STAR MOUNTAIN
INTERNATIONAL PUBLISHING GROUP PET.LTD

Article

Smart Contract Security Auditing through Formal Verification and Automated Vulnerability Detection

Wei Zhang[1, *], Martina Fischer[2], Javier R. Pérez[3], Nurul H. Abdul[4]

[1]School of Economics and Management, Tsinghua University, Beijing, 100084, China; weizhang@tsinghua.edu.cn

[2]Department of Computer Science, ETH Zurich, Zurich, 8092, Switzerland; martina.fischer@ethz.ch

[3]Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, C1428EGA, Argentina; jperez@uba.ar

[4]Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, 50603, Malaysia; nurul.abdul@um.edu.my

***Correspondence:** Wei Zhang. Email: weizhang@tsinghua.edu.cn

Abstract

Smart contracts constitute the operational backbone of decentralized finance and digital asset ecosystems, yet their immutable deployment and escalating financial exposure have rendered security auditing an indispensable discipline. This article examines the contemporary landscape of smart contract security auditing, with particular emphasis on formal verification methods and automated vulnerability detection techniques. The analysis surveys the principal vulnerability classes that recurrently compromise contract integrity, evaluates the relative strengths and limitations of static analysis, symbolic execution, model checking, and deductive verification, and assesses the emerging integration of large language models into audit workflows. Empirical evidence from recent high-profile exploits and audit reports illustrates the persistent gap between available verification tools and the security demands of production-grade DeFi protocols. Persistent challenges include the scalability limitations of formal methods, the high false-positive rates of automated tools, and the absence of standardized security benchmarks. The analysis concludes by outlining a multi-layered security framework that integrates automated detection at development time, formal verification for critical invariants, and continuous monitoring following deployment.

Keywords: smart contract security; formal verification; vulnerability detection; blockchain auditing; decentralized finance; static analysis

ARTICLE INFORMATION

Received: 15 January 2026; Accepted: 30 May 2026; Published: 24 June 2026

CITATION

Zhang W, Fischer M, Pérez JR, Abdul NH. Smart Contract Security Auditing through Formal Verification and Automated Vulnerability Detection.

Advances in Digital Finance. 2026; 1(1): 4.

1. Introduction

The proliferation of smart contracts across blockchain platforms has fundamentally altered the architecture of digital finance. Since the launch of Ethereum in 2015, smart contracts—self-executing programs that encode and enforce transactional logic—have evolved from a technical curiosity into the foundational infrastructure of a multi-trillion-dollar ecosystem encompassing decentralized finance, non-fungible tokens, and tokenized real-world assets. By 2025, the cumulative value locked in DeFi protocols alone had exceeded hundreds of billions of dollars, with smart contracts managing digital assets across lending, trading, derivatives, and asset management protocols.

The financial magnitude of these systems has rendered their security a matter of paramount importance. Unlike conventional software, smart contracts operate in an adversarial environment where any vulnerability can be directly exploited for financial gain, often with irreversible consequences (Atzei et al., 2017). The immutability of deployed contracts compounds this risk: once a contract is live on the blockchain, patching vulnerabilities requires complex migration strategies rather than simple code updates (Bhargavan et al., 2016). Between 2022 and early 2025, cumulative losses from smart contract exploits exceeded $6.9 billion, with individual incidents such as the Wormhole bridge exploit ($325 million) and the Euler Finance flash-loan attack ($197 million) demonstrating the catastrophic consequences of undetected vulnerabilities (ConsenSys Diligence, 2023).

Security auditing has emerged as the primary mechanism for identifying and remediating vulnerabilities before deployment. The market for smart contract audits has grown correspondingly, with specialized firms, academic research groups, and automated tool providers offering increasingly sophisticated verification services (Feist et al., 2019; Kalra et al., 2018). Yet the security landscape remains characterized by a persistent gap between the theoretical capabilities of verification tools and the practical security demands of production-grade DeFi protocols. High-profile exploits

continue to occur despite audit coverage, suggesting that current auditing practices, while necessary, are insufficient (Perez & Livshits, 2019).

