A Comparative Evaluating Auditing Tools for Unverified Smart Contracts on Ethereum Blockchain

Nashaib Akbar, Muhammad Saleem Vighio
Department of Computer Science Quaid-e-Awam University of Engineering, Sciences & Technology Nawabshah, Pakistan
***Correspondence**: nashaibakbar30@gmail.com, saleem.vighio@quest.edu.pk

Citation | Akbar. N, Vighio. M. S, "A Comparative Evaluating Auditing Tools for Unverified Smart Contracts on Ethereum Blockchain", IJIST, Vol. 07 Special Issue. pp 85-96, May 2025
Received | April 12, 2025 **Revised |** May 10, 2025 **Accepted |** May 12, 2025 **Published |** May 14, 2025.

The Ethereum blockchain has transformed decentralized finance (DeFi) and is widely used to issue ERC20 tokens. However, many of these tokens rely on unverified smart contracts, which pose serious security risks. Hackers can take advantage of vulnerabilities in these unverified ERC20 tokens, leading to scams, financial losses, and a decline in user trust. Although several tools are available to audit smart contracts, their effectiveness in analyzing unverified ERC20 tokens remains uncertain. This study examines three auditing tools HoneyBadger, Maian, and Mythril by testing how well they detect security issues in unverified ERC20 tokens. The SmartBugs framework was used to support the auditing process, enabling parallel execution, standardized reports, and bulk auditing of contracts. For a thorough evaluation, two datasets were used: one from 50,581 Ethereum blockchain blocks and another from the DappRadar list of blacklisted ERC20 tokens. These datasets were chosen to provide a broad and realistic view of how the tools perform on both typical and high-risk contracts. The tools were compared based on their ability to detect issues, their execution speed, and their overall effectiveness. The results revealed clear differences in performance: some tools were better at finding vulnerabilities accurately, while others focused more on speed than depth. This study emphasizes the need to improve smart contract auditing methods and highlights the importance of developing more effective security tools to strengthen the Ethereum blockchain.

Keywords: Ethereum Blockchain, smart contract, ERC20 token, security risks, vulnerabilities, runtime bytecode.

Introduction:

Ethereum is a blockchain that supports a Turing-complete programming language, allowing the creation of smart contracts capable of encoding complex state transition functions. This powerful functionality enables users to build a wide range of systems by writing relatively simple code [1]. Among its many applications, ERC20 tokens have become the most widely adopted standard for digital assets. These tokens easily integrate with decentralized applications (Dapps), supporting functions such as payments, staking, and governance.

However, the rise of scam tokens on the Ethereum blockchain is concerning. Fraudulent activities like rug pulls and counterfeit tokens have become increasingly common on decentralized exchanges. These scams often involve manipulated token prices that deceive investors, leading to significant financial losses [2]. One of the main challenges Ethereum faces is ensuring the security of smart contracts. Vulnerabilities in contracts have been exploited in high-profile attacks, resulting in major financial damage and a decline in public trust in blockchain ecosystems. With the growing use of decentralized applications and digital assets, ensuring the security of end-user interactions has become increasingly critical. Similar to how Android applications particularly games have become a common vector for malware due to their popularity and users' limited visibility into their inner workings [3], smart contracts on the Ethereum blockchain face similar risks when their source code is unverified. Just as malicious Android apps can exploit users trust and compromise sensitive information through hidden logic, unverified ERC20 tokens may embed vulnerabilities or malicious behaviors that go undetected. This highlights the urgent need for robust auditing tools capable of analyzing unverified smart contracts bytecodes to ensure security and protect users. Unverified ERC20 tokens are particularly problematic because, unlike verified contracts, their code is not openly accessible for review. This allows developers to hide malicious logic, leading to more frequent scams, greater financial loss, and growing distrust in blockchain systems. While current auditing tools are effective with verified contracts that have accessible source code, they may fall short when analyzing unverified contracts. Therefore, comprehensive studies are needed to assess these tools' abilities to detect vulnerabilities and malicious behaviors in unverified smart contracts.

This study aims to improve the security and reliability of smart contract ecosystems. It utilizes two datasets: the first includes data from 50,581 Ethereum blockchain blocks, and the second comes from a GitHub repository of blacklisted ERC20 tokens [4], containing 985 tokens.

