




HUNTING DEFI VULNERABILITIES VIA CONTEXT-SENSITIVE CONCOLIC VERIFICATION*

A PREPRINT

 **Yepeng Ding**
The University of Tokyo
Tokyo, Japan
yepengd@acm.org

 **Arthur Gervais**
University College London
London, United Kingdom
arthur@gervais.cc

 **Roger Wattenhofer**
ETH Zurich
Zurich, Switzerland
wattenhofer@ethz.ch

 **Hiroyuki Sato**
The University of Tokyo
Tokyo, Japan
schuko@satolab.itc.u-tokyo.ac.jp

ABSTRACT

Decentralized finance (DeFi) is revolutionizing the traditional centralized finance paradigm with its attractive features such as high availability, transparency, and tamper-proofing. However, attacks targeting DeFi services have severely damaged the DeFi market, as evidenced by our investigation of 80 real-world DeFi incidents from 2017 to 2022. Existing methods, based on symbolic execution, model checking, semantic analysis, and fuzzing, fall short in identifying the most DeFi vulnerability types. To address the deficiency, we propose Context-Sensitive Concolic Verification (CSCV), a method of automating the DeFi vulnerability finding based on user-defined properties formulated in temporal logic. CSCV builds and optimizes contexts to guide verification processes that dynamically construct context-carrying transition systems in tandem with concolic executions. Furthermore, we demonstrate the effectiveness of CSCV through experiments on real-world DeFi services and qualitative comparison. The experiment results show that our CSCV prototype successfully detects 76.25% of the vulnerabilities from the investigated incidents with an average time of 253.06 seconds.

Keywords Vulnerability finding · Smart contracts · Decentralized finance · Program analysis · Concolic verification

1 Introduction

Decentralized Finance (DeFi) has been marred by significant security challenges. In our investigation, 80 real-world DeFi incidents that occurred between November 2017 and December 2022 have proved to be considerable threats to the stability of the DeFi market, resulting in financial damages ranging from 2,400 to 600 million dollars. We classify their underlying vulnerabilities into six types based on their root causes and sort their severity regarding the average loss in US dollars per incident, as detailed in Table 1.

Unfortunately, the three most severe root causes, accounting for the largest financial losses, pose a formidable challenge for existing methods. **BF** (e.g., the Eleven Finance hack in 2021) and **RE** variants (e.g., the Rari Capital hack in 2022), which exploit conformance issues between requirement specifications and implementations, render highly automated methods (Frank et al. [2020], Nguyen et al. [2020]) ineffective, while other methods (Choi et al. [2021], So et al. [2021]) struggle with scalability when addressing these non-patterned vulnerabilities. Besides, **PM** vulnerabilities offer attackers lucrative arbitrage opportunities with minimal attack costs via flash loans, yet current methods based on control flow (Frank et al. [2020]), data flow analysis (Choi et al. [2021]), symbolic execution (So et al. [2021]), and fuzzing (Nguyen et al. [2020]) fall short in identifying these plausibly normal financial behaviors.

*This is the preprint version of the conference paper presented in *International Conference on Software Engineering, 2024*.

Table 1: Investigated Typical DeFi Vulnerabilities

Root Cause	Description	Total	Loss
BF	Business Logic Flaw	23	\$1.4B
RE	Reentrancy	9	\$147M
PM	Price Oracle Manipulation	20	\$92M
IV	Insufficient Validation	10	\$12M
AF	Access Control Flaw	13	\$14M
UE	Unexpected External Call	5	\$3M

Motivated by addressing the challenge, we propose Context-Sensitive Concolic Verification (CSCV) to automatically find all classified types of DeFi vulnerabilities by user-defined temporal properties. Concolic verification distinguishes itself from standard concolic execution by leveraging formal verification, effectively synergizing concolic execution (Baldoni et al. [2018]) with model checking (Clarke et al. [2018]).

2 CSCV in a Nutshell

In concolic verification of a temporal property ϕ specifying DeFi service \mathfrak{P} , finding a vulnerability is framed as searching an attack vector, an alternating sequence comprising global states \vec{S} and invocation of functions selected from set F of external write functions, such that $\exists s \in \vec{S} : s \not\models \phi$. However, the brute-force enumeration of all possible combinations of state variables and functions to identify an attack vector is computationally infeasible, given the enormous number of potential permutations. Therefore, we formulate CSCV by introducing contexts to guide concolic verification processes that dynamically construct context-carrying transition systems in tandem with concolic execution during transitions to find attack vectors. The overview of CSCV is visually shown in Figure 1.

