International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(18s) 2026 pp. 185-193
https://doi.org/10.70917/ijcisim-2026-4848
Article

Adaptive Multi-Agent Explainable Smart Contract Auditing Framework (AMESCAF): A Hybrid Approach Integrating Static Analysis, Symbolic Execution, Formal Verification, and Explainable Artificial Intelligence for Next-Generation Blockchain Security

Alfiya Sayyad[1], Tushar H. Ghorpade[2], Vanita Mane[3], Faimida Sayyad[4]

[1]Ramrao Adik Institute of Technology, Navi Mumbai, Maharashtra, India.
Email: sayyadalfiya2@gmail.com
[2]Ramrao Adik Institute of Technology, Navi Mumbai, Maharashtra, India.
Email: tushar.ghorpade@rait.ac.in
[3]Mukesh Patel School of Management & Engineering (MPSTME), SVKM's NMIMS, Mumbai, Maharashtra, India.
Email: Vanita.Mane@nmims.edu
[4]Kalsekar Technical Campus, New Panvel, Maharashtra, India.
Email: faimida.sayyad@aiktc.ac.in

Abstract: Smart contracts are a crucial element in blockchain technology, enabling the decentralisation and automation of digital transactions. However, re-entrancy, access control, integer overflows and logic errors all pose a danger to their security and reliability. Previously available audit solutions rely on a single analytical approach, resulting in insufficient detection coverage, high false-positive rate and poor explainability. We propose an AMESCAF to eliminate the aforementioned problems, which adopts static analysis, dynamic testing, symbolic execution, formal verification and AI-assisted reasoning. Common knowledge base and consensus validation method enhances vulnerabilities detection rate and minimizes the duplicate vulnerability discovery. Moreover, there is a context-aware risk assessment model which categorizes vulnerabilities according to technical severity, vulnerability exploitability, business impact and confidence scores. The explainability module gives visual hints on cures. The prototype has been analyzed with respect to the benchmark intelligent contracts, and benchmarked against the current smart contract auditing tools. Safe smart contract auditing was performed by comparing the proposed framework in terms of detection accuracy and analysis efficiency, achieving a 96.4% accuracy, 95.2% recall, 4.1% false positive rate, and 95.8% F1-score, demonstrating superior results.
Keyword: Blockchain Security, Smart Contract Auditing, Multi-Agent Systems, Explainable Artificial Intelligence (XAI), Static Analysis, Symbolic Execution, Formal Verification, Ethereum.

1. Introduction

With the development of blockchain technology, decentralised apps have been made possible, providing a secure, transparent and tamper-proof platform for digital transactions, without the use of any central middlemen. The most outstanding feature is the concept of Intelligent Contracts that permits automatic execution of pre-programmed business logic, thereby reducing operational costs, boosting confidence and enhancing the efficiency of transactions. Smart contract technology has been utilised in many areas such as DeFi, digital ID, healthcare, supply chain management, insurance and NFT with the use of platforms like Ethereum. But, due to the immutability of blockchain, security problems that have been propagated throughout the development process and can't be patched after deployment, can cause enormous financial and operational losses [1].

Over the last ten years, numerous attacks have been observed, including risks like reentrancy attacks, integer overflows, access control issues, front-running and oracles attacks. As the use of blockchain applications to utilise high value digital resources is increasing at a fast pace, correctness and security of the deployed intelligent contracts becomes one of the important topics for study. Several automated audit techniques have been put forward to uncover a product's vulnerabilities before deployment. Known pattern vulnerabilities can be easily identified by static analysis techniques, without executing the code. Symbolic execution can explore many paths in which the program can be executed and discover unknown logical bugs. The security is proved through formal

verification of the correctness properties specified in the contract, by using a mathematical proof. Dynamic testing involves the test of the contract behaviour in a real execution context [2] [3].

Though blockchain has significantly improved its security with current auditing techniques, there are still some drawbacks in practice. Static analysis has a lot of false positives because it doesn't have realtime context. There are however scaling issues with Symbolic execution due to path explosion in case of analysis of complicated contracts. However, the formal verification requires a lot of expertise on the topic and a strict specification of the formal language and is difficult to apply in large scale industrial applications. Moreover, current auditing systems are often standalone, and produce separate and different vulnerability assessments of varying levels of severity, and they typically fail to give any comprehensible decision making. This renders it challenging for developers to pinpoint the vulnerabilities that they should prioritize and understand the "why" behind the automation of security recommendations [4, 5].

