Evaluation of Contemporary Smart Contract Analysis Tools

Baocheng Wang[1] [a], Shiping Chen[2] [b] and Qin Wang[2]

[1] *Faculty of Engineering, University of Sydney, Australia*

[2] *CSIRO Data61, Australia*

Keywords: Evaluation, Smart Contract, Vulnerability, Solidity, Analysis Tools.

Abstract: Smart contracts are an innovative technology built into Blockchain 2.0 that enables the same program (business logic) to run on multiple nodes for consistent results. Smart contracts are widely used in current Blockchain systems such as Ethereum for different purposes such as transferring cryptocurrencies. However, smart contracts can be vulnerable due to intentional or unintentional injection of bugs, and due to the immutable nature of the Blockchain, any bugs or errors become permanent once published, which can lead to smart contract developers and users suffering from significant economic loss. To avoid such problems, it is necessary to perform vulnerabilities detection to the smart contracts before deployment, and a large number of analysis tools have also emerged to ensure the security. However, the quality of the analysis tools that currently exist on the market varies widely, and there is a lack of systematic quality assessment of these tools. Our research aims to fill this gap by conducting a systematic evaluation of some existing smart contract analysis tools.

1 INTRODUCTION

Blockchain is a distributed database shared among a peer-to-peer network without any centralised node. Ever since the advent of the Bitcoin system in 2009, the underlying Blockchain technology that ensured the system's scalability and security without any centralised organisation's governance had rapidly attracted significant interest, due to its unique characteristics of decentralisation, immutability and tractability, etc (Nakamoto, 2008). The Blockchain technology extracted from Bitcoin was then extended by Ethereum, and began to support the execution of smart contracts, which are Turing-complete programs that run on Blockchains that enable users to establish their own rules for ownership, transaction, and state transitions (Buterin et al., 2014). Because of the trustless, immutable, and transparent features, smart contracts have now been applied in a wide range of fields, and one of the most representative applications is Decentralized Finance (DeFi). The popularity of decentralized financial applications has been tremendously increasing in recent years, and the total value of the digital assets locked in DeFi application grew from US $675 million to over US $40 billion from 2020 to 2021 (Jensen et al., 2021). However, it also motivates attackers to attack the smart contracts that support these applications.

More importantly, smart contracts may be vulnerable due to the bugs injected with or without intentions, and can then lead to significant financial loss for the contract developers and users. One of the most representative cases is the TheDAO reentrancy attack on Ethereum in June 2016, which directly led to a loss of 3.6 million Ethers, worth about US $60 million (Mehar et al., 2019). Parity Wallet Hack in 2017 is another well-known incident, in which a simple vulnerability was found on the Parity Multisig Wallet contract, and then allowed an attacker to steal over 150,000 ETH (worth approximately 30M USD) from it (Walker, 2017). In 2018, the BEC attack exploiting an integer overflow vulnerability in the contract then led to an embezzlement of over 900 million USD (Almakhour et al., 2020).

Furthermore, because of the immutability feature of Blockchain, the smart contract could no longer be updated once deployed, even if the contract is facing a major security crisis. Therefore, it would be vital to ensure that all the stakeholders involved can be confident that the smart contract will be safe and secure after deployment. Auditing is an industrial approach commonly used to establish confidence for all parties, where auditors will thoroughly analyse the smart contracts to uncover some potential vul-

[a] https://orcid.org/0000-0001-7740-719X

[b] https://orcid.org/0000-0002-4603-0024

Wang, B., Chen, S. and Wang, Q.
Evaluation of Contemporary Smart Contract Analysis Tools.
DOI: 10.5220/0011769600003464
In *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2023)*, pages 322-329
ISBN: 978-989-758-647-7; ISSN: 2184-4895
Copyright © 2023 by SCITEPRESS – Science and Technology Publications, Lda. Under CC license (CC BY-NC-ND 4.0)

nerabilities as well as some high-level logic errors that could not be detected by current automatic tools (Perez and Livshits, 2019). Despite the high accuracy and capability, auditing is usually not the preferable choice because it is time consuming and high in cost, the cost of auditing a small or medium-sized projects ranges from 5,000 to 30,000 USD on average (Hacken, 2022). Other than auditing, a large number of smart contract analysis tools have been developed, which can automatically scan through the contract codes to detect security vulnerabilities. However, there is no systematic evaluation on existing smart contract analysis tools, because of the uneven qualities as well as the limited knowledge available regarding the real effectiveness of those tools (Dia et al., 2021).