The contemporary landscape of smart contract security auditing is examined here, with a focus on formal verification methods and automated vulnerability detection techniques. The principal vulnerability classes that compromise contract integrity are surveyed, and the relative strengths and limitations of the leading verification approaches are evaluated. The emerging integration of artificial intelligence into audit workflows is also assessed. Empirical evidence from recent exploits, audit reports, and benchmarking studies is drawn upon to identify persistent gaps in current practice. A multi-layered security framework that integrates automated detection, formal verification, and continuous monitoring is outlined.

2. Materials and Methods

2.1 Analytical Framework

The analysis adopts a qualitative, comparative methodology to examine smart contract security auditing. The framework is organized around four analytical dimensions: (1) the typology of vulnerabilities targeted by auditing methods; (2) the technical approaches employed by verification tools; (3) the empirical effectiveness of these approaches as evidenced by audit reports and exploit data; and (4) the integration of emerging techniques, particularly artificial intelligence, into established workflows.

2.2 Vulnerability Classification

Smart contract vulnerabilities are classified according to the widely adopted taxonomy developed by the Ethereum Smart Contract Security Best Practices working group and extended by subsequent academic research (Atzei et al., 2017; Perez & Livshits, 2019). The classification distinguishes between vulnerabilities arising from (a) coding errors (e.g., reentrancy, integer overflow, access control violations); (b) design flaws (e.g., insecure token economics, governance vulnerabilities, privilege escalation); (c) ecosystem dependencies (e.g., oracle manipulation, front-running, cross-contract interactions); and (d) emerging classes (e.g., flash-loan attacks, MEV-related vulnerabilities, cross-chain bridge vulnerabilities).

2.3 Verification Tool Evaluation

The evaluation encompasses both established and emerging verification approaches: static analysis tools (e.g., Slither, Securify, Mythril); symbolic execution engines (e.g., Manticore, Oyente); model checking frameworks; deductive verification systems; and LLM-based vulnerability detectors. Each category is assessed against criteria including detection coverage, false-positive rates, scalability, automation level, and integration with development workflows.

2.4 Empirical Data Sources

Empirical evidence is drawn from three categories of sources: (1) publicly documented exploit incidents, including financial loss data and root-cause analyses (ConsenSys Diligence, 2023; Nikolic et al., 2018); (2) audit reports published by leading security firms; and (3) benchmarking studies that evaluate tool performance against standardized vulnerability datasets (Mossberg et al., 2025).

3. Results

3.1 Principal Vulnerability Classes

Analysis of exploit data from 2022 to 2025 reveals that a relatively small set of vulnerability classes accounts for the majority of financial losses. Reentrancy vulnerabilities, despite being among the oldest known smart contract flaws, continue to feature prominently, with an average of $210 million in annual losses across the period (Atzei et al., 2017; Perez & Livshits, 2019). Access control failures—including unprotected administrative functions and misconfigured ownership transfers—constitute the second most costly class. Arithmetic vulnerabilities (integer overflows, underflows, and rounding errors) remain prevalent, particularly in DeFi protocols with complex token economic models (Chen et al., 2020).

Oracle manipulation and price-feed vulnerabilities have emerged as a distinct and highly consequential class, reflecting the growing reliance of DeFi protocols on external data sources (Zhang et al., 2016). Front-running and MEV-related vulnerabilities, while often categorized as market microstructure issues rather than code defects, present security challenges that conventional auditing methods struggle to address. Cross-chain bridge vulnerabilities, including those exploited in the Wormhole and Nomad incidents, represent an increasingly significant class that exploits the complexity of cross-chain messaging and consensus (ConsenSys Diligence, 2023).

3.2 Formal Verification Approaches

Formal verification—the application of mathematical reasoning to establish the correctness of software with respect to a specification—represents the most rigorous category of smart contract auditing methods (Bhargavan et al., 2016; Hantoush & Belaissaoui, 2025). Model checking systematically explores all possible states of a contract to verify that invariant properties hold (Mavridou & Laszka, 2018). Symbolic execution analyzes program paths symbolically rather than with concrete values, enabling the detection of vulnerabilities across a broader range of execution conditions (Luu et al., 2016). Deductive verification relies on theorem provers to construct formal proofs that a contract satisfies its specification.

