Short Communication

SARMF-Bench: A Reproducible Smart Contract Vulnerability Benchmark Dataset

Mohit Tiwari[1]

1. Department of Computer Science and Engineering, Bharati Vidyapeeth's College of Engineering, India

Smart contract vulnerability benchmarking lacks standardized, reproducible datasets that enable fair and consistent evaluation of static analysis tools. This paper presents SARMF-Bench, a compact, deterministic, and fully reproducible benchmark dataset comprising five SWC-aligned Solidity smart contracts (SC01–SC05) covering reentrancy (SWC-107), integer overflow/underflow (SWC-101), access-control weakness (SWC-105), unchecked external calls (SWC-104), and denial-of-service via unbounded loops (SWC-113). Each contract is intentionally minimal to isolate a single structural vulnerability pattern and is paired with machine-readable JSON outputs generated using Slither v0.11.5 in a version-locked environment, preserving detector identifiers, impact levels, and confidence metadata. SARMF-Bench is archived across multiple open repositories with permanent DOIs to enable fully reproducible smart contract security tool evaluation experiments. Baseline static analysis results are reported for each vulnerability class. All artifacts are publicly released under open licenses.

Corresponding author: Mohit Tiwari, mohit.tiwari@bharatividyapeeth.edu

1. Introduction

Smart contracts deployed on the Ethereum blockchain have become critical infrastructure for decentralised finance (DeFi), NFT platforms, and autonomous organisations. However, their immutable nature means that vulnerabilities—once deployed—cannot be patched without redeployment, leading to significant financial losses. The Ethereum ecosystem has witnessed numerous high-profile exploits attributable to well-known vulnerability classes including reentrancy (SWC-107), integer overflow/underflow (SWC-101), and access-control weaknesses (SWC-105) [1][2].

Static analysis tools such as Slither [3], Mythril [4], and Securify [5] have emerged as standard first-pass defences for smart contract auditors. However, fair and reproducible evaluation of these tools remains challenging due to the absence of standardised, version-locked benchmark datasets with machine-readable ground-truth annotations. Existing benchmarks such as SmartBugs Curated [6] and the Consolidated Ground Truth (CGT) [7] provide valuable foundations but do not enforce version-locking of analysis tools or provide structured JSON output for automated evaluation pipelines.

This paper addresses this gap by introducing SARMF-Bench: a minimal, deterministic, and fully reproducible benchmark dataset of five SWC-aligned Solidity smart contracts. The primary contributions of this work are: (1) five carefully crafted Solidity contracts, each isolating a single vulnerability pattern aligned to the SWC Registry; (2) version-locked Slither v0.11.5 analysis results in structured JSON format; (3) multi-repository archiving with permanent DOIs for long-term reproducibility; and (4) a reusable evaluation methodology for smart contract security researchers.

2. Related Work

Several benchmark datasets and evaluation frameworks have been proposed for smart contract security analysis. SmartBugs [8] is an extensible framework that integrates multiple analysis tools and provides a curated dataset of 143 annotated vulnerable contracts (SB Curated). The Consolidated Ground Truth (CGT) extends this to 3,103 contracts with 20,455 manually checked assessments across 10 vulnerability categories [7]. The SmartBugs Wild Dataset provides 47,398 real-world Ethereum contracts, though without ground-truth annotations [6].

Trail of Bits demonstrated that Slither achieves the lowest false positive rate (10.9%) among leading static analysis tools, outperforming Securify (25%), SmartCheck (73.6%), and Solhint (91.3%) in reentrancy detection benchmarks [3]. Empirical studies have further highlighted the challenge of tool interoperability, noting that different tools use incompatible output formats and detector naming conventions, making cross-tool comparison difficult without a standardised evaluation harness [9].

SARMF-Bench differentiates itself from prior work by focusing on minimal, isolated contract specimens with version-locked tool outputs and structured JSON annotations, enabling deterministic and reproducible evaluation pipelines without requiring access to the original tool installations.

3. Benchmark Design

Each contract in SARMF-Bench is designed to isolate a single structural vulnerability pattern aligned to the SWC Registry. Contracts are written in Solidity and compiled with version-appropriate compilers to preserve the intended vulnerability semantics. The benchmark comprises five contracts:

- **SC01 — Reentrancy (SWC-107):** Balance state variable is updated after the external call, enabling reentrant withdrawals. Pattern based on the classic Checks-Effects-Interactions violation.
- **SC02 — Integer Overflow (SWC-101):** Arithmetic overflow in Solidity 0.5.x without SafeMath protection. An unchecked uint addition wraps around to zero on overflow.
- **SC03 — Access-Control Weakness (SWC-105):** The changeOwner() function lacks an access restriction modifier, allowing any caller to take ownership of the contract.
- **SC04 — Unchecked External Call (SWC-104):** The return value of a low-level call() is not validated, allowing silent failures to go undetected.
- **SC05 — Denial-of-Service via Unbounded Loop (SWC-113):** A payout function iterates over an unbounded array of recipients, enabling a gas exhaustion attack by growing the array indefinitely.

4. Methodology

All contracts were analysed using Slither v0.11.5 in a version-locked Docker environment to ensure reproducibility. The analysis pipeline proceeds as follows: (1) each contract is compiled using the Solidity compiler version matching its pragma directive; (2) Slither is invoked with default detector settings and JSON output mode (--json flag); (3) output JSON files are validated for structural completeness; (4) detector findings are mapped to SWC identifiers using the official SWC Registry [10].