To fill this gap, we conduct a systematic evaluation of some existing analysis tools by comparing their results of analysing several chosen smart contracts. The main contributions of this work are:

1. Several representative smart contract analysis tools were selected, after investigated over 20 existing analysis tools commonly used on the market.
2. After systematic research, several representative sample smart contract projects were selected to be the benchmark of the experiments.
3. Two experiments were conducted to evaluate the **reliability**, **accuracy**, **performance**, and **robustness** of the selected analysis tools.
4. Some key observations were proposed based on the evaluation outcomes.
5. Some future directions were proposed based on the limitation of this study.

Paper Structure. The paper is organised as follows. In Chapter 2, we will define the scope and methodology of the research. The Chapter 3 will be divided into two sections. In Section 3.1, we will introduce the preparation work done for the experiments, while the experiment data and the evaluation outcomes will be presented in Section 3.2. In Chapter 4, we will draw some conclusions and propose some future directions based on the findings and limitations of this study.

2 METHODOLOGY

In this section, we will define the scope of this paper, and explain the methodology of the research.

2.1 Scope Definition

In this paper, we will only focus on the analysis of Ethereum smart contract source code written in Solidity, so we will not consider smart contracts in EVM Bytecode when selecting smart contracts for benchmarks, and we have to ensure that the Solidity version used in sample contracts is supported by all selected analysis tools.

In addition, this research will only focus on attacks that exploit vulnerabilities in smart contracts, rather than attacks against the network, or the Blockchain, such as DOS attacks, witch attacks, 51% attacks, etc.

Moreover, to ensure the fairness of the experiment, the parameters of the tool will not be fine-tuned in the experiment, so that all the evaluation results will be based on the default settings of the tool. Therefore, it would be reasonable that a tool used in the experiment may perform better after the parameters have been fine-tuned by experienced developers.

2.2 Research Methodology

This research can be divided into three stages, Preparation stage, Experiment stage, and Evaluation stage.

In the Preparation stage, a series of research on some regular or popular attacks against smart contracts will be carried out, and then design two selection criteria to choose a few representative contracts and analysis tools respectively.

In the Experiment Stage, we will conduct two groups of experiment using the analysis tools and sample smart contracts selected in Preparation stage, in order to evaluate the selected analysis tools both qualitatively and quantitatively.

In the Evaluation stage, these analysis tools will be evaluated by comparing the analysis results in those vulnerability reports, and some key observation could be derived from the evaluation outcomes.

2.3 Selection Criteria for Tools

After investigation, at the time of writing this article, there are more than 20 analysis tools frequently used on the market, and we will select some representative tools for experiments (Almakhour et al., 2020).

The tools used for the experiment will be selected based on the following criteria:

1. The tools should be able to analyse smart contracts by taking in Solidity code.
2. The Solidity version supported by the tool should be up to date. For instance, at the time of writing, the latest stable version of Solidity released is

0.8.17, so the latest version supported by the tool should not be lower than 0.5.

3. The analysis tools' projects should have active or large User Ecology.
4. The analytical methods implemented by the tools should be diverse to minimise bias.

2.4 Selection Criteria for Contracts

After investigating several existing open-sourced Decentralized Application (DApp) projects, some representative smart contracts will be selected to be benchmark (sample smart contracts) in the experiments based on the following criteria:

1. Smart Contract programming language: Solidity.
2. The Solidity version of most contracts in the project should be supported by all tools.
3. Diversity in contracts' purposes and features to minimise bias.
4. Trusted third-party analysis or verification results that can be used for reference, such as previous professional audit reports.

2.5 Evaluation Metrics

This study aims to evaluate four metrics of these selected analysis tools through two groups of experiments, including reliability, accuracy, performance, and robustness.

2.5.1 Reliability

The reliability of the analysis tools will be qualitatively evaluated by examining whether these tools can detect some representative smart contract flaws and flag them correctly. It is worth noting that the reliability in this evaluation metric is relative, not absolute, which means that the reliability of tools are not evaluated based on their individual experimental results, but to compare and analyze the similarities and differences in the experimental results among different tools. That is, it is irrational to assert that a tool is unreliable once it fails to detect a well-known vulnerability correctly, and it should be compared with the experimental results of other tools.