The study analyzes the performance of three popular auditing tools HoneyBadger [5], Maian [6], and Mythril [7] in detecting vulnerabilities and malicious behavior in unverified ERC20 tokens. The SmartBugs framework [8] was used to facilitate the process by enabling parallel execution and bulk auditing. This work provides a comparison of the tools in terms of accuracy, execution time, and vulnerability detection. It also offers valuable insights into the strengths and limitations of each tool, along with practical recommendations to improve smart contract auditing and enhance security on the Ethereum platform.

Objectives:

This research will assess each tool and provide insights into their capabilities and limitations. The key objectives of this research are as follows:

- To evaluate the effectiveness of tools named HoneyBadger [5], Maian [6], and Mythril [7] in finding the vulnerabilities in unverified ERC20 tokens. In this, we will check accuracy, execution time, and ability to detect the vulnerabilities.
- To check the performance of these tools we have two different datasets: One is collected directly from the Ethereum Blockchain and the second is collected from the

DappRadar scam token repository [4]. This comparison provides an depth understanding of tools performance over the unverified smart contracts.

- They contribute to the understating of the smart contract security in unverified ERC20 tokens and provide insights that can help improve the security of the Ethereum blockchain.

Novelty Statement:

This study uniquely focuses on auditing of unverified ERC20 smart contracts an area often overlooked in prior research, which typically centers on verified contracts with public source code. By evaluating the performance of three auditing tools Mythril, Maian, and HoneyBadger on runtime bytecode from unverified contracts, and using a two-dataset approach Ethereum blockchain and a blacklist of scam tokens, this research provides a practical and targeted analysis. The findings help uncover tools limitations and highlight the need for improved vulnerability detection in unverified smart contracts.

Related Work:

Recent advancements in the automated analysis, testing, and debugging of Ethereum smart contracts have spurred significant research progress. However, comparing and replicating these studies has been challenging due to variations in datasets and methodologies. To address this, a study conducted an empirical evaluation of nine automated analysis tools using two novel datasets. One dataset contained 69 annotated smart contracts with known vulnerabilities, serving as a benchmark for testing the tools' accuracy, while the second dataset included 47,518 smart contracts from the Ethereum blockchain, with Solidity code accessible via Etherscan. These datasets were integrated into SmartBugs, a framework designed to compare analysis tools. The study conducted a massive 428,337 analyses over approximately 564 days, making it the largest experimental setup of its kind. The results showed that only 42% of the vulnerabilities in the annotated dataset were detected across all tools, with Mythril detecting the highest rate (27%). In the larger dataset, 97% of contracts were flagged as vulnerable, revealing a high rate of false positives. Notably, only a few vulnerabilities were detected by multiple tools, highlighting the evolving and fragmented nature of smart contract security [9].

Smart contracts, which manage digital assets and power decentralized applications in DeFi and supply chains, require a high level of security. This is difficult to maintain due to the transparency of public blockchains and the rapid evolution of blockchain technologies. Several code analysis tools exist to help identify vulnerabilities, but staying up-to-date remains a challenge for both tools and developers. In another study [2], the authors evaluated the effectiveness of these tools by analyzing how vulnerability detection has evolved over six years of Ethereum data. They examined all deployed bytecode on Ethereum, using a "skeleton" approach to group similar contracts, reducing the dataset from 48 million to approximately 248,000 unique samples. By enhancing the SmartBugs framework and integrating 12 different tools, the study conducted a large-scale analysis that took 30 CPU years and identified over 1.3 million potential issues. However, it also showed a decline in reported vulnerabilities over time and varying performance among the tools.

As security breaches in smart contracts continue to result in major financial losses and reduced trust, it's become essential to detect vulnerabilities before deployment. While many static analysis tools exist for this purpose, there has been no standardized method to evaluate their effectiveness. In response, a study [10] introduced the SolidiFI framework—a structured approach for assessing smart contract analysis tools through bug injection. SolidiFI systematically injects vulnerabilities into various parts of a contract, creating faulty versions that are then scanned by the tools. This allows researchers to measure false negatives (missed bugs) and false positives (incorrectly flagged bugs). Using a dataset of 50 contracts and 9,369 injected bugs, the framework evaluated six popular tools: Oyente, Securify, Mythril, SmartCheck, Manticore, and Slither. The results showed that many vulnerabilities went undetected and all tools produced a high number of false positives, raising concerns about their practical reliability.