The introduction of AI and the creation of LLMs has provided fresh opportunities for increasing the study of software security by translating code in context, explaining weaknesses, and providing intelligent remedial suggestions. Meanwhile, it has been proved that multi-agent systems can solve complex analytical problems. This is made possible by the cooperation of specialised autonomous agents sharing intermediate knowledge and coming to a consensus-based judgement. The field of collaborative multi-agent architectures and traditional smart contract auditing techniques still has some unexplored areas, particularly with respect to explainability and adaptive vulnerability assessment [10, 12].

To address these challenges, we present a hybrid security framework, AMESCAF, where static analysis, dynamic testing, symbolic execution, formal verification and AI based reasoning are jointly deployed and operated in a collaborative framework. This platform features its own analytical agents, a Vulnerabilities Knowledge Repository, a consensus-based validation of vulnerabilities, and a context-aware risk assessment model to help achieve more accurate detection and to minimize false-positive results. An explanation module is also provided for the clear explanation and organised repair solution of software secure development. To illustrate the applicability of the proposed approach, we evaluate a prototype of its implementation on a set of benchmark smart contract data sets and compare it to a number of the most popular auditing tools.

The main contributions of this paper are as follows:

- Complementary approach of a single architecture for multi-agent smart contract auditing platform.
- A shared knowledge base for a consensus validation process regarding vulnerabilities and enhancing reliability of detection.
- A context-aware risk assessment and explainability approach for vulnerabilities with visible recommendations for remedy
- A prototype implementation and an evaluation study which compares the auditing performance of the prototype with the current smart contract analysis tools.

2. RELATED WORK

With the growing acceptance of blockchain technology and the rising significance of electronic assets managed by dApps, the security of intelligent contracts has garnered much attention. Most of the existing studies are aimed at enhancing the vulnerability identification with the help of static analysis, symbolic execution, formal verification, dynamic testing and more recently AI-based techniques. These techniques have contributed much to the security of blockchain . However, no one solution can encompass all of the vulnerabilities. This encourages the development of the hybrid audit systems by applying two or more complementary analytical processes.

Static analysis is still one of the most common approaches for smart contract audits since it is fast to conduct, scalable and fits easily into software development workflows. Most of the vulnerability like re-entrancy, access control, arithmetic and unsafe external calls can be identified using modern static analysis tools like creation of abstract syntax tree, control flow graph and data dependency graph. These techniques are quite effective on initial stage of development process, however they tend to generate false positive results, since the analysis does not take into account code execution paths and runtime conditions. Thus, they can't always be sure about their repair decisions as the results they get are not always verified [3,4].

To circumvent the limitations of static analysis, there have been a number of techniques to symbolic execution and formal verification. Symbolic execution treats inputs to a program symbolically, and can explore numerous execution paths at once. This enables us to discover hidden logical errors and complex situation scenarios which cannot be discovered by pattern analysis. Program assurance can be improved by quantitatively verifying pre-defined security features and contract invariants through formal verification. These methods improve the accuracy of the vulnerability detection process, and offer more accurate guarantees but they also are

computationally intensive. Path explosion does exist and these methods may be scalability issues. For real implementation in some cases, one would require all the domain knowledge and formal specification [2] [8].

In addition to static techniques, there are also dynamic techniques and fuzzing techniques to test the contract's behaviour in real running scenarios. They analyse inputs with intelligent contracts, which are either automatically generated or manually put together, to find runtime vulnerabilities, mistakes and behavioural anomalies. Recent benchmarking frameworks have shown that combination of various testing techniques have shown great improvement in the coverage of vulnerabilities than using a single analytical technique. But, complete covering a complicated IC [5] [9] using dynamic approaches is largely reliant on the quality of the tests and the variety of the inputs.

