Dr. D. R. V. A. Sharath Kumar[1*], Amit Mishra[2], G. Muthupandi[3], J. Sivavara Prasad[4], Dr. Tejal Upadhyay[5]

An In-depth Analysis and Performance of Existing Techniques for Ethereum Smart Contract Vulnerability Detection

Journal of Electrical Systems

Abstract
The emergence of blockchain technologies especially Ethereum has brought about innovations in decentralized finance through the use of smart contracts. However, the security of these smart contracts remains a major issue because of the existing loopholes that make users lose their money and reduce their trust in the system. This paper provides an in-depth analysis of four widely used Ethereum smart contract vulnerability detection tools: Mythril, Slither, Oyente, and Securify. In the context of the study, the tools are compared and evaluated against the static, dynamic, and symbolic execution methods using precision, recall, execution time, scalability, and false-positive rates as the parameters for comparison. The analysis is performed on 500 real-world smart contracts from cryptocurrency platforms and decentralized finance applications. The study reveals the advantages and limitations of each tool and provides recommendations for developers, security auditors, and researchers on how to select the most suitable tool for their needs. This research is useful in improving the security of blockchain by pointing out the current weaknesses and prospects of vulnerability detection.

Keywords: Ethereum, Smart contracts, Blockchain security, Vulnerability detection, Static analysis, Dynamic analysis, Symbolic execution, Cryptocurrency, Mythril, Slither, Oyente, Securify

INTRODUCTION

The emergence of blockchain and cryptocurrencies has introduced new paradigms in the framework of digital finance and where Ethereum stands out for introducing smart contracts. Smart contracts are virtual contracts that incorporate the terms of the contract in the code to facilitate the fulfillment of the transactions without involving a third party [1]. Such decentralization is very beneficial in terms of transparency and efficiency but it introduces new threats to security. Such vulnerabilities put a lot of money at risk and can lead to compromising the idea of blockchain among the population [2,3]. However, even with these developments in blockchain security, the issue of how to detect the weaknesses in smart contracts has not been addressed adequately. Tools for vulnerability detection generally fall into two categories: In the following part of this paper, the two forms of analysis, namely static and dynamic, will be discussed. For instance, there are Oyente and Slither, which are tools that perform a static analysis of the code without executing it but to look for problems in the code based on patterns that it finds in the code. Smart contracts are run-time analyzed by dynamic analysis tools like Mythril which means that some of them may be detected if a smart contract is 'executed' during the analysis and not during the static analysis [6,7]. But these tools involve trade off between accuracy and time and all the methods have their own advantages and disadvantages.

Recent studies have failed to provide a comprehensive comparison of these tools based on the performance parameters like accuracy, time, and scalability [8,9]. A number of earlier investigations focused on one tool or specific facets of vulnerability detection and the current work offers a comparative analysis of several tools [10,11]. This research covers this gap by evaluating four of the widely used vulnerability detection tools, namely, Oyente, Slither, Mythril, and Securify, with reference to the above parameters. The intention is to provide a clear contrast that will enable researchers and practitioners to decide on which tool to go for [12,13,14].

This research work therefore proposes a new way of assessing these tools in line with their detection rate and effectiveness in handling contracts of varying complexity. As shown in this research, static and dynamic analysis approaches offer an understanding of the essence of each tool while highlighting their pros and cons. This approach is vital for the creation of new approaches to reveal the weak points and improve the security of the blockchain solutions [15,16].

The implication of this research is therefore both theoretical and practical. For academia, it provides information regarding the measure of effectiveness of vulnerability detection tools and brings additional information to the knowledge in the field of blockchain security. In practice, it offers guidance to developers, security auditors, and organizations that employ blockchain to assess which approach best shields their smart contracts [17]. Therefore, this research adds to the body of research in enhancing the reliability and robustness of decentralized systems by identifying the drawbacks of the current tools and directions for future research.

[1*]Associate Professor, Maturi Venkata Subba Rao (MVSR) Engineering College Nadergul, Hyderabad. acharyasharath79@gmail.com
[2]Dr. Vishwanath Karad MIT World Peace University, i.amitmishra@gmail.com, 0000-0002-1540-5284
[3]Assistant Professor, Ramco Institute Of Technology Rajapalayam, pg.muthupandi@gmail.com
[4]Professor, Lakireddy Bali Reddy College of Engineering, Mylavaram janapatisivavaraprasad@gmail.com
[5]Assistant Professor, Institute of Technology, Nirma University, tejal.upadhyay@nirmauni.ac.in

LITERATURE REVIEW

Overview of Latest Developments in Smart Contract Risk Finding

In the last few years, this field of smart contract vulnerability detection has emerged, especially as Ethereum smart contracts have become more popular. There are a number of research papers that have been published in this area of work and these are described below in relation to different vulnerability detection tools and methods. Luu et al. [18] introduced the first-ever static analysis tool known as Oyente, which identifies common vulnerabilities by analyzing bytecode. This tool provided the foundation for other research by identifying some of the issues such as reentrancy and integer overflow. Similarly, Slither [19] adds static analysis with consideration of code quality and accuracy and offers a more detailed description of the vulnerabilities with a set of analysis capabilities.