Smart contract security has become an increasingly important area of research, prompting the development of many testing tools aimed at detecting vulnerabilities before deployment. However, each tool is typically evaluated under different experimental setups, using varied datasets and performance metrics. This inconsistency makes it difficult to compare results across studies. In a comprehensive study [11], the authors addressed this issue by introducing a standardized four-step evaluation methodology. They compiled a large dataset of 46,186 source-available smart contracts from four major platforms to ensure diversity in code structure and vulnerability types. Using nine well-known testing tools, they conducted controlled experiments to assess performance differences under uniform conditions. The results highlighted how changes in evaluation design can significantly affect tool outcomes, sometimes leading to misleading conclusions. The study also identified limitations in current testing tools and proposed directions for improving consistency and reliability in future evaluations.

Research Questions:

RQ1: What is the prevalence of unverified ERC20 token contracts on the Ethereum blockchain, and how are they linked to scam activities and security risks?

RQ2: How effective are current smart contract auditing tools at identifying vulnerabilities and detecting malicious functions in unverified ERC20 token contracts?

RQ3: What are the limitations of existing auditing tools, and how can their accuracy and performance be improved for analyzing unverified contracts in the Ethereum ecosystem?

Methodology:

This study adopts a structured approach that includes selecting auditing tools, collecting relevant data, and evaluating performance. Effective tools are chosen to analyze unverified ERC20 contracts using datasets from the Ethereum blockchain and a blacklist repository. The tools are assessed based on their detection accuracy and operational efficiency. The goal is to provide valuable insights for enhancing smart contract security by addressing key research questions.

Table 1. Ethereum dataset findings.

#	Category	Smart Contracts Count
1	Unverified Contracts	3,510
2	Verified Contracts	2,437
3	ERC20 Verified Tokens	1,364
4	ERC20 Unverified Tokens	353
5	Total ERC20 Tokens	1,717
6	Total Smart Contracts	5,947

The second source of the data is the DappRadar [4] blacklisted token repository, which has a list of tokens that were involved in the scam activity over the Ethereum blockchain. This is an important dataset that is involved in fraudulent activities. By using both datasets from the Ethereum blockchain and DappRadar blacklisted tokens we create a comprehensive dataset that allows us to explore the more common vulnerabilities in this study.

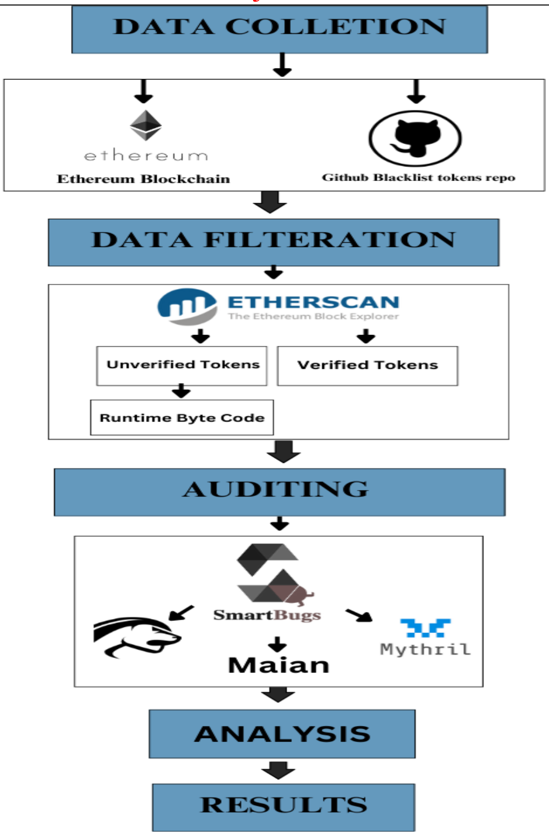

Figure 1. Flow diagram

Data Collection:

The first step of the methodology involved collecting data from the Ethereum blockchain. We scanned blocks starting from block number 18,908,895 to block number 18,959,476, covering a total of 50,581 blocks. After examining each block, we identified and collected a total of 5,947 smart contracts. Following this, we checked the status of each contract to determine whether it was verified or unverified, as our study specifically focuses on unverified ERC20 tokens.

Table 2. DappRadar dataset findings

#	Category	Smart Contracts Count
1	ERC20 Verified Tokens	675
2	ERC20 Unverified Tokens	310
3	Total ERC20 Tokens	985

After collecting data from both sources, we proceeded to categorize the smart contracts into verified and unverified contracts, as our study focuses only on unverified smart contracts. From the 5,947 smart contracts collected, we found that 3,510 were unverified and 2,437 were verified. Focusing on ERC20 unverified tokens, we examined the 3,510 unverified contracts and identified 353 as ERC20 tokens. The verification status of these contracts (verified or unverified) was confirmed using the Etherscan API [12].