2.5.2 Accuracy

The accuracy of these smart contract analysis tools will be evaluated based on two indicators: False Positive Rate (FPR) and False Negative Rate (FNR).
False Positive Rate (FPR). Once the analysis tool reports a vulnerability that does not truly exist in the contract, it will be counted as a false positive, and the false positive rate measures the probability of false alarms. In order to measure the false positive rate of vulnerabilities of the tool, we will first summarise the vulnerabilities reported by all tools in the process of analysing a certain smart contract, including the category of vulnerability and the location of the defective code. Then each flagged code segment in the contract will be manually checked to confirm the existence of the reported vulnerability. If it is confirmed that the marked code segment does not have the reported vulnerability, it will then be considered as a False Positive. For example, if an analysis tool marks line 8 of the sample contract and reports that there is an integer overflow issue, but the marked code segment does not involve arithmetic operation, it could be considered as a False Positive.
False Negative Rate (FNR). The False Negative Rate measures the probability of an analysis tool neglecting an existing vulnerability. To measure the false negative rate of vulnerabilities, we will first aggregate all the vulnerabilities that are proved to exist in a certain smart contract from multiple trusted sources, including third-party audit reports, inspection reports, and some manually confirmed vulnerabilities reported by some analysis tools. The only exception is the SWC Registry, since most of the contracts in the SWC Registry are naive contracts dedicated to demonstrating some specific defects, the contract vulnerability report provided by the Ethereum Community will be used as the only criterion. If one of the confirmed vulnerabilities were not correctly reported by the tool, it will be considered as a False Negative. For example, when we choose to use The DAO contract that was attacked due to reentrancy as a sample for experiment, if the analysis tool fails to report the reentrancy risk, it can be counted as a False Negative.

2.6 Performance

The performance of a smart contract analysis tool will be evaluated based on two indicators, the first one is the average execution time consumed to analyse selected contracts. The other one is the number of time-out execution, out of the consideration of possible timeouts due to deep recursive analysis. We set the maximum time-out parameter to 120 seconds, so the execution will be counted as a time-out, if the execution times exceed the maximum time. It is also worth noting that due to the internal dependencies among the smart contracts in the real-world DApp projects, some smart contracts may not need to be directly executed, but imported in other contracts. Therefore, the number of contract analysis performed in the experi-

Table 1: Summary of the version information of each analysis tool.

	Versions	Solidity Versions	Commits
Mythril	v0.22.42	0.*.*	4728
Slither	v0.8.3	0.*.*	2559
Securify	v2.0	0.5.8 - 0.6.*	171

ment may not be strictly equal to the total number of contracts in the project.

2.6.1 Robustness

If a analysis tool fails to examine a contract due to some technical issues, such as unsupported Solidity version used in the contracts, and unable to compile or cannot resolve some certain symbols, etc, then it will also be counted as a failure, which could be used to evaluate the robustness of a analysis tool when analysing various smart contracts.

3 EVALUATION

The Evaluation could be divided into three steps. First of all, we will prepare for the experiments by selecting the representative tools and sample smart contracts. Then, we will perform the designed experiments and collect data. The final step is to compare the experimental data and make some observations.

3.1 Preparation

Three representative analysis tools were selected based on the criteria mentioned in Chapter 2, which are Mythril, Slither, and Securify. Some key information about these tools, such as the version, Solidity versions supported[1], total commits on the GitHub repositories, will be demonstrated in the Table 1 given on page 4.

In addition, four representative smart contract projects were selected as the benchmarks in experiments based on the criteria mentioned in Chapter 2, which are the Smart Contract Weakness Classification Registry (SWC Registry), Uniswap V2, PancakeSwap, and BTRST contracts. Within the sample smart contract projects, SWC Registry is a collection of sample smart contracts demonstrating specific vulnerabilities, which is initiated by the Ethereum Community and now maintained by Consensys MythX team, and three real-world DeFi projects that have been deployed to Ethereum: Uniswap V2, PancakeSwap, and BTRST Contracts. Some key information about these benchmark contracts, such as the commit id of the project used in experiment, Solidity version, total number of errors recorded by the audit reports and inspection reports, will be demonstrated in the Table 2 given on page 5. The total number of errors aggregates information from multiple sources, including the audit reports, manual checking, and vulnerability reports generated by selected tools, which means once the existence of an error reported by a analysis tool is then confirmed by checking contract source code manually, it will also be added to the total number of errors, although not mentioned in the audit report.