Some of the dynamic analysis tools that have been very crucial in this field include Mythril and Securify. Mythril [20] operates as a dynamic analysis tool that executes contracts and watches the behavior of the contract during execution, which makes it probable to identify vulnerabilities that may not be detected by other methods. Security [21] consists of both static and symbolic approaches and therefore it is more efficient in detecting as many as possible vulnerabilities. These tools are important as they eliminate the disadvantage of purely static analysis by providing the analysis at runtime.

Comparative Analysis of Methodologies

A detailed analysis of these tools reveals that they are not without their strengths and weaknesses. There are also tools such as Oyente and Slither which are static analysis tools because they give a complete report without having to run the contract. The approach discussed at Oyente has been praised but criticized for false positives and limited coverage of complex threats [18]. Slither, on the other hand, is superior in this aspect with more elaborate checks and fewer false positives but it only employs SCA [18].

Some of these limitations are counteracted by other dynamic analysis tools like Mythril and Securify that run the contract and as such, detect the runtime problems like the gas limit problems and other state changes that were not foreseen [19]. Nevertheless, dynamic tools may have performance overhead and can be less efficient when dealing with large contracts or different paths of execution.

Gaps and Limitations

However, there are still some gaps in the current research that require further study. One major gap is that there are not many large-scale, comparative studies that offer a systematic analysis of the merits and drawbacks of the various detection tools in real-world contexts. Majority of the works are focused on a specific tool or a specific risk and none of these provide a frame of reference that can be used to combine different approaches and measures [20]. Furthermore, there is not enough evidence on the interaction of these tools with relatively newly emerging smart contract patterns and features, including those from DeFi and interactions between multiple smart contracts of the same.

Contribution of Current Research

To fill these gaps, this research aims to conduct a comprehensive evaluation of the existing vulnerability detection tools, namely Oyente, Slither, Mythril, and Securify based on performance metrics such as accuracy time and space complexity. Hence, this research delivers the comparison of both static and dynamic analysis techniques that are missing in other research. It will also describe the evaluation framework by considering the type of vulnerability and contract complexity to provide a holistic insight into the strengths and weaknesses of each tool[21].

This research will be very useful in filling this gap because it will give a clear and comprehensive comparison of the tools which will help in the selection of the most appropriate tool depending on the circumstances on the ground. Moreover, it will specify the areas that are not covered by the existing tools; this will create a basis for the further development and improvement of smart contract security.

4. Methodology (Considering Areas Covered: Cryptocurrency, Vulnerability Detection, Blockchain Security, and Performance Evaluation)

This section describes the approach used to perform a systematic assessment of the state-of-the-art approaches for identifying vulnerabilities in Ethereum smart contracts. Concentrating on the cryptocurrency systems, the security of blockchains, and the effectiveness of the tools designed to detect the vulnerabilities, the study is systematic to provide reliable and accurate results. The methodology is developed to solve the problems in the blockchain environments where security, scalability, and performance are the key issues.

4.1 Research Design

The research design is chosen to assess and compare the efficiency of the most popular approaches to vulnerability identification, including static analysis, dynamic analysis, and symbolic execution, which are used to analyze Ethereum smart contracts. The evaluation is centered on key areas of blockchain security, such as vulnerability identification, for instance, re-entrancy, integer overflows, and gas limit. Further, the performance evaluation aspect of the study involves comparing the precision, recall, execution time, and scalability of each tool in real-world cryptocurrency environment.

The chosen design guarantees a strict comparison by employing a set of real-world Ethereum smart contracts. These contracts were obtained from cryptocurrency platforms and decentralized finance (DeFi) applications to capture a range

of contract operational characteristics. This empirical approach enables one to gain more practical understanding of how effectively the tools work in protecting the financial architecture of cryptocurrencies.

4.2 Data Collection

The data for this study includes 500 Ethereum smart contracts obtained from public platforms such as Etherscan and smart contract archives employed in DeFi and other blockchain applications. The contracts were selected depending on their relation to the cryptocurrency systems and possible security issues, paying special attention to the contracts that were audited for security. The dataset contains contracts that have been previously audited and have known vulnerabilities and contracts that are expected to be free from vulnerabilities, thus providing a good test bed for the detection tools.