Data Filtration:

The data filtration phase focuses on the get the final data for the input of the auditing tools from the unverified smart contract to the runtime bytecode of all collected data from both of the datasets which are sourced from the Ethereum blockchain and DappRadar blacklist token repository. The goal of this phase is to organize the collected smart contracts into verified and unverified smart contracts to make sure they are perfectly categorized. The dataset which is a direct source from the Ethereum blockchain has multiple types of contracts, not a single category like they follow multiple standards like ERC711 and others, our study

focused on the ERC20 standard so first we filter the smart contract to only the ERC20 standard and we need to select the only unverified ERC20 token so we used the Etherscan API and selected only unverified contract, now comes to the second dataset which is already the blacklisted ERC20 tokens we need to divide this into two categories unverified ERC20 and verified ERC20 tokens and selects the addresses of the first unverified ERC20 tokens.
To evaluate the tools based on runtime bytecode, it was essential to collect the runtime bytecode of these unverified ERC20 tokens. The bytecode is available on Etherscan, so we utilized the Etherscan API and developed a Python script to gather the bytecode. We input the contract addresses of the identified unverified tokens into the script and ran it on Google Colab [13], collecting the runtime bytecode for all unverified ERC20 tokens, which was then used in the subsequent auditing process.

Once the data is organized and well-structured now it is prepared for the next phase of the auditing. The collected runtime bytecode which is sourced from the Ethereum blockchain and the DappRadar blacklist token repository is now ready for the auditing phase. This preparation includes the tagging of each runtime bytecode with their contract address and the dataset group by this both datasets are correctly formatted and aligned for the use of the auditing process.

Auditing:

The purpose of this phase is to audit the collected runtime bytecode of two datasets to identify the vulnerabilities among them by the selected auditing tools. This phase is important because in this phase we examined the auditing tools and give them input as selected runtime bytes which are collected from the last few phases to find out the common vulnerabilities such as reentrancy attacks, integer overflow, and funds leakages that could potentially compromise the functionality and security ER20 tokens.

Selection of Tools:

In our study, we aim to evaluate the effectiveness of smart contract auditing tools that support runtime bytecode analysis. From an initial list of 140 tools identified in the literature [14], we narrowed the selection to 46 tools that specifically support bytecode analysis. However, several of these tools had inactive repositories or were no longer fully accessible.

Selection Criteria:

To ensure the selection of effective and practical tools for analyzing unverified smart contracts, the following four criteria were applied:

- **Criterion #1: Public Availability & CLI Support:** The tool must be publicly available and support a command-line interface (CLI) to enable automation and bulk analysis.
- **Criterion #2: Compatibility with Runtime Bytecode:** The tool must be capable of analyzing runtime bytecode, as unverified smart contracts do not have accessible source code.
- **Criterion #3: Vulnerability Detection:** The tool must detect security vulnerabilities. Tools that only provide metrics or basic information were excluded.
- **Criterion #4: Active Repository:** The tool must be actively maintained, ensuring that it receives regular updates and security patches.

Table 3. Tools identified as potential candidates for this study

#	Tools	Tools Url
1	Mythril [7]	https://github.com/wflk/mythril
2	Maian [6]	https://github.com/ivicanikolicsg/MAIAN
3	Honey Badger [5]	https://github.com/christoftorres/HoneyBadger

Selected Tools:

Mythril [7]: A symbolic execution-based tool designed to detect issues such as reentrancy, integer overflows, and access control flaws. It uses advanced execution techniques to enhance detection accuracy and supports both source code and runtime bytecode analysis. Mythril is actively maintained and widely used in the security community.

Maian [6]**:** Focuses on identifying vulnerabilities like fund leakage, self-destruct risks, and frozen funds. It analyzes Ethereum Virtual Machine (EVM) bytecode to find transaction sequences that may trigger these critical issues.

HoneyBadger [5]**:** Specializes in detecting honeypot contracts, which are fraudulent smart contracts that trick users into sending funds they cannot retrieve. It systematically analyzes execution paths to uncover deceptive behaviors designed to trap funds.