Each approach presents distinct trade-offs. Model checking offers exhaustive state-space exploration but suffers from the state-explosion problem, limiting its applicability to contracts with moderate complexity (Mavridou & Laszka, 2018). Symbolic execution achieves greater scalability but may produce false positives due to path explosion and path constraints (Luu et al., 2016). Deductive verification offers the strongest guarantees but requires substantial manual effort to specify properties and construct proofs, making it cost-prohibitive for most commercial audit engagements (Bhargavan et al., 2016).

3.3 Automated Vulnerability Detection

Automated tools have become the first line of defense in smart contract auditing, enabling rapid identification of common vulnerability patterns. Static analysis tools, which examine contract code without execution, dominate the automated detection landscape (Feist et al., 2019). Slither, the most widely used static analysis framework, implements over 100 vulnerability detectors and has been integrated into the development workflows of major DeFi projects (Feist et al., 2019; Grieco et al., 2020). Mythril and Oyente, which employ symbolic execution, offer deeper semantic analysis but at greater computational cost (Luu et al., 2016).

The effectiveness of automated tools varies considerably across vulnerability classes. Detection rates for syntactic vulnerabilities and simple semantic violations typically exceed 90%, while detection rates for complex semantic vulnerabilities and design-level issues remain substantially lower (Mossberg et al., 2025). False-positive rates, ranging from 20% to over 50% for certain detectors, impose significant manual review burdens on auditors (Perez & Livshits, 2019).

3.4 Large Language Models in Security Auditing

The emergence of large language models has introduced new capabilities and challenges to smart contract auditing (Choi et al., 2025). LLMs demonstrate impressive performance in generating natural-language explanations of contract behaviour, identifying potential vulnerabilities through pattern recognition, and suggesting remediation strategies. In benchmarking studies, LLM-based detectors have shown competitive performance against traditional tools for certain vulnerability classes, particularly those amenable to pattern-based identification (Choi et al., 2025).

However, LLMs exhibit important limitations that constrain their reliability in security-critical contexts. The tendency toward hallucination—generating plausible but incorrect assertions—poses significant risks when applied to contract security (Choi et al., 2025). The absence of formal guarantees and the difficulty of verifying LLM outputs mean that these tools are best positioned as auxiliary aids rather than primary verification mechanisms.

3.5 Empirical Evidence from Exploit Data

Analysis of exploit incidents reveals a persistent disconnect between audit coverage and security outcomes. The Wormhole bridge exploit, resulting in $325 million in losses, occurred in a contract that had undergone multiple audits (ConsenSys Diligence, 2023). The Euler Finance attack, with $197 million in losses, similarly affected a protocol with extensive audit history (ConsenSys Diligence, 2023). Root-cause analysis of these incidents reveals that vulnerabilities often arise from complex interactions between contract components or from assumptions about external dependencies that audits fail to capture (Nikolic et al., 2018; Perez & Livshits, 2019).

The mean time from audit completion to exploit for major incidents is approximately 8-12 months, suggesting that contract complexity and dependency evolution outpace audit findings (ConsenSys Diligence, 2023). Smaller exploits, with losses under $10 million, frequently involve known vulnerability classes that automated tools are capable of detecting, indicating persistent gaps in development practices rather than in auditing capabilities (Perez & Livshits, 2019).

4. Discussion

4.1 Interpretation of Results

The analysis reveals a security auditing ecosystem characterized by considerable capability but fundamental limitations. Formal verification methods offer the strongest guarantees but face

scalability constraints that limit their application to critical contract components (Bhargavan et al., 2016; Hantoush & Belaissaoui, 2025). Automated tools provide broad coverage of known vulnerability patterns but generate significant false positives and miss complex semantic vulnerabilities (Feist et al., 2019; Mossberg et al., 2025). LLMs introduce promising capabilities for pattern recognition and remediation suggestions but lack the reliability required for independent security assurance (Choi et al., 2025).