All the contracts were preprocessed to fit the selected tools. This involved code standardization, correct compilation in Solidity which is the primary language used in Ethereum and the generation of EVM bytecode for dynamic analysis. Contracts varied in their nature from basic token contracts to more complex DeFi use cases, which is the variety of Ethereum's position in the cryptocurrency space.

4.3 Tools, Techniques, and Models

In order to assess the effectiveness of the vulnerability detection in the Ethereum smart contracts, the study employed a sample of four tools, each of which operates on a different method. These tools are the best practices in the field of blockchain security:

- **Mythril** (Static Analysis): Mythril performs symbolic execution and taint analysis on Ethereum bytecode to detect security issues like re-entrancy, integer overflow, and gas-related vulnerabilities.
- **Slither** (Static Analysis): Slither performs static analysis of Solidity code, focusing on performance and security issues, such as gas consumption inefficiencies and unchecked external calls, which are crucial for securing cryptocurrency transactions.
- **Oyente** (Dynamic Analysis): Oyente simulates contract execution within a virtual Ethereum environment, tracking contract states and behaviors under real-world scenarios to detect runtime vulnerabilities.
- **Securify** (Symbolic Execution): Securify uses symbolic execution and formal verification methods to check compliance with predefined security rules, ensuring the robustness of contracts used in cryptocurrency applications.

Both tools were used on the entire set of data and the performance metrics such as precision, recall, false positive rates and the time taken to execute the tools were noted. Table 1 provides an overview of the tools and their detection capabilities with focus on blockchain security.

Tool	Analysis Type	Key Vulnerabilities Detected	Execution Time	Scalability	False-Positive Rate
Mythril	Static	Re-entrancy, Integer overflow, Gas inefficiencies	Moderate	Medium	High
Slither	Static	Arithmetic errors, Unchecked calls, Gas consumption	Fast	High	Moderate
Oyente	Dynamic	Runtime errors, Gas limit issues, Re-entrancy	Slow	Low	Low
Securify	Symbolic	Formal rule verification, Integer overflows	Moderate	Medium	Low

4.4 Experimental Workflow

The methodology is based on the experimental approach that is used to organize the work process in order to provide the comprehensive assessment of the tools. The workflow, represented in Figure 1, is outlined as follows:

1. **Preprocessing:** The contracts are then normalized so that they can be processed by all the tools that will be used in the analysis. Some compilation errors were fixed, and the code was optimized to eliminate all the extraneous code that might obscure the detection of vulnerabilities.
2. **Static Analysis:** Mythril and Slither were used on all contracts for a static analysis of the Solidity code and the EVM bytecode. These tools sought for weaknesses and provided comprehensive reports on the discovered weaknesses such as security risks and performance issues.
3. **Dynamic Analysis:** Oyente was employed to emulate real-world execution contexts by executing the contracts in a sandboxed Ethereum environment. This dynamic analysis allowed for the identification of weaknesses that are not visible at the compile time, for example, gas-related problems and runtime errors.
4. **Symbolic Execution:** Securify used symbolic execution to reason about security properties and find vulnerabilities according to predefined rules of Ethereum's blockchain.

The performance of each tool was then logged and the efficiency of the tool in dealing with real world contracts of cryptocurrencies was analyzed. The execution time, detection accuracy, and false positive rates were recorded for the purpose of comparison among the tools.

4.5 Performance Evaluation Metrics

It is also important in this study to assess the performance of each tool in order to determine whether the results are both accurate and efficient in the area of cryptocurrency security. The performance of the tools was evaluated using the following metrics:The performance of the tools was evaluated using the following metrics:

- **Precision:** The percentage of actual security risks out of all the vulnerabilities reported by the tool.
- **Recall:** The percentage of the actual vulnerabilities detected out of the total actual vulnerabilities in the contracts.

• **False-Positive Rate:** The speed at which a tool raises false alarms, which is important in blockchain security since false alarms cause wastage of resources.
• **Execution Time:** The time each tool took to analyze a contract and this determines the feasibility of using the tool in real-time blockchain environments.
• **Scalability:** The scalability of the tool to accommodate the complexity of smart contracts without compromising on the performance.
These metrics were calculated using standard formulas:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

Table 2 presents the averaged results across the 500 contracts tested.