Execution Framework: SmartBugs:

SmartBugs [8] is an extensible framework designed to streamline the auditing process of Ethereum smart contracts. It supports integration with a wide range of static analysis tools, making it well-suited for identifying vulnerabilities in unverified contracts. By offering a centralized and modular environment, SmartBugs simplifies the auditing workflow and enhances the detection of security issues in contracts lacking source code. In this study, all selected tools were executed using the SmartBugs framework, which enabled parallel execution and ensured consistent and efficient auditing of unverified ERC20 contracts.

Analysis and Results:

The results from the auditing tools Maian, Mythril, and HoneyBadger are carefully reviewed to understand the vulnerabilities identified in the runtime bytecode of the unverified ERC20 tokens. Since the source code is unavailable for these contracts, the analysis is based on patterns identified within the bytecode. The tools detect vulnerabilities such as reentrancy attacks, integer overflows, and honeypot traps. The findings from each tool are compared to assess their consistency, and any discrepancies are examined for potential false positives. Each detected vulnerability is validated by cross-referencing with known scam patterns from the second dataset, which includes tokens flagged for fraudulent activities.

To assess the effectiveness of the selected tools, we evaluated their detection accuracy, efficiency, and limitations. The tools were compared based on the following criteria:

- **Detection Accuracy:** The ability to identify critical vulnerabilities, such as access control issues, reentrancy, and integer overflows.
- **Efficiency:** Measured by execution time per contract, highlighting the trade-off between speed and the depth of analysis.

Table 4. Vulnerabilities mapping of tool findings

#	Category	HoneyBadger	Main	Mythril
1	Access Control	0	0	12
2	Arithmetic	0	0	329
3	Denial of Service	0	0	0
4	Reentrancy	0	0	2
5	Unchecked Low Calls	0	0	1
6	Bad Randomness	0	0	0
7	Front Running	0	0	0
8	Time Manipulation	0	0	0
9	Short Addresses	0	0	0
10	Other	1	214	39
	Total	1	214	383

RQ1: Prevalence of Unverified ERC20 Tokens and Security Risks:

In this research, we audited the runtime bytecode of unverified ERC20 tokens to identify vulnerabilities within those contracts. From both datasets, we found that 3,820 out of 6,932 contracts were unverified, accounting for 55% of the total collected data. This highlights the critical need for checking unverified smart contracts on the Ethereum blockchain.

Unverified ERC20 token smart contracts pose a significant security risk and are increasingly involved in scam activities. Due to the lack of verification, these contracts often escape proper scrutiny, creating opportunities for malicious actors to embed harmful or hidden functions in the code. Since users cannot view the source code, they are unaware of the full functionality of these contracts and become vulnerable to attacks such as reentrancy, integer overflows, access control flaws, or other malicious operations. These vulnerabilities can be exploited to steal funds, lock assets, or perform unauthorized actions within the Ethereum ecosystem.

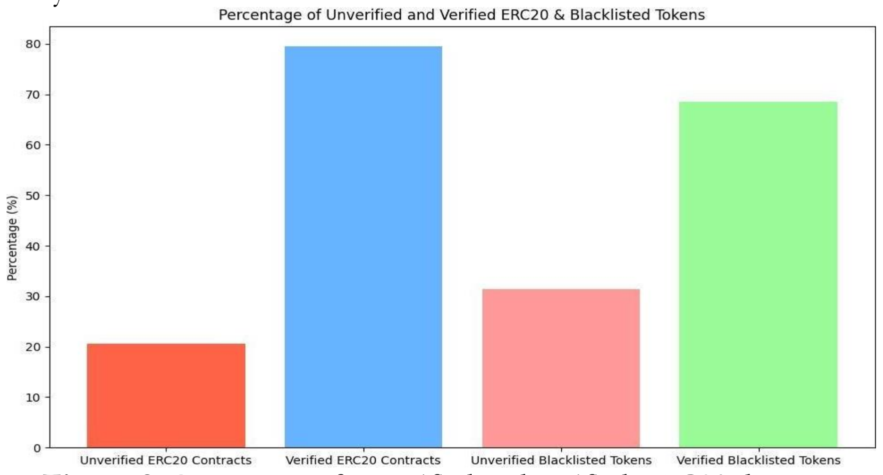

Figure 2. Percentage of unverified and verified ERC20 datasets.