[1]The '*' symbol means all the Solidity versions available are supported by the tool

3.2 Baseline Experiment

The first group of experiments is a baseline experiment that qualitatively evaluates the general reliability by examining whether these tools can detect some representative smart contract flaws and flag them correctly, including the attacked version of The DAO contract with reentrancy vulnerability, the hacked version of the Parity Multi-sig Wallet with unchecked call vulnerability (Palladino, 2017), as well as the attacked version of the BEC Token contract with integer overflow vulnerability (Thanh, 2018). In the baseline experiment, we will only focus on three vulnerabilities that have been exploited by attackers in the history, namely the reentrancy risk of The DAO, the unchecked call vulnerability of Multisig Wallet, and the integer overflow risk in BEC Token. If the target vulnerability is not included in the vulnerability report generated by the tool, or the risk is incorrectly flagged (including the wrong risk name and wrong location marked), it is considered to fail the test. For example, if the Integer Overflow risk is not included in Mythril's report on the BEC Token contract, the qualitative result of this test will be false. The qualitative results are listed below.

Mytrhil. Mythril has successfully detected the reentrancy vulnerability in the attacked The DAO contract, and the use of delegatecall() in the hacked version of the Parity Multisig Wallet contract. However, it failed to detect the integer overflow vulnerability within the attacked BEC Token contract.

Slither. Slither has successfully detected the reentrancy vulnerability in the attacked version of The DAO contract, as well as the unchecked call vulnerability in the hacked version of the Parity Multisig Wallet contract. However, it failed to detect the integer overflow vulnerability within the attacked BEC Token contract.

Securify. Securify failed to execute when analysing The DAO contract and the Parity Multisig Wallet con-

SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 2: Summary of the key information of the benchmarks.

	Commit ID	Solidity Version	#Contracts	#Errors	Inspected
SWC Registry	b014203	0.4.13 - 0.6.4	115	116	✓
Uniswap V2	4dd5906	=0.5.16	12	40	✓
PancakeSwap	3b21430	=0.5.16	13	46	✓
BTRST	ad4fb48	=0.5.16	5	36	✓

tract, because they are using a lower version of Solidity than the minimum version supported by Securify. Though it succeeds in analysing the BEC Token contract, it fails to flag the right code segment in which the integer overflow vulnerability exists.

3.3 Quantitative Experiment

The second one is a quantitative experiment that evaluates the accuracy, performance, and the robustness of these tools. The experimental data of each tool is listed below.

Mythril. After analysing all the sample smart contracts, Mythril has flagged 55 errors in total, where 10 of them are then found to be false positives, from which can deduce that the Overall FPR is 18.18%, and 45 confirmed vulnerabilities were found. According to Table 2, there are 238 confirmed vulnerabilities in total, which then leads to 193 False Negatives in total. With a total of 238 confirmed vulnerabilities, the Overall FNR is 81.09%. It is worth noting that the average execution time of Mythril is 42.61 seconds, which is relatively long, and there are 5 time-outs. The detailed experiment data is organised in the table given below.

Slither. After analysing all the sample smart contracts, Slither has flagged 1062 errors in total, where 46 of them are found to be False Positives, from which can deduce the overall FPR is 4.33%, and it finds 89 reported errors, which then leads to 60 False Negatives in total. With a total of 238 confirmed vulnerabilities, the Overall FNR is 61.38%. Far fewer errors were reported by Slither in the other three real-word DeFi projects. As for the performance metrics, the Average Execution Time of Slither is 0.84 second.

Securify. After analysing all the sample smart contracts, Securify has flagged 787 errors in total, while it failed to examine 25 contracts due to unsupported Solidity versions. After manual inspection, 56 of them were found to be false positives, from which can deduce that the overall FPR is 7.12%, and 59 reported errors were found successfully, which then leads to 57 false negatives. With a total of 238 confirmed vulnerabilities, the Overall FNR is 47.48%. It is also worth mentioning that the frequency of failure occurs during the execution of Securify is relatively high. Generally, the failures are caused by the fact that the Solidity version used in the contract exceeds the upper or lower limit supported by the tool.

Table 3: Overall summary of the Baseline Experiment data.

