Automated Smart Contract Repair with Formal Guarantees

Tamer Abdelaziz, Ph.D.

tamer.m@nyu.edu
tamer-abdelaziz.github.io

Abstract

While vulnerability detection has advanced, the lack of reliable automated repair mechanisms leaves smart contracts vulnerable to exploits post-audit. This project presents a hybrid repair framework that combines generative AI with formal verification to synthesize patches guaranteeing functional preservation, vulnerability elimination, and bounded gas overhead. Extending our team's contributions to contract analysis [2, 5, 3, 4, 6, 7], it addresses gaps in recent automated repair literature [8, 9]. The framework will release open-source tools and verified patch corpora.

Problem Formalization

Existing automated repair tools are limited by weak formal guarantees, frequent introduction of new vulnerabilities, and poor gas-efficiency preservation, resulting in patches that developers cannot trust in production. This project solves these by fusing LLM-guided constrained synthesis with relational equivalence checking, multi-property SMT verification, and adversarial mutational testing to produce patches that are provably correct, safe, and efficient.

Detection without reliable repair leaves developers with brittle and risky remediation options. This project develops an automated repair framework that synthesizes patches while guaranteeing that (1) patched contracts preserve intended functional behaviors, (2) they remove targeted vulnerabilities, and (3) they maintain gas-efficiency and composability constraints. The approach fuses generative models (LLMs and program synthesizers) with formal verification (SMT-based equivalence and invariant checking).

We formalize the task as follows. Given a vulnerable contract C (bytecode B and specification Ψ capturing intended behavior) and a vulnerability predicate ϕ_V, produce a patched contract $\hat{C}$ that satisfies three constraints:

$$\begin{aligned}
&\text{(Correctness)} \quad \forall\sigma, \forall\tau \;\; \Psi(\sigma, \tau, C) \Rightarrow \Psi(\sigma, \tau, \hat{C}),\\
&\text{(Safety)} \quad \neg\exists\tau \;\; \phi_V(T(\sigma_0, \tau), \hat{C}),\\
&\text{(Resource)} \quad \mathrm{cost}(\hat{C}) \leq \mathrm{cost}(C) + \epsilon,
\end{aligned}$$

where cost is a gas consumption estimate and ϵ is an allowable overhead.

The repair pipeline composes three modules. First, a vulnerability-localizer identifies the minimal program region R associated with the violation using trace attribution and influence analysis. Second, a constrained synthesizer (guided by LLM proposals) enumerates candidate patches $\{\hat{C}_j\}$ restricted to a syntactic and semantic patch grammar $\mathcal{G}_R$ (e.g., reentrancy guards, access checks, integer-safe arithmetic). Third, a verifier performs (a) functional equivalence checking under the intended specification Ψ for non-affected behaviors and (b) vulnerability absence checking via bounded symbolic execution or SMT proving. Equivalence checking uses relational verification techniques that assert observational equivalence on specified

public interfaces; we employ a bounded equivalence decision procedure $E(\hat{C}, C, \Psi)$ that returns SAT/UNSAT or unknown, and the synthesizer rejects candidates that violate constraints.

To bridge the gap where formal specs Ψ are absent, we adopt specification mining techniques that infer likely invariants and postconditions from test suites and historical execution traces. A reinforcement learning layer optimizes the synthesizer to prefer small, generalizable patches; the reward function balances vulnerability suppression, functional preservation, and gas cost. Formally, for a policy π producing patch proposals, the RL objective maximizes expected reward

$$\mathbb{E}_{\hat{C}\sim\pi}\big[R(\hat{C})\big] = \mathbb{E}\big[w_1 \cdot \mathbf{1}_{\text{safety}}(\hat{C}) + w_2 \cdot \mathbf{1}_{\text{equiv}}(\hat{C}) - w_3 \cdot \Delta\text{cost}(\hat{C})\big].$$

A key research challenge is avoiding introducing new vulnerabilities. We therefore explicitly perform multi-property verification and integrate adversarial mutational testing: after a candidate patch passes the verifier, we use mutation operators and exploit-generation models to attempt to find secondary vulnerabilities; only patches that survive both formal checks and adversarial testing are recommended.

Expected outcomes include an open-sourced synthesizer-verifier pipeline, a corpus of verified patched contracts, and empirical studies comparing human-applied patches versus automated patches with respect to security, correctness and gas overhead. We invite collaborators with expertise in program synthesis, SMT/theorem proving, and large-scale deployment to join the project and co-author evaluation papers; industry partners who are willing to test patches on audit pipelines are particularly welcome.

References

[1] T. Abdelaziz, A. Sedky Adly, B. Rossi, and M.-S. Mostafa. Identification and assessment of software design pattern violations. *Informatics Bulletin*, 1(2):6–13, 2019. https://fcihib.journals.ekb.eg/article_107517_62d89752f7d871844b0e5dd1601da4f5.pdf.

[2] T. Abdelaziz and A. Hobor. Smart learning to find dumb contracts. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1775–1792, 2023. https://www.usenix.org/conference/usenixsecurity23/presentation/abdelaziz.

[3] T. Abdelaziz and A. Hobor. Smart learning to find dumb contracts (extended version). arXiv:2304.10726, 2023. https://arxiv.org/abs/2304.10726.

[4] T. Abdelaziz and A. Hobor. USENIX'23 artifact appendix: Smart learning to find dumb contracts. USENIX Association, 2023. https://www.usenix.org/system/files/usenixsecurity23-appendix-abdelaziz.pdf.

[5] T. Abdelaziz and A. Hobor. Schooling to exploit foolish contracts. In *2023 Fifth International Conference on Blockchain Computing and Applications (BCCA)*, pages 388–395, 2023. https://ieeexplore.ieee.org/document/10338924.

[6] T. A. Abdelmegid Mohamed. Towards secure smart contracts: A deep learning approach for detecting security threats. PhD thesis, National University of Singapore, 2023. https://scholarbank.nus.edu.sg/handle/10635/247301.

[7] T. Abdelaziz, S. Alsaghir, and K. Ali. Where do smart contract security analyzers fall short? *Proceedings of the 2026 IEEE/ACM 23rd International Conference on Mining Software Repositories (MSR)*, 2026.

[8] S. Bobadilla et al. Do Automated Fixes Truly Mitigate Smart Contract Exploits? arXiv:2501.04600, 2025. https://arxiv.org/abs/2501.04600.

[9] R. Kiani et al. Automated Repair of Smart Contract Vulnerabilities: A Systematic Literature Review. Electronics, 2024. https://www.mdpi.com/2079-9292/13/19/3942.

[10] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014. https://ethereum.github.io/yellowpaper/paper.pdf.

[11] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.

[12] N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts (soK). In *Proceedings of the 6th International Conference on Principles of Security and Trust (POST)*, 2017.