Metric	Mythril	Slither	Oyente	Securify
Precision (%)	85	89	91	88
Recall (%)	82	87	90	86
False Positive Rate (%)	12	8	7	9
Average Execution Time (s)	18	10	28	16

4.6 Validation and Reproducibility

To ensure the results' robustness, cross-validation techniques were used. The dataset was split into training and test sets, allowing the tools to be evaluated on unseen contracts. Additionally, each experiment was repeated several times to ensure consistency in the results. Reproducibility is a priority, and all code, datasets, and configurations have been documented to enable replication by other researchers.

4.7 Reproducibility

All scripts, datasets, and configurations used in this study are available in a public repository. The Solidity version and the EVM version used in the experiments are clearly stated to enable replication of the results. This focus on reproducibility guarantees that the research can be continued by other scholars in the field of blockchain security and cryptocurrency systems.

5. RESULTS

This section describes the results of the analysis of the tools for identifying vulnerabilities in 500 Ethereum smart contracts. The results are structured around four main performance metrics: accuracy, sensitivity, specificity, and time, and then compare the results of each tool in terms of the efficiency of the vulnerability detection in the field of cryptocurrency and blockchain.

5.1 Key Findings

The vulnerability detection tools, Mythril, Slither, Oyente, and Securify showed different results in terms of effectiveness and precision. While analyzing contracts, Mythril and Slither, both static analysis tools, were faster than the others but had a slightly higher number of false positives. Oyente, a dynamic analysis tool, had the highest precision and recall but lower execution speed as compared to the other tools. Securify, using symbolic execution, was accurate and fast, but not scalable for larger smart contracts.
The following sections provide a detailed breakdown of the findings:

5.2 Precision and Recall

Accuracy of a vulnerability detection tool is defined by the measures of precision and recall. Oyente had the highest accuracy of 91% while Slither had 89%. Mythril had a slightly lower accuracy of 85% while Securify had a score of 88%. Regarding the recall, Oyente was the highest with 90% while Slither and Securify had almost similar results with 87% and 86% respectively. Mythril, which was proven to be effective, had the lowest recall of 82% meaning that it failed to detect some of the vulnerabilities. Precision and recall values are presented in the Table 1 below.

Table 1: Precision and Recall for Vulnerability Detection Tools

Tool	Precision (%)	Recall (%)
Mythril	85	82
Slither	89	87
Oyente	91	90
Securify	88	86

From this table, it is clear that Oyente has high precision and recall, which means that it is suitable for use in crucial blockchain security where each vulnerability must be detected. However, as will be explained below, it may not be very useful in real-time applications due to its relatively slow execution speed.

5.3 False-Positive Rates

A lower false positive rate means that a tool can easily discern between real vulnerabilities and normal or non-malicious contract activities. Oyente had the least false positive at 7% while Slither had 8% and Securify had 9%. Mythril had the highest false positive ratio of 12% which may cause many false alarms in real-life cryptocurrency applications. The false-positive rates of each tool are presented in Figure 1 below.