Moreover, unverified ERC20 token smart contracts often appear on blacklists due to their association with fraudulent activities. In our experiment, we also examined a set of blacklisted tokens, comprising those reported for fraudulent or suspicious activities. Of the 985 blacklisted tokens, 310 were unverified ERC20 tokens, making up 31.5%. These tokens are frequently used in scams like pump-and-dump schemes, where token prices are artificially inflated to deceive investors, or honeypots, where users are tricked into depositing funds into a contract they cannot withdraw from. The data [12] reveals that unverified ERC20 token smart contracts account for a significant portion of the Ethereum blockchain, with 59% of all smart contracts being unverified and around 21% of ERC20 tokens falling into this category. These unverified contracts are major contributors to security risks and scams, undermining trust and security in the Ethereum ecosystem.

The vulnerability detection results, presented in Table 4, were derived from both datasets to ensure a comprehensive evaluation. The first dataset provided a broad sample of unverified contracts, while the second, containing blacklisted tokens, represented high-risk cases. The consistent 0's across multiple vulnerability categories indicates that these issues were either absent in the dataset or not detected by the auditing tools used.

The presence of multiple zero detections across various vulnerability categories may reflect limitations in the dataset size rather than the absence of such vulnerabilities. This suggests a potential limitation of the current study, as a larger and more diverse dataset might yield more meaningful insights. Future research should explore whether expanding the dataset leads to improved detection outcomes or reveals vulnerabilities that remained undetected in this study.

RQ2: Effectiveness of Smart Contract Auditing Tools in Detecting Vulnerabilities in Unverified ERC20 Contracts:

The HoneyBadger tool completed the audit in an average time of 2.28 seconds per contract, across both datasets, which included 353 unverified ERC20 tokens and 310 blacklisted unverified ERC20 tokens. Although HoneyBadger successfully audited all the contracts, it did not identify any major vulnerabilities. This suggests that, while HoneyBadger excels in processing speed, it may lack the capability to detect more complex vulnerabilities or malicious functions in unverified and blacklisted ERC20 contracts. The inability of the tool to identify common issues in these tokens often associated with fraudulent activities and hidden risks remains a significant limitation, despite its speed.

Table 6. Average execution time for each tool.

#	Tools	Average Execution Time	Total
1	Mythril	0:12:14	132:27:46
2	MAIN	0:00:11	1:59:35
3	HONEY BADGER	0:00:02	0:24:40

Maian showed better vulnerability detection capabilities than HoneyBadger. In the dataset of 663 contracts, which included 353 unverified ERC20 tokens and 310 blacklisted ERC20 tokens, Maian completed the audits in 7,175.90 seconds (~0.08 days), averaging 11.06 seconds per contract. It detected vulnerabilities in 214 contracts, accounting for 32.3% of the total audited contracts. While Maian required more processing time than HoneyBadger, it proved to be more consistent in identifying security vulnerabilities across both datasets. Finally, despite having the longest execution time, Mythril proved to be the most thorough auditing tool. It took an average of 734.77 seconds per contract to analyze the 663 contracts from the combined dataset (310 blacklisted ERC20 tokens and 353 unverified ERC20 tokens). Mythril detected vulnerabilities in 383 contracts, accounting for 57.8% of the total analyzed contracts.

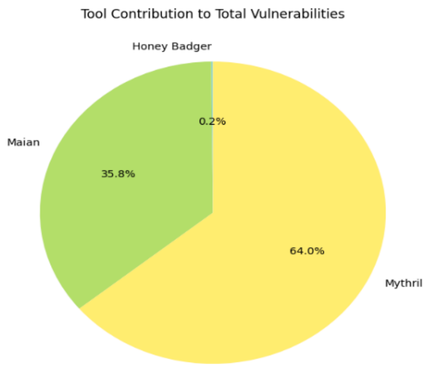

Figure 3. Tools contribution to total vulnerability detection.

Although Maian detected fewer vulnerabilities than Mythril, it was still more effective than HoneyBadger. However, Mythril proved to be the most effective tool for identifying critical security issues and detecting hidden malicious features, especially within complex contracts. Among all three tools, Mythril provided the most comprehensive analysis, making it a valuable auditing resource despite its longer execution time.