Smart Contracts Audits are now being enriched with new possibilities thanks to recent advancements in AI (e.g., semantic code understanding, vulnerability classification, documentation and corrective advice automation). Transformer-based models and LLMs have shown promise in comprehending intricate source code and providing security explanations with a developer-centric perspective. The findings of this study, however, indicate that AI is not a guaranteed sign of security and the lack of consistent thinking and in-depth blockchain knowledge could hinder its trustworthiness. AI is therefore seen as a complement to the existing security analysis tools and not a replacement for traditional auditing methods [10] [11].

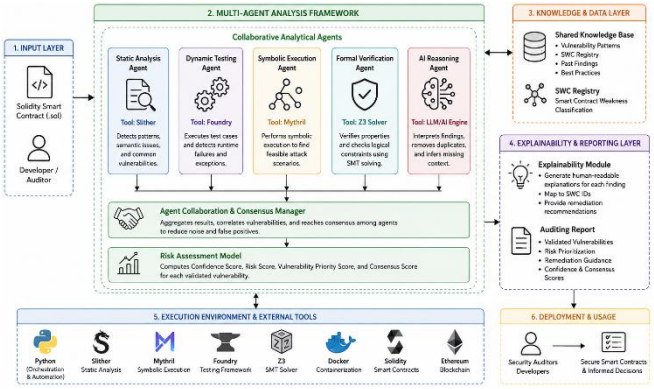
Another emerging research area is in collaborative multi-agent systems and applying them to software security testing. Unlike the sequential execution of standard auditing pipelines of security tools, multi-agent architectures split and spread the analytical duties to specialised agents that can share intermediate knowledge and collaboratively check the security outcomes. The architectures provide flexibility, adaptability and decision consistency, while decreasing redundant vulnerability reports. But previous studies have been either on AI-assisted auditing or multi-agent cooperation without considering their explainability and contextual risk assessment, and their vulnerability validation in a consensus-based framework [12, 13].

Literature reviewed indicates that there are still research gaps. The current smart contract auditing methods largely rely on a single analysis method, resulting in inadequate coverage of vulnerabilities, inconsistencies in severity, and poor explicability. Also, lack of knowledge sharing between the analytical components of the system limits their ability to correlate evidence and, therefore, their certainty of vulnerability detection. These constraints have sparked the development of AMESCAF that combines static analysis, dynamic testing, symbolic execution, formal reasoning, and AI-assisted reasoning in a collaborative fashion. The framework leverages open source security approaches, consensus based validation and context based risk assessment to improve the accuracy of security detections, reduce false positive detections and deliver clear and developer orientated security recommendations.

3. Materials And Methods

A. Overall Framework

The proposed AMESCAF attempts to enhance the security of intelligent contracts and merge a number of free and complimentary auditing approaches together in a single framework. The system is based on different specialised agents for static analysis, dynamic testing, symbolic execution, formal verification, AI assisted reasoning and vulnerability assessment and not on a single analytical technique. Common knowledge-base, on which intermediate information is exchanged between agents, leads to consensus decision mechanism to agree detected vulnerabilities. It also has a context-aware risk assessment approach where it can prioritize the discovered vulnerabilities according to their severity, exploitability, confidence and business implications. Explainability module gives the developer short and succinct explanations and guidelines for repair. Fig.1. represents the comprehensive process of proposed AMESCAF framework. It shows the interplay of the analytical agents and end-to-end smart contract auditing pipeline.



"Fig. 1. System Architecture for AMESCAF"

Fig. 1 shows the structure of the proposed framework. The audit procedure starts with the submission of the intelligent contracts and the preparation and simultaneous execution of the specialised analytic agents. Generated discoveries are validated and integrated with other discoveries in a common knowledge repository using a consensus decision engine. Then it's the risk assessment and explainability modules that list vulnerabilities and give intelligible security advise to developers before it's time to output the final complete auditing report.