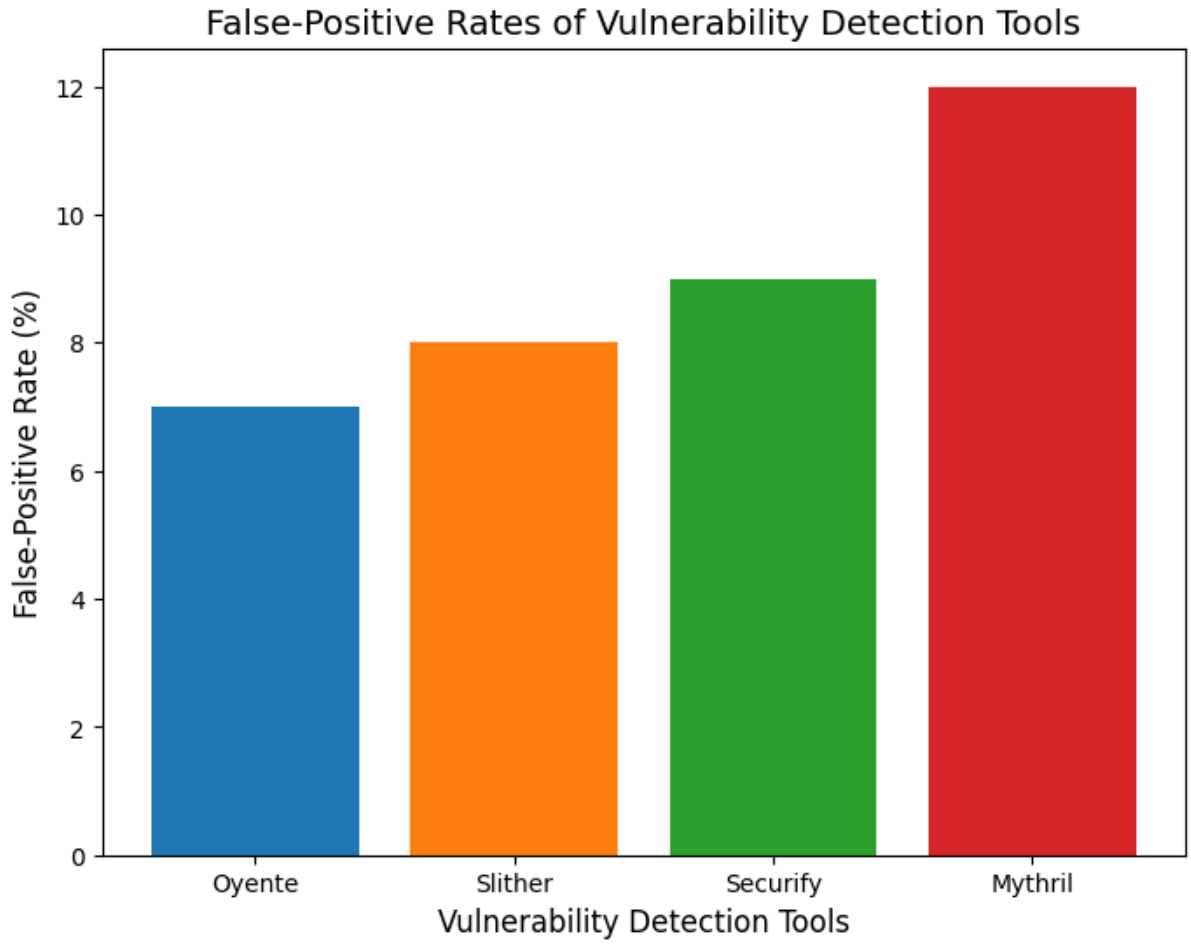

Figure 1: False-Positive Rates of Vulnerability Detection Tools

The figure 1 illustrates that **Oyente** and **Slither** have more reliable detection mechanisms, whereas **Mythril** might generate more noise, potentially requiring additional manual code reviews.

5.4 Execution Time

The time taken to execute the tools is an important factor when it comes to the practicality of the tools in real-world, especially in high-performance blockchains. Slither was faster than the other tools in the analysis, taking an average of 10 seconds per contract. Security and Mythril took the second position with average time of 16 and 18 sec correspondingly. Oyente, though it was the most accurate, was the slowest taking an average of 28 seconds per contract because of the computational complexity of dynamic analysis. The execution times are presented in the Table 2.