The version-locked environment is specified in a requirements.txt file archived alongside the dataset. All analysis was conducted on a reproducible compute environment with full dependency pinning, including Slither v0.11.5, solc-select v1.0.4, and Python 3.10. The complete environment specification and run scripts are available in the GitHub repository and Zenodo archive.

5. Static Analysis Results

Table 1 summarises the Slither analysis results for each contract in SARMF-Bench. All five contracts produced the expected vulnerability findings, confirming that the benchmark contracts exhibit the intended vulnerability patterns under the version-locked analysis environment.

Contract ID	SWC ID	Slither Detector	Impact / Confidence
SC01	SWC-107	reentrancy-eth	High / Medium
SC02	SWC-101	integer-overflow	High / High
SC03	SWC-105	unprotected-upgrade	High / High
SC04	SWC-104	unchecked-lowlevel	Medium / Medium
SC05	SWC-113	costly-loop	Informational / Medium

Table 1. *Slither v0.11.5 analysis results for SARMF-Bench contracts.*

All JSON output files are provided in machine-readable format in the repository, preserving detector names, impact levels, confidence ratings, source file locations, and line numbers. This structured format enables automated evaluation pipelines to parse and compare results without manual inspection.

6. Data Availability and Repository Links

SARMF-Bench is archived across multiple open repositories to ensure long-term availability and citability:

- **Primary code and artifacts (GitHub):** https://github.com/profmohit-edu/sarmf-framework
- **Software snapshot with DOI (Zenodo):** https://doi.org/10.5281/zenodo.18754015
- **Benchmark dataset (Harvard Dataverse):** https://doi.org/10.7910/DVN/0SP3OO
- **Dataset mirror (Mendeley Data, Elsevier):** https://doi.org/10.17632/kd3vcpnn9v.1
- **Open project (OSF):** https://doi.org/10.17605/OSF.IO/EJWDC
- **Reproducibility protocol (protocols.io):** https://doi.org/10.17504/protocols.io.bp216eyxdgqe/v1

7. Discussion

SARMF-Bench provides a minimal but complete foundation for controlled smart contract security tool evaluation. The use of intentionally minimal contracts, each isolating a single vulnerability pattern, reduces confounding factors that arise when evaluating tools on complex real-world contracts with

multiple interacting vulnerabilities. This design choice prioritises analytical clarity over breadth, enabling researchers to attribute detection outcomes unambiguously to specific tool capabilities.

The version-locked analysis environment addresses a critical reproducibility gap in prior benchmarks, where tool version drift over time leads to divergent results when experiments are re-run. By archiving exact tool versions alongside the dataset, SARMF-Bench enables future researchers to reproduce baseline results precisely, facilitating longitudinal comparisons as tools evolve.

A limitation of the current benchmark is its small scale (five contracts, five SWC classes). Future versions will extend the dataset to cover additional SWC classes, include real-world contract instances from Etherscan, and incorporate multi-tool consensus annotations from Mythril and Securify alongside Slither.

8. Conclusion

This paper presented SARMF-Bench, a compact, deterministic, and fully reproducible benchmark dataset for smart contract vulnerability detection research. The benchmark comprises five SWC-aligned Solidity contracts with version-locked Slither v0.11.5 analysis results in structured JSON format. SARMF-Bench is publicly archived across six repositories with permanent DOIs, enabling transparent and reproducible smart contract security research. We invite the research community to use, extend, and contribute to SARMF-Bench as a shared evaluation resource.

Statements and Declarations

AI and Large Language Model Disclosure

AI-assisted writing tools were used to assist in drafting and editing portions of this manuscript. The author takes full responsibility for the accuracy, integrity, and originality of the content.

References

1. ^*Atzei N, Bartoletti M, Cimoli T (2017). "A Survey of Attacks on Ethereum Smart Contracts (SoK)." International Conference on Principles of Security and Trust (POST). Springer. pp. 164–186.*
2. ^*Chen T, et al. (2020). "A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses." ACM Comput Surv. 53(3):1–43.*

3. ^a, ^b *Feist J, Grieco G, Groce A (2019). "Slither: A Static Analysis Framework For Smart Contracts." In Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). IEEE. arXiv:1908.09878.*
4. ^ *Mueller B (2018). "Mythril: A Security Analysis Tool for Ethereum Smart Contracts." https://github.com/ConsenSys/mythril.*
5. ^ *Tsankov P, et al. (2018). "Securify: Practical Security Analysis of Smart Contracts." In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 67–82.*
6. ^a, ^b *Durieux T, et al. (2020). "Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts." In Proceedings of the 42nd International Conference on Software Engineering (ICSE). pp. 530–541.*
7. ^a, ^b *Ferreira J, et al. (2020). "SmartBugs: A Framework to Analyze Solidity Smart Contracts." In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 1349–1352.*
8. ^ *SmartBugs Curated Dataset. https://github.com/smartbugs/smartbugs-curated.*
9. ^ *Salzano F, et al. (2025). "An Empirical Analysis of Vulnerability Detection Tools for Solidity Smart Contracts Using Line Level Manually Annotated Vulnerabilities." arXiv:2505.15756.*
10. ^ *SWC Registry — Smart Contract Weakness Classification and Test Cases. https://swcregistry.io.*

Declarations

Funding: No specific funding was received for this work.

Potential competing interests: No potential competing interests to declare.