B. Multi-Agent Workflow

Our proposed AMESCAF system is structured around a multi-agent pipeline where each agent focuses on a particular complementary security study and shares the intermediate results with the rest of the agents by using a shared knowledge repository. The first step is to acquire the Solidity smart contract and then preprocess it, i.e., generate intermediate representations for analysis. Static Analysis, Dynamic Testing, Symbolic Execution and Formal Verification agents work in parallel to give syntax-level, runtime, logical and specification-based error identification. The AI Reasoning agent incorporates their responses and provides context and ideas for improvement. Then the Correlation module merges redundant data in a consensus manner. The Risk Assessment module ranks the vulnerabilities according to severity, exploitability and confidence and business effect. Finally, the Explainability module gives interpretable arguments and the Report Generation module delivers a security audit report to the developers.

C. Risk Assessment Model

The suggested framework AMESCAF computes a set of assessment metrics on the output of the collaborating analytical agents, to rank the detected vulnerabilities. The metrics are based on the confidence in the detections, the overall security risk, the prioritisation of security vulnerabilities, and the consensus (across agents) in the quality and the ability to explain the conclusions.

Confidence Score (CS): The odds that a vulnerability will be correctly detected. This is based on the number of analysis agents that have detected the same vulnerability.

$$CS = \frac{N_d}{N_t} \qquad (1)$$

where N_d is the number of agents that detect the vulnerability and N_t the number of agents involved.

Risk Score (RS) is the overall severity of a vulnerability, taking into account the technical severity, the exploitability and the business impact.

$$RS = w_s S + w_e E + w_b B \qquad (2)$$

S, E and B are the normalised severity, exploitability and business impact ratings respectively. w_s, w_e, and w_b are their weighting coefficients so that $w_s + w_e + w_b = 1$.

Vulnerability Priority Score (VPS) An ordering of validated vulnerabilities by the calculated risk and confidence score.

$$VPS = \frac{RS}{CS} \qquad (3)$$

The more vulnerabilities that VPS numbers have, the more vulnerabilities that have to be patched now.

Finally, the Consensus Score (ConS) is a measure of the consensus between the analytical agents and it is calculated as

$$ConS = \frac{N_a}{N_t} \quad (4)$$

Given Na number of agents, who agree on the stated vulnerability. The consensus ratings were higher with reliability of final security recommendations. It is also useful in minimizing false positive results.

D. Prototype Implementation

To make the proposed AMESCAF viable, the prototype of the proposed framework was developed. The framework is realized in the core language Python and is used to orchestrate analytical agents as well as the execution of processes. Ethereum-based platform. Solidity is used for banking development and testing of benchmark intelligent contracts. Static vulnerability study We used Slither for the static vulnerability investigation. We included Mythril symbolic execution and security checks. Foundry has provided the test environment to compile, deploy and test intelligent contracts in different scenarios. Problematically, the Z3 SMT Solver for symbolic reasoning and constraint resolution was used to verify the vulnerabilities. To be portable, reproducible and segregated for use, all components are containerised using Docker. Automated vulnerability detection, consensus-based validation, risk assessment and explainable security reporting can all be integrated in a single auditing framework with the help of an integrated technological stack.

E. Algorithm

Algorithm 1: The algorithm of AMSA is presented.

Input: Solidity SC

Output: Execute Static Analysis, Dynamic Testing, Symbolic Execution and Formal Verification agents in parallel.

1. Group and consolidate vulnerabilities that are found by all agents.
2. Data were triangulated and AI-assisted reasoning was used to analyse the results.
3. They are the same vulnerabilities and agree on them by consensus.
4. Find out your confidence level(s)
5. It is called the Score, Risk Score and Vulnerability Priority Score.
6. Provide clear suggestions for remedial work.
7. Auditor's final audit document (R).
8. of results obtained from validation.
9. Back (R)

4. EXPERIMENTAL RESULTS

A. Experimental Setup

In the most common types of vulnerabilities (arithmetic, access control and re-entrancy), the proposed AMESCAF framework was found to be valid with the SmartBugs benchmark data of smart contract data and SWC Registry. The tests were executed using different versions of the Solidity compiler, in order to support different versions of the contracts. The following are evaluated as criteria for a full comparison with the existing smart contract audit tools: Precision, Recall, F1-score, FPR, Analysis Time.