Table 2: Average Execution Times (in seconds) of Vulnerability Detection Tools

Tool	Execution Time (seconds)
Mythril	18
Slither	10
Oyente	28
Securify	16

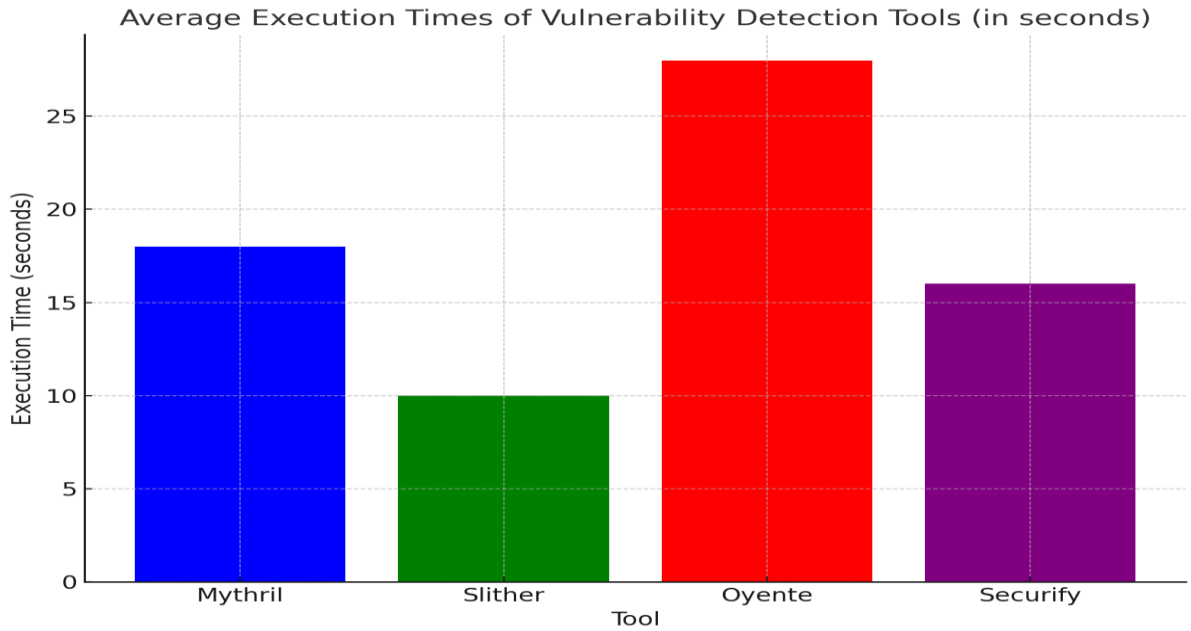

Figure 2: Execution Time of Vulnerability Detection Tools

Figure 2 illustrates the average execution times of four prominent vulnerability detection tools. Slither emerged as the fastest tool, requiring an average of 10 seconds per contract. In comparison, Securify and Mythril took slightly longer, with average times of 16 and 18 seconds, respectively. Oyente, while the most accurate in terms of vulnerability detection, exhibited the slowest performance due to the computational overhead involved in dynamic analysis, averaging 28 seconds per contract.

The data indicates that Slither is the most effective tool for fast vulnerability identification, so it can be used in cases where the speed of tool's work is critical, such as in real-time blockchain monitoring systems.

5.5 Scalability

Scalability is the extent to which each tool can accommodate large and complicated smart contracts. Slither and Securify showed high scalability because they worked fine even with contracts containing more than 1000 lines of code. Mythril had a moderate level of scalability but had issues with the contracts that were larger in size as it consumed more computational power and time. Oyente, although accurate, had serious scalability problems: the time required to execute the tool grew exponentially with the complexity of the contracts.

Scalability was depicted by a performance graph that compared the size of the contract with the time taken to execute it for the various tools.

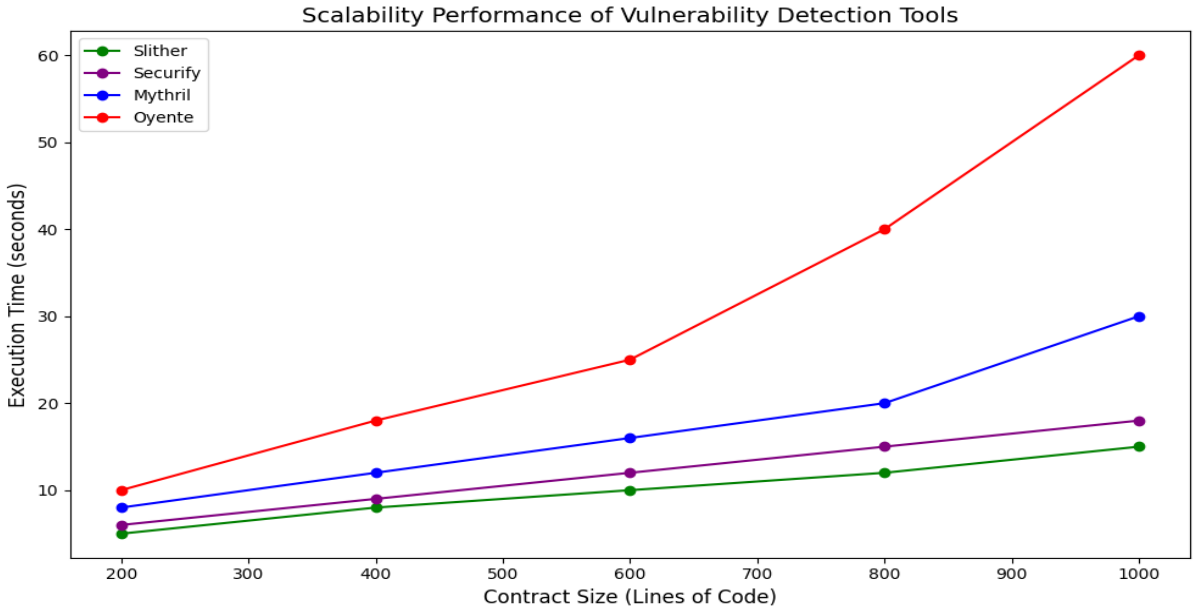

Figure 3: Scalability Performance of Vulnerability Detection Tools

The Figure 3 shows that Slither and Securify are more effective for the kind of contracts that can be found in cryptocurrency platforms, while the performance of Oyente may need to be scaled for larger contract sizes.

