. After analyzing the effectiveness rate of both tools, we can say that Remix is more effective than the Solium tool.

B. Analysis of Accuracy

Accuracy refers to an instrument's capacity to measure a precise value. The degree to which a measurement is accurate relates to how close it is to the correct value. A measurement's uncertainty is an estimate of how much the measurement result might differ from this value. For calculating accuracy, we have taken 25 vulnerable SC and 5 non- vulnerable audited SC because false positive and false negative rates are used to determine accuracy, so we require secure/trusted and tested smart contracts that are bug-free or at least without false positives in order to estimate the false-positive rates.
The accuracy rate of Solium is 83.3 % and Remix is 93.3 % as shown in Fig. 3. This means that Remix is giving more accurate results as compared to Solium.

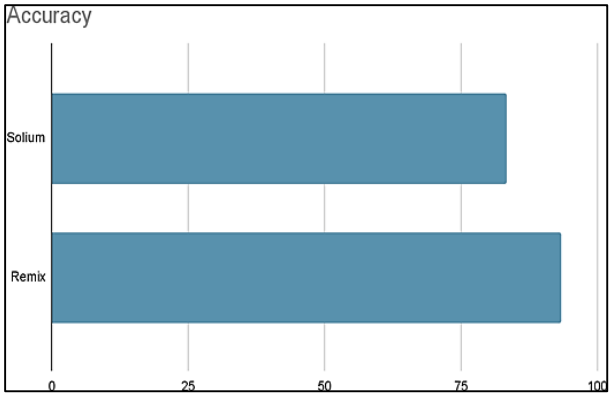

Figure 3: Comparison of the Accuracy of the Security Analysis Tool (Remix and Solium)

Figure 4 shows a comparison regarding different vulnerabilities that have been detected by Solium and Remix. It can be seen here both the tools have a different levels of vulnerability detection against different SC vulnerabilities.

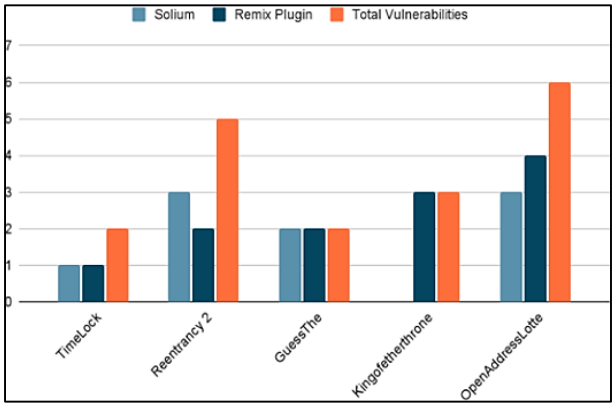

Figure 4: Comparison of the Number of Vulnerabilities Detected by Tools across Smart Contracts

VI. CONCLUSION

Distributed and decentralized applications are using Ethereum SC as digitized agents. Assuring the security of smart contracts will help to prevent needless losses and harmful assaults. Numerous analysis techniques have been built to verify and ensure the accuracy of the SC and their non-vulnerabilities. In this paper, we gave a thorough empirical review of two open-source automatic security analysis tools for detecting security vulnerabilities in Ethereum SC written in Solidity language. We put those tools to the test on 25 real-world smart contracts to see how effective they were at detecting vulnerabilities and how accurate they were at it. The results of our experiment showed that the Remix security testing tool for solidity smart contracts is more effective and accurate statistically than Solium. As far as the detection and mitigation of security vulnerabilities in smart contracts are considered, there is still plenty of room for improvement, such as a greater effort to taxonomize them, automate the test environments once the code analysis tools are out of beta, or cutting the void in research on zero-day vulnerabilities. The necessity of a secure development process should also be understood by developers.

Acknowledgment

The authors would like to thank the NED University of Engineering and Technology, Karachi. Pakistan, for all the support, provided to accomplish this research work

Authors Contributions

In this research work, the authors attempt to make the following contributions to investigating and highlighting potential smart contracts' vulnerabilities;
We have performed empirical analysis to evaluate the two FOSS tools: Remix solidity static analysis plugin and solium on the basis of accuracy and effectiveness.
A methodology for detecting smart contracts vulnerabilities has been proposed with our own designed algorithm that has been applied to show the effectiveness of our proposed approach
A comprehensive comparative analysis of automated tools and frameworks is proposed in supporting the state-of-the-art empirical approach to smart contracts vulnerability detection.

The individual author contributions are as follows; Kashif Mehboob Khan performed the conceptualization and supervision of the paper while Ansha Zahid and Kashif Mehboob Khan jointly carried out the investigation, methodology, and draft preparation.

Conflict of Interest

There is no conflict of interest between all the authors.

Data Availability Statement

The data has been obtained from the github free repository and the links have been provided in the experimentation section where it has been evaluated.

Funding

This research received no external funding.

References

[1] Atzei, N., Bartoletti, M., & Cimoli, T. (2017, April). A survey of attacks on ethereum smart contracts (sok). In *International conference on principles of security and trust* (pp. 164-186). Springer, Berlin, Heidelberg.

[2] Parizi, R. M., & Dehghantanha, A. (2018, June). Smart contract programming languages on blockchains: An empirical evaluation of usability and security. *In International Conference on Blockchain* (pp. 75-91). Springer, Cham.

[3] Epiphaniou, G., Karadimas, P., Ismail, D. K. B., Al-Khateeb, H., Dehghantanha, A., & Choo, K. K. R. (2017). Nonreciprocity compensation combined with turbo codes for secret key generation in vehicular ad hoc social IoT networks. *IEEE Internet of Things Journal*, *5*(4), 2496-2505.