	Mythril	Slither	Securify
The DAO	✓	✓	/
Parity Multisig Wallet	✓	✓	/
BEC Token	x	x	x

4 PUT IT ALL TOGETHER

In this section, we will aggregate all experimental data and discuss some important observations based on these data. In addition, we will discuss some limitations of this experiment and research methodology and then propose some future directions.

4.1 Overall Evaluation

In order to better evaluate the existing analysis tools, a lateral comparison will be made among the selected tools based on the experimental data collected in Chapter 3, i.e. comparing the same indicators among different tools. To facilitate the presentation of the experimental data collected in Chapter 3, they are summarised in the Table 3 and Table 4 given on page 6, where Table 3[2] is the summary of the Baseline Experiment data, and Table 4 is the Quantitative Experiment data. The detailed experimental data set obtained from the Quantitative Experiment is aggregated into Table 5 on page 8, which summarizes in detail the data obtained by each tool in analyzing specific contract items.

4.2 Key Observation

In this section, we will discuss some of the observations that can be made from the experimental data

[2] In Table 3, there are 3 types of symbols used to indicate the qualitative results of the Baseline Experiment.
"✓" means that the vulnerable code is correctly flagged
"x" means that the tool failed to flag the vulnerable code
"/" means that the tool failed to execute due to some technical issues, e.g. unsupported Solidity versions.

Table 4: Overall summary of the experiment data.

	Mythril	Slither	Securify
Number of Errors	55	1062	787
False Positive	10	46	56
False Negative	193	89	113
False Positive Rate	18.18%	4.33%	7.12%
False Negative Rate	81.09%	61.38%	47.48%
Average Execution Time	42.61 s	0.84 s	2.12 s
Time-out	16	0	0
Failures	2	1	25

listed above, including some features or issues that are prevalent in the selected analysis tools, as well as some characteristics of a particular tool compared to others.

The first key observation based on the Baseline Experiment data demonstrated in the Table 5 is the lack of reliability of analysis tools when they are attempting to detect vulnerabilities with high complexity or hidden in depth.

As shown in the Figure 1 given on page 6, it can be observed that all the selected tools are high in False Negative Rate, ranging from 47.48% to 81.09%, while the False Positive Rate of these tools is relatively lower, ranging from 4.33% to 18.18%.

The high FNR can be explained by the limitations on the types of defects the tools can detect. Because the vulnerabilities in the sample contracts might be divided into multiple categories, and the types of vulnerabilities that a analysis tool can detect are limited. For example, the Mythril version used in this study can only detect 11 categories of defects, while there are 37 different kinds of defects summarised in the SWC Registry, which means that there are at least 26 kinds of defects that are undetectable for Mythril. It is worth noting that only 35 kinds of defects were actually verified in the experiment, because the SWC Registry does not provide any sample smart contracts for the other two defect types.

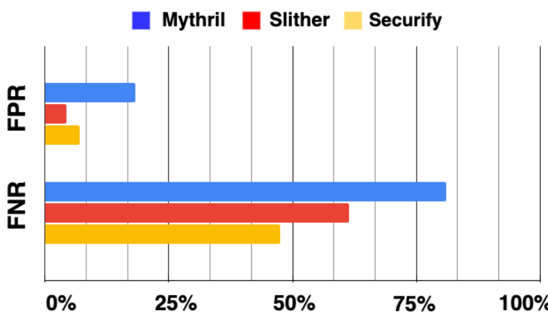

Figure 1: Bar chart that compares the FPR and FNR among the tools.

Mythril. According to the experimental data, it can be observed that Mythril has the least ideal accuracy because it has the highest false positive rate and false negative rate. However, according to the Figure 2 given on page 6, Mythril has the least number of false positives. This phenomenon could be explained by the fact that Mythril has much fewer detectors than Slither and Securify, which means fewer categories of vulnerabilities could be detected by Mythril, and then result in the least total number of vulnerabilities reported compared with other analysis tools.

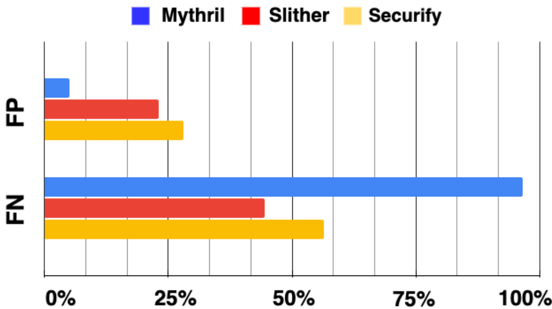

Figure 2: Bar chart that compares the FP and FN among the tools.