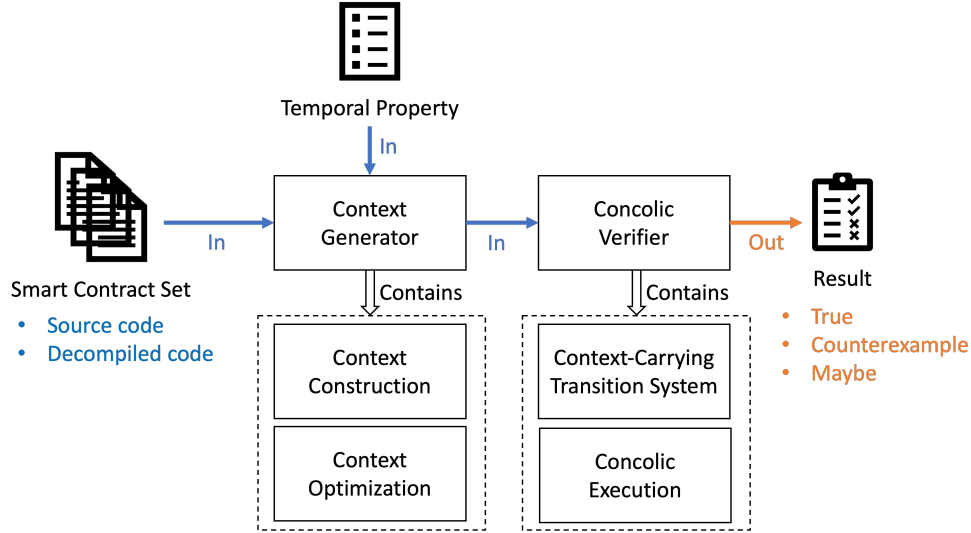


Figure 1: Overview of CSCV.

Context Construction The context generator processes \mathfrak{P} along with ϕ to construct a context that encapsulates an **evaluation function** of state variables at a specific block, and a **relevance function** hierarchizing F for each $f \in F$. The context construction is governed by a property-based algorithm that filters out extraneous information regarding ϕ . The algorithm formulates the smallest evaluation function by analyzing the state variable dependency and shapes the relevance function by ranking the self-excluded F based on the number of commonly shared state variables.

Context Optimization **Property spatialization** mitigates the state explosion problem in temporal property verification by encoding temporal formulas into non-temporal assertions that are coded as preconditions and postconditions into respective functions in F . **Function constantization** simplifies the execution logic by substituting free nullary read function calls with constants based on the dependency analysis regarding modified state variables. **Heuristic**

identification accelerates concolic verification processes by pinpointing heuristics obtained from properties, business logic, and historical incidents in context elements.

Concolic Verification Concolic verification dynamically constructs a context-carrying transition system that steers concolic executions. Each conditional transition corresponds to an external write function invocation, where conditions are derived from the preconditions encoded from ϕ . The nondeterminism of transition selection is resolved by the relevance function in contexts. Each transition also initiates a concolic execution following the shortest execution path from the initial state to the current state. If a postcondition is violated, the concolic verifier reports a counterexample and terminates. In cases when the completeness threshold is guaranteed, the concolic verifier confirms that ϕ holds. Otherwise, the concolic verifier terminates within a predefined time frame or diameter.

3 Evaluation

We mechanized CSCV into a prototype primarily in Java and backed by the Z3 solver for experiments.

Effectiveness Our experiments are designed to selectively utilize a specific proportion (0%, 25%, 50%, and 75%) of heuristics, randomly chosen from an established heuristic base for each investigated DeFi vulnerability in Table 1. The experiment results show that our prototype successfully identified 38 (47.50% of the total) vulnerabilities and 432 attack vectors, including all six classified vulnerability types, even with 0% heuristics. Moreover, with 75% heuristics, our prototype identified 61 vulnerabilities (76.25% of the total) and 1,498 attack vectors, including 20.96% of previously unknown attack vectors, with an average time of 253.06 seconds.

Comparison Our qualitative evaluation offers a comparison between the CSCV methodology and existing methods across a set of criteria, including: vulnerable function path finding (PF), malicious assignment generation (AG), code-level property specification (CP), protocol-level property specification (PP), cross-contract analysis (CC), and DeFi-focus analysis (DF). The evaluation results, detailed in Table 2, demonstrate the potential of CSCV in addressing the challenge of effectively identifying various DeFi vulnerabilities.

Table 2: Feature comparison with existing methods.

Method	PF	AG	CP	PP	CC	DF
sFuzz (Nguyen et al. [2020])	●	●	○	○	○	○
ETHBMC (Frank et al. [2020])	○	●	●	●	●	○
Smartian (Choi et al. [2021])	○	●	●	○	○	○
SmarTest (So et al. [2021])	○	●	●	○	○	○
CSCV	●	●	●	●	●	●

●: full support, ●: partial support, ○: no support.

References

- Joel Frank, Cornelius Aschermann, and Thorsten Holz. {ETHBMC}: A Bounded Model Checker for Smart Contracts. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2757–2774, 2020. ISBN 1-939133-17-3.
- Tai D. Nguyen, Long H. Pham, Jun Sun, Yun Lin, and Quang Tran Minh. sfuzz: An efficient adaptive fuzzer for solidity smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 778–788, 2020.
- Jaeseung Choi, Doyeon Kim, Soomin Kim, Gustavo Grieco, Alex Groce, and Sang Kil Cha. SMARTIAN: Enhancing smart contract fuzzing with static and dynamic data-flow analyses. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 227–239. IEEE, 2021. ISBN 1-66540-337-3.
- Sunbeom So, Seongjoon Hong, and Hakjoo Oh. SmarTest: Effectively Hunting Vulnerable Transaction Sequences in Smart Contracts through Language Model-Guided Symbolic Execution. In *USENIX Security Symposium*, pages 1361–1378, 2021.
- Roberto Baldoni, Emilio Coppa, Daniele Cono D’elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *ACM Computing Surveys (CSUR)*, 51(3):1–39, 2018. ISBN: 0360-0300 Publisher: ACM New York, NY, USA.
- Edmund M Clarke, Thomas A Henzinger, Helmut Veith, Roderick Bloem, et al. *Handbook of model checking*, volume 10. Springer, 2018.