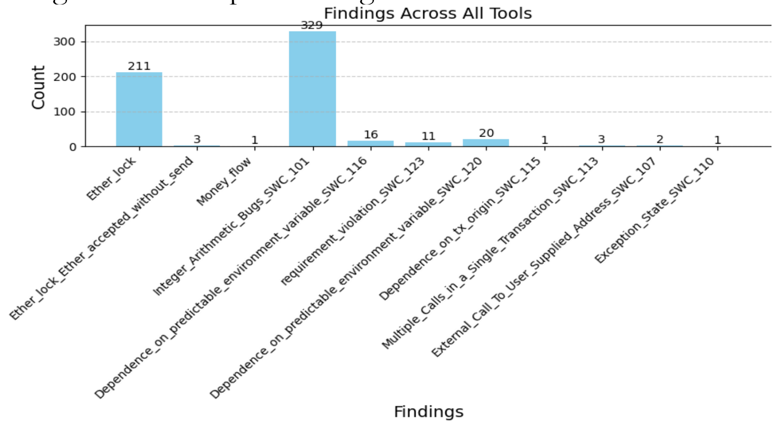

Figure 4. Comparing vulnerabilities detected by each tool

All the results, including the input runtime bytecode, output reports from each auditing tool Mythril, Maian, and HoneyBadger, and the Python scripts used for data collection and processing, are available in the public GitHub repository [10].

RQ3: Limitations of Auditing Tools and Improving Accuracy for Unverified Ethereum Contracts:

The findings of this study highlight several limitations in existing auditing tools when analyzing unverified ERC20 contracts. While Mythril, Maian, and HoneyBadger provided valuable insights, their effectiveness varied significantly based on contract complexity, execution time, and detection capabilities.

One major limitation observed in Table 4 was the presence of multiple zero detections across various vulnerability categories. This raises concerns about whether the tools were unable to detect specific vulnerabilities or if such issues were genuinely absent in the dataset. The zero detections in categories such as Denial of Service, Bad Randomness, and Short Addresses suggest that either these vulnerabilities are rare in unverified ERC20 contracts or that current tools are not fully equipped to detect them. To ensure accuracy, we verified that the tools were properly configured and executed, confirming that the results were not due to technical errors.

Additionally, these tools exhibited varying levels of false negatives, particularly in detecting complex attack vectors like fund leakage and logic manipulation. While Mythril provided the most comprehensive analysis, its high execution time limited its scalability for large-scale contract auditing. Maian, though efficient, struggled with detecting deeper vulnerabilities, and Honey Badger's speed came at the cost of reduced accuracy, detecting only a limited set of issues.

To improve the accuracy and reliability of smart contract auditing, future improvements should focus on:

- Integrating machine learning models to detect evolving vulnerabilities more effectively.
- Leveraging real-time threat intelligence to dynamically update detection mechanisms.
- Developing a hybrid approach that combines automated scanning with manual auditing for better validation of results.

These advancements would enhance vulnerability detection rates, reduce false negatives, and strengthen the overall security framework for Ethereum smart contracts. Future studies should also expand the dataset size to verify whether certain vulnerabilities become more apparent with larger contract samples.

Discussion:

The core difference between prior auditing evaluations and our proposed study lies in the contracts analyzed and the methodology used. Traditional studies primarily assess tools using verified contracts with publicly available source code [9][11], which simplifies vulnerability detection. In contrast, our study focuses on unverified ERC20 smart contracts using runtime bytecode as the sole input. This approach allows our evaluation to reflect real-world conditions where source code is unavailable and contracts may conceal malicious logic. We tested three tools Mythril, Maian, and HoneyBadger on two datasets one collected from 50,581 Ethereum blockchain blocks and another from a blacklist of scam tokens [4]. This dual-dataset strategy provided both breadth and depth to our analysis. Mythril outperformed the others in detecting complex vulnerabilities but required significantly more execution time [7]. Maian offered a balance between speed and accuracy [6], while HoneyBadger, though the fastest, detected only limited issues [5]. These results show that when auditing unverified contracts, tool selection must consider both detection capability and efficiency.

Conclusion:

In this study, we analyzed unverified smart contracts by auditing their runtime bytecodes to uncover potential vulnerabilities. Our findings revealed that 3,820 out of 6,932 smart contracts (around 55%) were unverified, many of which were linked to scams and posed significant security risks. To address this, we tested three auditing tools: HoneyBadger, Maian, and Mythril, each demonstrating distinct strengths and weaknesses. Mythril emerged as the best tool, capable of detecting a broad range of vulnerabilities such as reentrancy attacks, integer overflows, and access control flaws. However, its long execution time made it less practical for large-scale audits. Maian, while much faster and efficient for scanning large datasets, struggled with detecting more complex vulnerabilities. HoneyBadger, the fastest of the three, could only detect a limited number of vulnerabilities, notably missing critical issues like reentrancy and fund leakage, which are major gaps in its detection capabilities.