Moreover, the average execution time Mythril consumes to analyse all the sample smart contracts is much longer compared to Slither and Securify. The long execution time is often caused by two factors, the most common one is the differences in the operating system on which the analysis tool is running. However, in this study, the three tools are run on the same system, so the difference of the system cannot explain this observation. Another factor is the inappropriate parameter setting. According to the Issue section in the Mythril GitHub community, users are allowed to specify the maximum depth and transaction times via command line options, which can directly determine the overall complexity of analysis and then affect the execution time. As the default maximum depth of Mythril analysis is 22 and the number of transactions is 2. Therefore, the execution time could be reduced by fine-tuning the parameters of transaction time and maximum analysis depth.

In addition, the usability of Mythril is the best among the three analysis tools, because it supports the most detailed vulnerability reports, including the description of the vulnerability, location of the vulnerable code segment, estimated gas usage, initial state, and transaction sequence, etc.

Slither. According to the Table 4 given on page 6 and Figure 1 given on page 6, Slither reports the most vulnerabilities in total after analysing all the sample smart contracts, and has shown a relatively high accuracy due to the least false positive rate and false negative rate. Moreover, Slither shows the most ideal per-

formance in the experiment, because its average execution time is only 0.8s, which is the lowest among the three tools.

However, it is worth mentioning that Slither defines the vulnerabilities more broadly, and has pre-defined a large number of vulnerabilities with extremely low severity that appear frequently in the smart contracts. It categorises the severity of the vulnerabilities into five levels, including High, Medium, Low, Informational, and Optimisation, whereas Mythril has only categorised three levels, including High, Medium, and Low. The Informational or Optimisation levels vulnerabilities usually do not pose a threat to security, but cannot be refuted, for example, the variable names that violate the Solidity naming convention would be reported as an Informational level vulnerability. Over 50% of the vulnerabilities reported by Slither have Informational and Optimisation levels of severity. This phenomenon may directly lead to a falsely low false positive rate, so it might make the experiment fairer, if these types of vulnerabilities reported could be eliminated before evaluation.

Securify. According to the experimental data in the table, it could be observed that Securify has the least ideal robustness, as it has failed to analyse 25 smart contracts for a variety of reasons, such as the unsupported Solidity version used in the smart contracts, or some unrecognised tokens by the tool, etc. However, the performance of Securify is relatively high, as its average execution time is around 2.1 seconds. Securify has also shown a relatively high accuracy due to its least false negative rate, which could be explained by the fact that Securify supports the detections of the most types of vulnerabilities when compared with Mythril and Slither.

Similar to Slither, Securify has categorised the severity of the vulnerabilities it could detect into five levels, including Critical, High, Medium, Low, and Info. The majority of the vulnerabilities reported by Securify are in the Low level or Info level, which may also cause the falsely low false positive rate.

4.3 Future Directions

In this section, we will propose three future directions based on some key observations of this research or intentions to address some limitations of the study.

1. For the future studies that attempt to optimise the accuracy of the analysis tools, reducing the false negative rate might be a more valuable direction, as it has a higher marginal benefit, which could be observed from the experimental data that the false negative rate of the current analysis tools is generally much higher than the false positive rate. However, in contrast, reducing FPR is more friendly to beginners and saves more time, because developers only need to optimise existing detectors, while reducing false negative rate might require much more workload in developing new detectors.
2. As mentioned in the previous section, a limitation of this study was identified from the Slither and Securify evaluation process that the severity factor was neglected when conducting the experiments and might have resulted in the falsely high false positive rate estimation. Therefore, in future experiments, it would be better if the reported vulnerabilities reported can be classified according to their severity, and compute the proportion of vulnerabilities of each severity in the total number. It could help avoid some flooding data affecting the fairness of the experiment by setting a threshold of vulnerability severity based on the computed proportion.
3. Due to the time constraint of this study, the total number of sample smart contracts used for experiments is kept below 200, and this limitation might reduce the generalizability of the study among various analysis tools and increase the margin of error, because the characteristics of some analysis tools cannot be reflected when analysing the sample smart contracts.