B. Performance Evaluation

A few commonly used smart contract auditing tools are compared to the proposed AMESCAF framework in Table 1. Based on the evaluation, AMESCAF has the highest accuracy, the lowest false-positive ratio, and the highest recall and F1-Score. The performance gains are due to the multiple free analytic approaches, the consensus-based validation and the context-aware risk assessment which collectively boost the reliability of detections, and minimise the amount of false vulnerability reports.

Table 1. Performance Comparison of Smart Contract Auditing Methods

Method	Precision (%)	Recall (%)	F1-score (%)	FPR (%)
Slither	91.8	88.4	90.1	10.6
Mythril	93.1	91.5	92.3	8.4
Security	92.4	90.6	91.5	8.9

SmartCheck	89.6	87.2	88.4	12.8
AMESCAF (Proposed)	96.4	95.2	95.8	4.1

C. Discussion

The findings of the experiments show that the proposed AMESCAF framework is more precise and more recalling than traditional smart contract auditing tools and has significantly fewer false-positive detections. This complementary vulnerability detection is realized by employing static analysis, dynamic testing, symbolic execution and formal verification and the vulnerability detected by different vulnerability analyzers can be ignored before the report is generated through the consensus-based validation process. The AI-driven explainability module also provides developers with easy-to-understand explanations and corrective actions in security reports. Another part of vulnerability management is context aware risk assessment model which prioritises the findings on the basis of severity, exploitability, confidence and business impact which can help in successful remediation planning. But the present prototype is primarily adapted to the smart contract platform Ethereum, which is based on the programming language Solidity. When auditing larger and/or more complex decentralised applications, there are a number of analytical components that may drive up processing costs. Future work includes the optimisation of the execution efficiency and support for other blockchain systems and smart contract languages.

D. Performance Graph

Figure 2 provides a summary of the results of the various smart contract auditing approaches after choosing the evaluation metrics. The proposed AMESCAF algorithm provides the best accuracy, recall, F1-score and the lowest percentage of false positive continually compared with other algorithms. We show that the combination of complementary analytical methodologies, consensus validation and risk assessment with explanation increases the reliability of detection and minimises the number of false vulnerability reports. Hence, the suggested system is more efficient while doing secure smart contract auditing.