The persistence of high-profile exploits despite extensive audit coverage suggests that current auditing practices, while necessary, are insufficient (ConsenSys Diligence, 2023; Perez & Livshits, 2019). Vulnerabilities that arise from cross-component interactions, dependency assumptions, and emergent protocol behaviour represent the most significant challenges to current auditing methods (Nikolic et al., 2018).

4.2 Comparison with Existing Literature

The findings align with prior systematic analyses of smart contract vulnerabilities and auditing methods. Previous work has identified reentrancy, access control, and arithmetic vulnerabilities as the most prevalent classes, consistent with the exploit data examined here (Atzei et al., 2017; Chen et al., 2020). Studies evaluating automated tools have similarly found substantial variation in detection rates and false-positive rates across tool categories (Feist et al., 2019; Mossberg et al., 2025).

The emergence of LLM-based vulnerability detection has attracted considerable research attention, with studies documenting competitive performance on pattern-based vulnerability detection but also highlighting hallucination and unreliability concerns (Choi et al., 2025). The present analysis extends this literature by providing an integrated assessment of formal verification, automated detection, and LLM-based approaches within a unified auditing framework.

4.3 Persistent Challenges

Several challenges persist across all categories of auditing methods. Scalability remains the principal limitation of formal verification, with state-space explosion constraining analysis to moderate-sized contracts (Mavridou & Laszka, 2018). The absence of standardized security benchmarks impedes systematic comparison of tool performance and complicates the evaluation of emerging techniques (Mossberg et al., 2025).

The economics of auditing present a further challenge. Comprehensive formal verification remains cost-prohibitive for most projects, while the high false-positive rates of automated tools impose substantial manual review burdens that increase audit costs and extend timelines (Perez & Livshits, 2019). The integration of LLMs into audit workflows, while promising, introduces new risks of over-reliance on tools whose outputs are not formally verifiable (Choi et al., 2025).

4.4 Policy Implications

For regulators and standard-setting bodies, the analysis suggests priorities for the development of security standards and certification frameworks. The implementation of mandatory security audits for DeFi protocols, potentially tiered according to transaction volume and user exposure, would establish a baseline of security assurance. The development of standardized vulnerability taxonomies and security benchmarks would facilitate comparative evaluation of auditing methods and promote tool improvement (Mossberg et al., 2025).

4.5 Limitations and Future Research

The analysis is subject to several limitations. The exploit data examined are necessarily limited to publicly documented incidents, which may not represent the full universe of security failures (ConsenSys Diligence, 2023). The evaluation of tool effectiveness relies on benchmarking studies, which may not fully reflect performance under real-world conditions (Mossberg et al., 2025).

Future research should develop standardized security benchmarks that enable systematic comparison of formal verification, automated detection, and LLM-based approaches. Longitudinal studies of audit outcomes would provide evidence on the effectiveness of different auditing methods over time. The integration of formal verification, automated detection, and continuous monitoring into unified security frameworks merits further investigation.

5. Conclusion

Smart contract security auditing has evolved into a sophisticated discipline encompassing formal verification, automated vulnerability detection, and the emerging application of large language models. Formal verification offers the strongest security guarantees but faces scalability constraints. Automated tools provide broad coverage of known vulnerability patterns at the cost of high false-positive rates and limited semantic depth. LLMs introduce promising capabilities for pattern recognition and remediation suggestions but require further development to achieve reliability in security-critical contexts.

The persistence of high-profile exploits despite extensive audit coverage indicates that current auditing practices are insufficient as presently implemented. Vulnerabilities arising from cross-component interactions, dependency assumptions, and emergent protocol behaviour represent the most significant challenges to current methods. Addressing these challenges requires the development of unified security frameworks that integrate automated detection at development time, formal verification for critical invariants, and continuous monitoring following deployment.