   Therefore, the most straightforward direction to address this limitation in the future study is to introduce more sample smart contracts in the experiments. Moreover, in order to reduce the bias in the evaluation, these smart contracts could be diverse in application categories, such as DeFi, Decentralized Exchange (DEX), Gaming, etc.

5 CONCLUSIONS

In general, the existing smart contract analysis tools can indeed effectively detect some certain categories of vulnerabilities in the smart contract, but may lack reliability when attempting to detect some more complicated vulnerabilities or defects hidden in-depth, such as the integer overflow defect in the BEC Token contract. More importantly, despite the advantages of time efficiency and low cost, smart contracts analysis tools could not fully replace the manual auditing performed by professional audit teams, because the analysis tools can only detect a certain number of vulnerabilities and defects via some predefined logics or processes, which often covers only a small part of the

Table 5: Detailed Quantitative Experiment data of all tools.

	SWC Registry	Uniswap V2	PancakeSwap	BTRST	**Overall**
			Mythril		
Number of Errors	46	3	6	0	55
False Positive	9	0	1	0	10
False Negative	79	37	41	36	193
False Positive Rate	19.6%	0%	0%	0%	18.18%
False Negative Rate	68.1%	92.50%	89.13%	100.00%	81.09%
Average Execution Time	41.3 s	24.0 s	46.2 s	120 s	42.6 s
Time-out	4	3	5	4	16
Failures	0	1	0	1	2
			Slither		
Number of Errors	845	68	73	76	1062
False Positive	20	8	10	8	46
False Negative	60	13	7	9	89
False Positive Rate	2.40%	11.76%	13.70%	10.53%	4.33%
False Negative Rate	51.70%	32.50%	15.22%	25.00%	61.38%
Average Execution Time	0.6 s	1.4 s	2.1 s	1.9 s	0.84 s
Time-out	0	0	0	0	0
Failures	0	0	0	1	1
			Securify		
Number of Errors	845	68	73	76	1062
False Positive	20	8	10	8	46
False Negative	60	13	7	9	89
False Positive Rate	2.40%	11.76%	13.70%	10.53%	4.33%
False Negative Rate	51.70%	32.50%	15.22%	25.00%	61.38%
Average Execution Time	0.6 s	1.4 s	2.1 s	1.9 s	0.84 s
Time-out	0	0	0	0	0
Failures	0	0	0	1	1

set of all vulnerabilities which might be exploited to breach the security in smart contracts.

REFERENCES

Almakhour, M., Sliman, L., Samhat, A. E., and Mellouk, A. (2020). Verification of smart contracts: A survey. *Pervasive and Mobile Computing*, 67:101227.

Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1.

Dia, B., Ivaki, N., and Laranjeiro, N. (2021). An empirical evaluation of the effectiveness of smart contract verification tools. In *2021 IEEE 26th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 17–26. IEEE.

Hacken (2022). Ethereum smart contract audit - how much does it cost to audit an ethereum smart contract? [online] Available at: https://hacken.io/services/blockchain-security/ethereum-smart-contract-audit/ Last accessed on October 17, 2022.

Jensen, J. R., von Wachter, V., and Ross, O. (2021). An introduction to decentralized finance (defi). *Complex Systems Informatics and Modeling Quarterly*, (26):46–54.

Mehar, M. I., Shier, C. L., Giambattista, A., Gong, E., Fletcher, G., Sanayhie, R., Kim, H. M., and Laskowski, M. (2019). Understanding a revolutionary and flawed grand experiment in blockchain: the dao attack. *Journal of Cases on Information Technology (JCIT)*, 21(1):19–32.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260.

Palladino, S. (2017). The parity wallet hack explained. *OpenZeppelin blog, https://blog. openzeppelin. com/on-the-parity-wallet-multisig-hack-405a8c12e8f7*.

Perez, D. and Livshits, B. (2019). Smart contract vulnerabilities: Does anyone care? *arXiv preprint arXiv:1902.06710*, pages 1–15.

Thanh, L. Y. (2018). Prevent integer overflow in ethereum smart contracts. [online] Available at: https://yenthanh.medium.com/prevent-integer-overflow-in-ethereum-smart-contracts-a7c84c30de66 Last accessed on June 30, 2022.

Walker, J. (2017). Lost in the ether: Parity still scratching its head over multi-sig issue. [online] Available at: https://portswigger.net/daily-swig/lost-in-the-ether-parity-still-scratching-its-head-over-multi-sig-issue Last accessed on July 1, 2022.