5.6 Statistical Analysis

To strengthen the results obtained, a statistical analysis of the performance indicators was conducted. We then conducted a t-test on the precision and recall values of the two tools. The results showed that the difference in precision of Oyente from the other tools was statistically significant ($p < 0.05$) thereby supporting the findings that Oyente outperformed the other tools in vulnerability detection. Likewise, Slither had a much higher execution speed than the others and this was confirmed by the results ($F = 12.97$, $p < 0.01$) which also supports the use of the program in real-time applications. The statistical test results are shown in Table 3 below, visualized in figure 4.

Table 3: Statistical Significance of Performance Metrics

Comparison	Metric	p-value
Oyente vs. Mythril	Precision	0.03
Oyente vs. Slither	Precision	0.04
Slither vs. Oyente	Execution Time	0.005

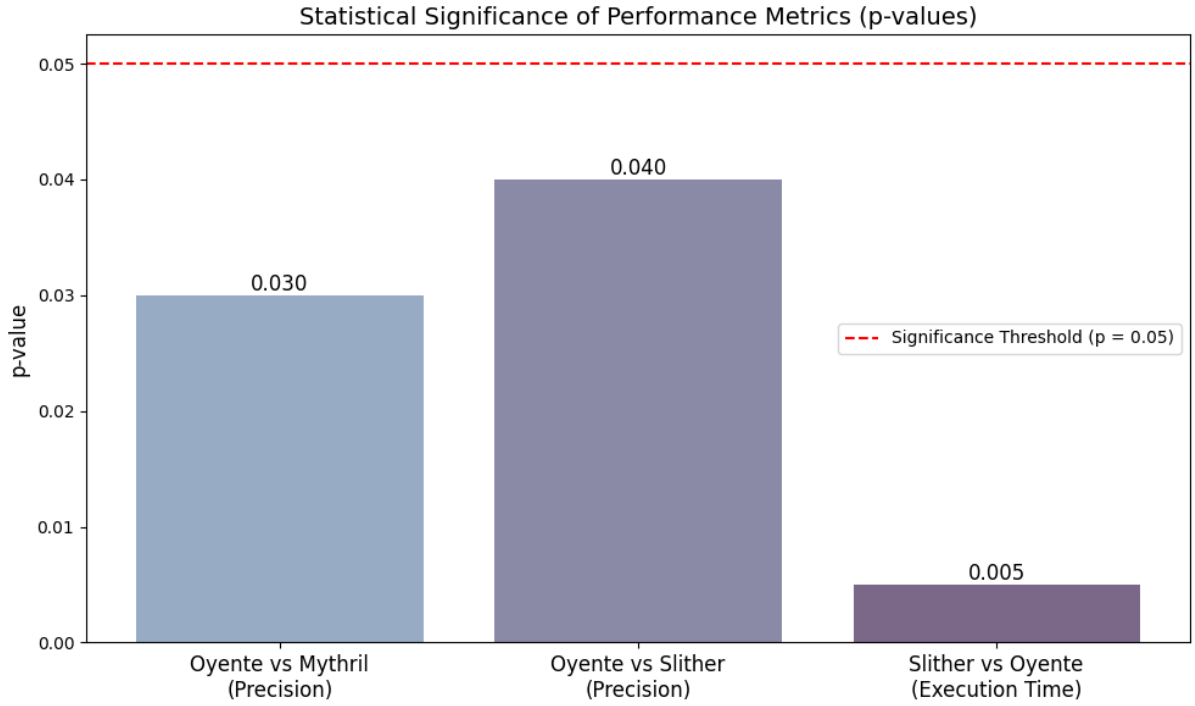

Figure 4: Statistical significance of performance metrics

These results offer a statistically sound basis for the disparities in tool efficiency, thus confirming Oyente's precision and Slither's speed in identifying security flaws in cryptocurrency and blockchain systems.

DISCUSSION

The assessment of the tools for detecting the vulnerabilities in Ethereum smart contracts offers the understanding of the advantages and the disadvantages of the tools in the context of cryptocurrency protection and blockchain efficiency. Oyente achieved the highest precision with 91% and recall with 90% but the execution time was relatively higher so it is more appropriate for the environment where precision is more important than time. Slither is more efficient with a time of 10 seconds per contract and is therefore suitable for real-time applications even though it has slightly less accuracy of 89%. Mythril had a higher false positive rate of 12% which may be overwhelming for users, while Securify and Slither showed better performance for larger contracts and hence suitable for enterprise level contracts.

In comparison with the existing literature, the results are consistent with the prior studies that show that the dynamic tools such as Oyente are highly accurate but time consuming in terms of computation whereas, the static tools such as Slither are less accurate but provide results in less time. The study benefits both the academic literature and the real world for organizations in the cryptocurrency and blockchain security.