[4] Gao, C. Z., Cheng, Q., He, P., Susilo, W., & Li, J. (2018). Privacy-preserving Naive Bayes classifiers secure against the substitution-then-comparison attack. *Information Sciences*, *444*, 72-88.

[5] Jhaveri, R. H., Patel, N. M., Zhong, Y., & Sangaiah, A. K. (2018). Sensitivity analysis of an attack-pattern discovery based trusted routing scheme for mobile ad-hoc networks in industrial IoT. *IEEE Access*, *6*, 20085-20103.

[6] Conti, M., Dehghantanha, A., Franke, K., & Watson, S. (2018). Internet of Things security and forensics: Challenges and opportunities. *Future Generation Computer Systems*, *78*, 544-546.

[7] Andersen, M. P., Kolb, J., Chen, K., Fierro, G., Culler, D. E., & Popa, R. A. (2017). Wave: A decentralized authorization system for iot via blockchain smart contracts. *University of California at Berkeley, Tech. Rep.* Retrieved from http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-234.html

[8] Christidis, K., & Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, *4*, 2292-2303.

[9] Azmoodeh, A., Dehghantanha, A., & Choo, K. K. R. (2018). Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE transactions on Sustainable Computing*, *4*(1), 88-95.

[10] Chess, B., & McGraw, G. (2004). Static analysis for security. *IEEE security & privacy*, *2*(6), 76-79.

[11] Parizi, R. M., Qian, K., Shahriar, H., Wu, F., & Tao, L. (2018, July). Benchmark requirements for assessing software security vulnerability testing tools. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* (Vol. 1, pp. 825-826). IEEE.

[12] Chen, X., Zhao, S., Qi, J., Jiang, J., Song, H., Wang, C., ... & Cui, H. (2022). Efficient and DoS-resistant Consensus for Permissioned Blockchains. *Performance Evaluation*, *153*, 102244.

[13] Wang, W., Song, J., Xu, G., Li, Y., Wang, H., & Su, C. (2020). Contractward: Automated vulnerability detection models for ethereum smart contracts. *IEEE Transactions on Network Science and Engineering*, *8*(2), 1133-1144.

[14] Ankit. E. (2019). Solidity Static Analysis. Retrieved From: https://github.com/ethereum/remix-ide/blob/master/docs/static_analysis.md

[15] Wood, G. (2014). *Solium: analyzes your Solidity code for style & security issues and fixes them*. Retrieved From: URL https://github.com/iost-official/Solium

[16] Ethereum-Wiki. (2022). Safety. Retrieved From: https://github.com/ethereum/wiki/wiki/Safety

[17] Aldweesh, A., Alharby, M., Mehrnezhad, M., & van Moorsel, A. (2021). The OpBench Ethereum opcode benchmark framework: Design, implementation, validation and experiments. *Performance Evaluation*, *146*, 102168.

[18] Baliga, A. (2017). Understanding blockchain consensus models. *Persistent*, *4*, 1-14.

[19] Kremenova, I., & Gajdos, M. (2019). Decentralized networks: The future internet. *Mobile Networks and Applications*, *24*(6), 2016-2023.

[20] Valenta, M., & Sandner, P. (2017). Comparison of ethereum, hyperledger fabric and corda. *Frankfurt School Blockchain Center*, *8*, 1-8.

[21] Buterin, V. (2014). A next-generation smart contract and decentralized application platform. *white paper*, *3*(37), 2-1.

[22] Parizi, R. M., Dehghantanha, A., Choo, K. K. R., & Singh, A. (2018). Empirical vulnerability analysis of automated smart contracts security testing on blockchains. *arXiv preprint arXiv:1809.02702*.

[23] Wang, H., Li, Y., Lin, S. W., Ma, L., & Liu, Y. (2019, May). VULTRON: catching vulnerable smart contracts once and for all. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)* (pp. 1-4). IEEE.

[24] Liu, Z., Wu, L., Meng, W., Wang, H., & Wang, W. (2021). Accurate Range Query With Privacy Preservation for Outsourced Location-Based Service in IoT. *IEEE Internet of Things Journal*, *8*(18), 14322-14337.

[25] Feist, J., Grieco, G., & Groce, A. (2019, May). Slither: a static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)* (pp. 8-15). IEEE.

[26] Praitheeshan, P., Pan, L., Yu, J., Liu, J., & Doss, R. (2019). Security analysis methods on ethereum smart contract vulnerabilities: a survey. *arXiv preprint arXiv:1908.08605*.

[27] Kalra, S., Goel, S., Dhawan, M., & Sharma, S. (2018, February). Zeus: analyzing safety of smart contracts. In *Ndss* (pp. 1-12).

[28] Luu, L., Chu, D. H., Olickel, H., Saxena, P., & Hobor, A. (2016, October). Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (pp. 254-269).

[29] Dika, A., & Nowostawski, M. (2018, July). Security vulnerabilities in ethereum smart contracts. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (pp. 955-962). IEEE.

[30] Chen, T., Li, X., Luo, X., & Zhang, X. (2017, February). Under-optimized smart contracts devour your money. In *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)* (pp. 442-446). IEEE.

[31] Brent, L., Jurisevic, A., Kong, M., Liu, E., Gauthier, F., Gramoli, V., ... & Scholz, B. (2018). Vandal: A scalable security analysis framework for smart contracts. *arXiv preprint arXiv:1809.03981*.

[32] Albert, E., Gordillo, P., Livshits, B., Rubio, A., & Sergey, I. (2018, October). Ethir: A framework for high-level analysis of ethereum bytecode. In *International symposium on automated technology for verification and analysis* (pp. 513-520). Springer, Cham.

[33] github.com. (2022). ConsenSys/mythril. Retrieved From: https://github.com/ConsenSys/mythril