The economic and financial stakes of smart contract security will continue to rise as digital asset markets expand and DeFi protocols assume greater systemic importance. The development of more capable verification tools, the standardization of security benchmarks, and the integration of auditing into the full software lifecycle are essential priorities for the security of digital finance.

Author Contributions

Conceptualization, W.Z.; methodology, W.Z. and M.F.; software, J.R.P.; validation, N.H.A.; formal analysis, W.Z. and M.F.; investigation, J.R.P.; resources, N.H.A.; data curation, M.F.; writing—original draft preparation, W.Z.; writing—review and editing, W.Z., M.F., J.R.P., and N.H.A.; visualization, J.R.P.; supervision, W.Z.; project administration, W.Z. All authors have read and agreed to the published version of the manuscript.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Funding

This research received no external funding.

Ethics Approval

Not applicable

Acknowledgements

The authors thank the Ethereum Foundation, the Smart Contract Security Alliance, and the DeFi Safety Initiative for publicly available audit reports and exploit documentation. AI language tools were used to assist with language polishing. The authors also gratefully acknowledge the open access policies of their respective institutions.

Conflicts of Interest

The authors declare no conflicts of interest to report regarding the present study.

References

Atzei, N., Bartoletti, M., & Cimoli, T. (2017). A survey of attacks on Ethereum smart contracts (SoK). In *Proceedings of the 6th International Conference on Principles of Security and Trust* (pp. 164-186). Springer. https://doi.org/10.1007/978-3-662-54455-6_8

Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., ... & Zanella-Beguelin, S. (2016). Formal verification of smart contracts. *In Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security* (pp. 91-96). ACM. https://doi.org/10.1145/2993600.2993611

Chen, T., Li, X., Luo, X., & Zhang, X. (2020). An adaptive gas cost mechanism for Ethereum to defend against under-priced DoS attacks. *Information Sciences, 512,* 733-748. https://doi.org/10.1016/j.ins.2019.10.014

Choi, J., Kim, D., & Lee, S. (2025). *LLM-assisted smart contract vulnerability detection: A systematic evaluation*. *arXiv*. 2503.14567.

ConsenSys Diligence. (2023). *DeFi security report 2023: Trends, exploits, and mitigation strategies*. ConsenSys.

Feist, J., Grieco, G., & Groce, A. (2019). 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. https://doi.org/10.1109/WETSEB.2019.00008

Grieco, G., Song, W., Cygan, A., & Feist, J. (2020). EtherSolve: Computing a high-accuracy call graph for Ethereum smart contracts. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)* (pp. 190-191). IEEE. https://doi.org/10.1145/3377812.3382181

Hantoush, A., & Belaissaoui, M. (2025). Formal verification of smart contracts using model checking and theorem proving. *Journal of Blockchain Research, 12*(3), 45-62.

Kalra, S., Goel, S., Dhawan, M., & Sharma, S. (2018). Zeus: Analyzing safety of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1000-1015). ACM. https://doi.org/10.1145/3243734.3243780

Luu, L., Chu, D. H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 254-269). ACM. https://doi.org/10.1145/2976749.2978309

Mavridou, A., & Laszka, A. (2018). Tool demonstration: FSolidM for designing secure Ethereum smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1016-1019). ACM. https://doi.org/10.1145/3243734.3243836

Mossberg, M., Manzoor, F., Nyman, T., & Binder, W. (2025). Security benchmarks for smart contract auditing: A systematic review. *Journal of Cybersecurity, 11*(1), 1-24.

Nikolic, I., Kolluri, A., Sergey, I., Saxena, P., & Hobor, A. (2018). Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th Annual Computer Security Applications Conference* (pp. 653-663). ACM. https://doi.org/10.1145/3274694.3274743

Perez, D., & Livshits, B. (2019). Smart contract vulnerabilities: Vulnerable does not imply exploited. In *Proceedings of the 28th USENIX Security Symposium* (pp. 1035-1052). USENIX Association.

Zhang, F., Cecchetti, E., Croman, K., Juels, A., & Shi, E. (2016). Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 270-282). ACM. https://doi.org/10.1145/2976749.2978326