This study emphasizes the importance of selecting the appropriate tool based on whether speed or detection depth is prioritized. Future research should focus on developing smarter and more efficient auditing tools that reduce false negatives, improve detection accuracy, and handle complex contract analysis more effectively. By highlighting the strengths and weaknesses of each tool, this study contributes to the advancement of Ethereum Blockchain smart contract security and supports the overall Ethereum ecosystem.

Acknowledgment:

This paper is based on our Master's thesis defended in April 2025 at Quaid-e-Awam University of Engineering Science & Technology, Nawabshah. I also acknowledge the developers of SmartBugs, HoneyBadger, Maian, and Mythril for providing the tools necessary for this study. Additionally, I appreciate the Ethereum community for their contributions and the reviewers for their helpful feedback.

References:

[1] V. Buterin, "Ethereum white paper," *gitHub Repos.*, pp. 22–23, 2013.

[2] A. Ghaleb and K. Pattabiraman, "How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection," *ISSTA 2020 - Proc. 29th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, pp. 415–427, Jul. 2020, doi: 10.1145/3395363.3397385;SUBPAGE:STRING:ABSTRACT;TOPIC:TOPIC:CONFERENCE-COLLECTIONS>ISSTA;CSUBTYPE:STRING:CONFERENCE.

[3] B. A. Dahri, K. A., Vighio, M. S., & Zardari, "Detection and prevention of malware in the Android operating system," *Mehran Univ. Res. J. Eng. Technol.*, vol. 40, no. 4, pp. 847–859, 2021, [Online]. Available: https://www.researchgate.net/publication/355269909_Detection_and_Prevention_of_Malware_in_Android_Operating_System

[4] DappRadar, "Tokens Blacklist," *GitHub Repos.*, 2024, [Online]. Available: https://github.com/dappradar/tokens-blacklist/tree/main

[5] R. Torres, C. F., Steichen, M., & State, "The Art of The Scam: Demystifying Honeypots in Ethereum Smart Contracts," *Adv. Comput. Syst. Assoc.*, 2019, [Online]. Available: http://usenix.org/system/files/sec19-torres.pdf

[6] A. H. Ivica Nikolić,, Aashish Kolluri, Ilya Sergey, Prateek Saxena, "Finding The Greedy, Prodigal, and Suicidal Contracts at Scale," *ACSAC '18 Proc. 34th Annu. Comput. Secur. Appl. Conf.*, pp. 653–663, 2018, doi: https://doi.org/10.1145/3274694.3274743.

[7] B. Mueller, "Smashing Ethereum Smart Contracts for Fun and ACTUAL Profit," *HITB SECCONF*, 2018, [Online]. Available: https://archive.conference.hitb.org/hitbsecconf2018ams/sessions/smashing-ethereum-smart-contracts-for-fun-and-actual-profit/

[8] M. Di Angelo, T. Durieux, J. F. Ferreira, and G. Salzer, "SmartBugs 2.0: An Execution Framework for Weakness Detection in Ethereum Smart Contracts," *Proc. - 2023*

38th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2023, pp. 2102–2105, 2023, doi: 10.1109/ASE56229.2023.00060.
[9] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 ethereum smart contracts," *Proc. - Int. Conf. Softw. Eng.*, pp. 530–541, Jun. 2020, doi: 10.1145/3377811.3380364;PAGE:STRING:ARTICLE/CHAPTER.
[10] Nashaib Akbar, "Unverified Smart Contracts," *GitHub*, [Online]. Available: https://github.com/Nashaibakbar/Unverified-Smart-Contracts-
[11] M. Ren *et al.*, "Empirical evaluation of smart contract testing: What is the best choice?," *ISSTA 2021 - Proc. 30th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, pp. 566–579, Jul. 2021, doi: 10.1145/3460319.3464837.
[12] Etherscan API, "Etherscan API Documentation," *Etherscan*, [Online]. Available: https://etherscan.io/
[13] Google Colab, "Google Colaboratory: Python in the Cloud", [Online]. Available: https://colab.research.google.com
[14] G. S. Heidelinde Rameder, Monika di Angelo, "Review of Automated Vulnerability Analysis of Smart Contracts on Ethereum," *Front. Blockchain*, vol. 5, 2022, doi: https://doi.org/10.3389/fbloc.2022.814977.