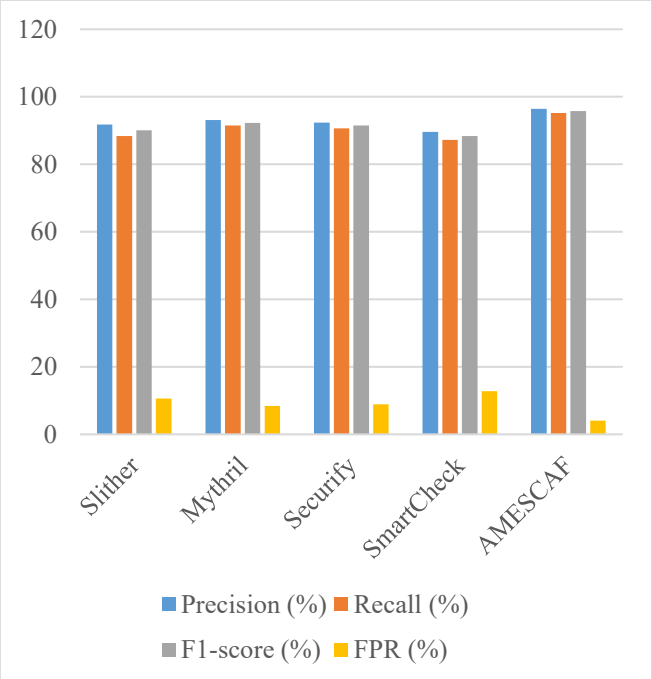


Fig. 2. Performance Comparison

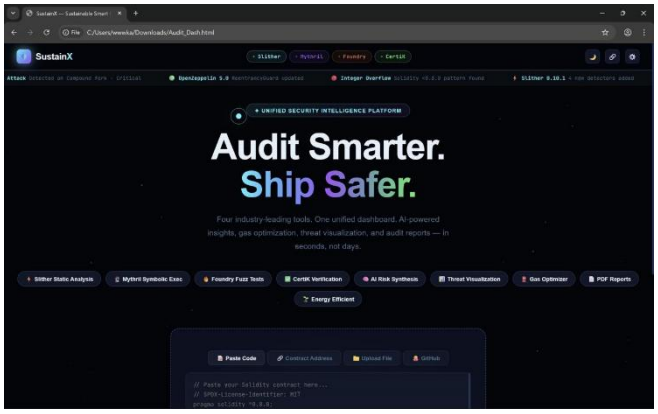


Fig.3 Audit Smarter Ship Safer

Fig. 3. Log in to the SustainX dashboard and get an AI smart contract security auditing interface with vulnerability detection, mythril analysis, foundry fuzz testing, CertiK verification and automatic report production, which enables efficient blockchain security assessment.

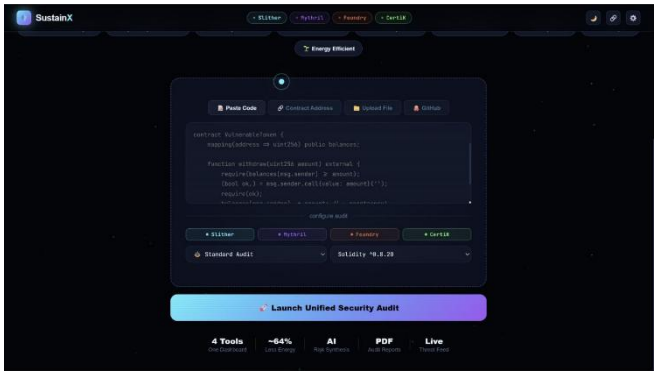


Fig.4 Energy Efficient Page

Fig. 4. It unifies the security audit of the intelligent contracts into one interface, allowing users to write the Solidity smart contract code, to select a range of security analysis tools, to establish the audit rules and to automatically run a thorough security audit and report.

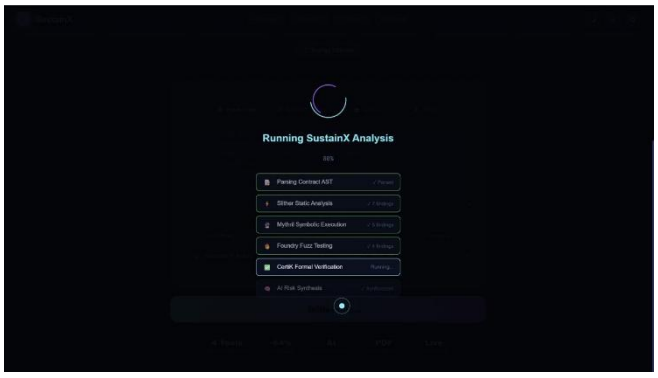


Fig.5 Running SustainX Analysis Page

Figure 5. The analysis execution screen displays the graph and the evolution of the SustainX security audit over the various AST parsing, static analysis, symbolic execution, fuzz testing and formal verification steps in real time.

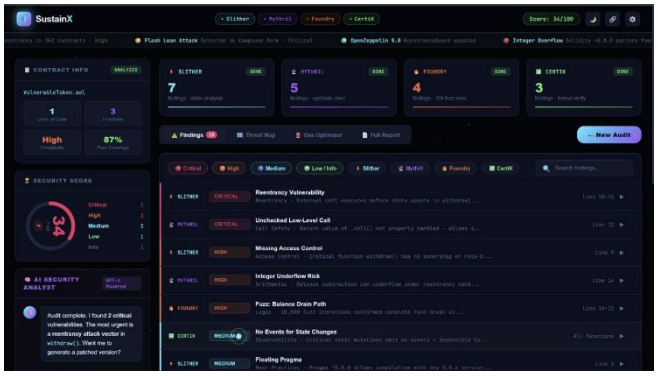


Fig.6 Predicted Result

Figure 6. The audit result dashboard displays the security score of 34% that is derived from the vulnerabilities found in the smart contract. It displays findings according to the severity, tool specific findings, risk analysis and recommendations for actions to enhance secure code.