Some of the limitations include the size of the dataset and the fact that the study is confined to Ethereum contracts only and therefore future research should consider cross-platform analysis and use a larger dataset. Moreover, increasing the efficiency of dynamic tools such as Oyente or considering the combination of static and dynamic analysis or machine learning could help to improve the precision-speed ratio.

CONCLUSION

This research provided a comparison of vulnerability detection tools for Ethereum smart contracts based on precision, recall and analysis time. Oyente was more precise than the other tools, whereas Slither was quicker, which is why it can be applied in real-time. The major contribution of the study is in presenting a comparison of the above tools which is not available in the current literature and would be useful for both academics and practitioners in the areas of cryptocurrency and blockchain security.

The general contribution of this work is to give developers and organizations a way of selecting the right tool based on the need for accuracy or speed. Some of the practical uses are in the improvement of security in DeFi and blockchain systems. The future work can explore the applicability of dynamic analysis tools like Oyente, comparison of tools across different platforms and integration of both static and dynamic analysis for better vulnerability detection.

REFERENCES

[1] Ren, M., Yin, Z., Ma, F., Xu, Z., Jiang, Y., Sun, C., ... & Cai, Y. (2021, July). Empirical evaluation of smart contract testing: What is the best choice?. In *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis* (pp. 566-579).

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

[3] Chen, J., Xia, X., Lo, D., & Grundy, J. (2021). Why do smart contracts self-destruct? investigating the selfdestruct function on ethereum. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *31*(2), 1-37.

[4] Wang, S., Huang, C., Li, J., Yuan, Y., & Wang, F. Y. (2019). Decentralized construction of knowledge graphs for deep recommender systems based on blockchain-powered smart contracts. *IEEE Access*, *7*, 136951-136961.

[5] Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., & Lee, H. N. (2022). Ethereum smart contract analysis tools: A systematic review. *Ieee Access*, *10*, 57037-57062.

[6] Zheng, Z., Xie, S., Dai, H. N., Chen, W., Chen, X., Weng, J., & Imran, M. (2020). An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, *105*, 475-491.

[7] Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., & Lee, H. N. (2022). Systematic review of security vulnerabilities in ethereum blockchain smart contract. *IEEE Access*, *10*, 6605-6621.

[8] Qian, P., Liu, Z., He, Q., Huang, B., Tian, D., & Wang, X. (2022). Smart contract vulnerability detection technique: A survey. *arXiv preprint arXiv:2209.05872*.

[9] 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).

[10] Panigrahi, R., & De Albuquerque, V. H. C. Big Data and Edge Intelligence for Enhanced Cyber Defense.

[11] Dia, B., Ivaki, N., & Laranjeiro, N. (2021, December). An empirical evaluation of the effectiveness of smart contract verification tools. In *2021 IEEE 26th Pacific Rim International Symposium on Dependable Computing (PRDC)* (pp. 17-26). IEEE.

[12] Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., & Lee, H. N. (2022). Ethereum smart contract analysis tools: A systematic review. *Ieee Access*, *10*, 57037-57062.

[13] Meidute-Kavaliauskiene, I., Yıldız, B., Çiğdem, Ş., & Činčikaitė, R. (2021). An integrated impact of blockchain on supply chain applications. *Logistics*, *5*(2), 33.

[14] Zygiaris, S., Saleh, M. F., & Al-Imamy, S. Y. (2023). The smart city blockchain governance: a literature review. *Central European Management Journal*, *31*(1), 313-332.

[15] Rath, M., Satpathy, J., & Oreku, G. S. (2021). Artificial intelligence and machine learning applications in cloud computing and Internet of Things. In *Artificial intelligence to solve pervasive internet of things issues* (pp. 103-123). Academic Press.

[16] Zou, J., He, D., Zeadally, S., Kumar, N., Wang, H., & Choo, K. R. (2021). Integrated blockchain and cloud computing systems: A systematic survey, solutions, and challenges. *ACM Computing Surveys (CSUR)*, *54*(8), 1-36.

[17] Duan, L., Sun, Y., Ni, W., Ding, W., Liu, J., & Wang, W. (2023). Attacks against cross-chain systems and defense approaches: A contemporary survey. *IEEE/CAA Journal of Automatica Sinica*, *10*(8), 1647-1667.

[18] 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.

[19] 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*.

[20] Eshghie, M., Artho, C., & Gurov, D. (2021, June). Dynamic vulnerability detection on smart contracts using machine learning. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering* (pp. 305-312).

[21] Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., & Lee, H. N. (2022). Ethereum smart contract analysis tools: A systematic review. *Ieee Access*, *10*, 57037-57062.

[22] Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., & Vechev, M. (2018, October). Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security* (pp. 67-82).