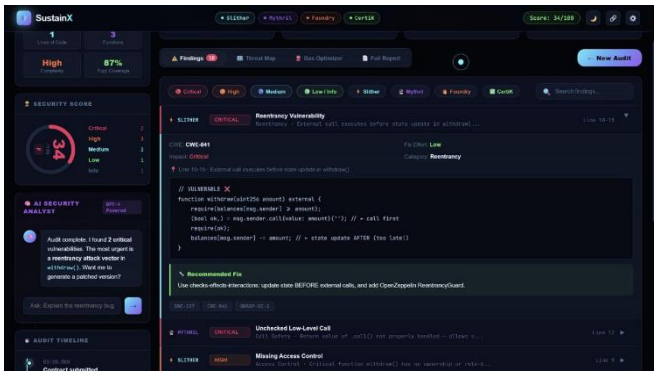


Fig.7 Predicted Result

Figure 7. The thorough vulnerability report shows a high confidence (34%) reentrancy vulnerability and provides the following affected Solidity code, CWE classification, severity, risk explanation and mitigation methods to assist improve the security of intelligent contracts.

5. CONCLUSION

In this study, in an effort to increase the reliability and transparency of the smart contract auditing process, we presented a hybrid security architecture AMESCAF. We introduce a multi-agent environment that is synergistically composed of static analysis, dynamic testing, symbolic execution, formal verification and AI-based reasoning. An analytical agent can provide a validation method and common knowledge store to synthesize the results and minimize duplicate or conflicting vulnerability reporting. The proposed Context-aware Risk Assessment Model also classifies detected vulnerabilities based on technical severity, exploitability, confidence and business impact and the explainability module will explain the reason for the level of risk and will provide suggestions for remediation with the aim of building safe software. The experimental results of the benchmarked intelligent contracts show that the proposed method outperforms the existing smart contract auditing tools by exhibiting higher precision, recall, and F1 Score and lower false-positive rate. The results indicate that, where such complementary analysis approaches are used, they can be combined in one joint analysis framework, thereby significantly increasing the accuracy of detection, which in turn strengthens the confidence of the developers and allows more effective security audits of smart contract systems built on Ethereum.

Future development will be to enable other multi-blockchain systems as Hyperledger Fabric, Solana and Binance smart chain. On top of this, advanced large language models for automatic vulnerability mitigation, computational optimisation of multi-agent coordination and continuous security monitoring for new decentralised apps will be included. The proposed auditing approach will be scaled up and generalised to real world setting via reinforcement learning enabled adaptive agent collaboration and benchmark extension.

REFERENCES

1. Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli, “A Survey of Attacks on Ethereum Smart Contracts (SoK),” in *Principles of Security and Trust: 6th International Conference, POST 2017*, M. Maffei and M. Ryan, Eds., Lecture Notes in Computer Science, vol. 10204. Cham, Switzerland: Springer, 2017, pp. 164–186, doi: 10.1007/978-3-662-54455-6_8.

2. Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor, “Making Smart Contracts Smarter,” in *Proc. 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Vienna, Austria, 2016, pp. 254–269, doi: 10.1145/2976749.2978309.
3. Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin Vechev, “Securify: Practical Security Analysis of Smart Contracts,” in *Proc. 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Toronto, ON, Canada, 2018, pp. 67–82, doi: 10.1145/3243734.3243780.
4. Josselin Feist, Gustavo Grieco, and Alex Groce, “Slither: A Static Analysis Framework for Smart Contracts,” in *Proc. 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, Montreal, QC, Canada, 2019, pp. 8–15, doi: 10.1109/WETSEB.2019.00008.
5. Asem Ghaleb and Karthik Pattabiraman, “How Effective Are Smart Contract Analysis Tools? Evaluating Smart Contract Static Analysis Tools Using Bug Injection,” in *Proc. 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2020, pp. 415–427, doi: 10.1145/3395363.3397385.
6. Christof Ferreira Torres, Julian Schütte, and Radu State, “Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts,” in *Proc. 34th Annual Computer Security Applications Conference (ACSAC)*, San Juan, PR, USA, 2018, pp. 664–676, doi: 10.1145/3274694.3274737.
7. Bernhard Mueller, “Smashing Ethereum Smart Contracts for Fun and Real Profit,” in *Black Hat USA*, 2018.
8. João F. Ferreira, Pedro Cruz, Thomas Durieux, and Rui Abreu, “SmartBugs: A Framework to Analyze Solidity Smart Contracts,” in *Proc. 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Melbourne, Australia, 2020, pp. 1349–1352, doi: 10.1145/3324884.3415298.
9. Monika di Angelo, Thomas Durieux, João F. Ferreira, and Gernot Salzer, “SmartBugs 2.0: An Execution Framework for Weakness Detection in Ethereum Smart Contracts,” in *Proc. 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2023, pp. 2102–2105, doi: 10.1109/ASE56229.2023.00060.
10. Ben Wang, Yanxiang Tong, Shunhui Ji, Hai Dong, Xiapu Luo, and Pengcheng Zhang, “A Review of Learning-Based Smart Contract Vulnerability Detection: A Perspective on Code Representation,” *ACM Transactions on Software Engineering and Methodology*, vol. 35, no. 6, Art. no. 167, pp. 1–46, 2026, doi: 10.1145/3750042.
11. Zhiyuan Wei, Jing Sun, Zijian Zhang, Zhe Hou, Xianhao Zhang, Meng Li, Yuqiang Sun, Daoyuan Wu, Yang Liu, Chunmiao Li, Mingchao Wan, and Jin Dong, “HKT-SmartAudit: Distilling Lightweight Models for Smart Contract Auditing,” *IEEE Transactions on Information Forensics and Security*, vol. 21, pp. 4446–4459, 2026, doi: 10.1109/TIFS.2026.3683274.
12. Yepeng Ding, Ahmed Twabi, Junwei Yu, Lingfeng Zhang, Tohru Kondo, and Hiroyuki Sato, “SEMA: Self-Evolving Multi-Agent Auditing for Smart Contracts,” *Electronics*, vol. 15, no. 10, Art. no. 2187, 2026, doi: 10.3390/electronics15102187.
13. Tamer Abdelaziz, Salma Alsaghir, and Karim Ali, “Where Do Smart Contract Security Analyzers Fall Short?,” in *Proc. 23rd IEEE/ACM International Conference on Mining Software Repositories (MSR)*, Rio de Janeiro, Brazil, 2026, pp. 1–13, doi: 10.1145/3793302.3793338.
14. Ilya Grishchenko, Matteo Maffei, and Clara Schneidewind, “A Semantic Framework for the Security Analysis of Ethereum Smart Contracts,” in *Principles of Security and Trust: 7th International Conference, POST 2018*, Lecture Notes in Computer Science, vol. 10804, Thessaloniki, Greece, 2018, pp. 243–269, doi: 10.1007/978-3-319-89722-6_10.
15. Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov, “SmartCheck: Static Analysis of Ethereum Smart Contracts,” in *Proc. IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, Gothenburg, Sweden, 2018, pp. 9–16, doi: 10.1145/3194113.3194115.
16. Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi, “Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks,” in *Proc. Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, 2019, doi: 10.14722/ndss.2019.23413.
17. Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis, “MadMax: Surviving Out-of-Gas Conditions in Ethereum Smart Contracts,” *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, Art. no. 116, pp. 1–27, 2018, doi: 10.1145/3276486.
18. Lexi Brent, Anton Jurisevic, Michael Kong, Eric Liu, François Gauthier, Vincent Gramoli, Ralph Holz, and Bernhard Scholz, “Vandal: A Scalable Security Analysis Framework for Smart Contracts,” *arXiv preprint arXiv:1809.03981*, 2018.
19. Nami Ashizawa, Naoto Yanai, Jason Paul Cruz, and Shingo Okamura, “Eth2Vec: Learning Contract-Wide Code Representations for Vulnerability Detection on Ethereum Smart Contracts,” *Blockchain: Research and Applications*, vol. 3, no. 4, Art. no. 100101, 2022, doi: 10.1016/j.bcra.2022.100101.
20. Fernando Richter Vidal, Naghmeh Ivaki, and Nuno Laranjeiro, “OpenSCV: An Open Hierarchical Taxonomy for Smart Contract Vulnerabilities,” *Empirical Software Engineering*, vol. 29, Art. no. 101, 2024, doi: 10.1007/s10664-024-10446-8.