Smart Contract Coverage Report

SiloCore v2

November 2024

Prepared for
Silo Team

certora

Silo

Table of Content

Essential Prerequisites

This document is a supplementary report to Certora's comprehensive smart contract audit report. Please review the primary audit report first to fully understand the findings discussed here.

Using the Certora Prover, we conducted a code coverage specification and verification for all **Silo Contracts in V2** that were in scope. This report outlines the tested code segments, examined attack vectors, key findings, and assumptions. The verification took place between **November 4th and November 25th, 2024.**

Through formal verification, the Certora Prover confirmed that the **Solidity** contract implementations adhere to the formal rules established by the Certora team to mitigate various attack vectors. During this analysis, our team identified several bugs in the contracts, which are thoroughly documented in the audit report.

Fuzzing vs. Symbolic Execution Coverage

Both fuzzing and formal verification (FV) use rules to verify code correctness, but their approaches and coverage depth differ fundamentally. Fuzzing tests only a subset of possible execution paths by generating random inputs. While this method can find bugs, it cannot guarantee complete coverage.

Formal verification, in contrast, mathematically proves that a rule holds for all possible execution paths and states. FV exhaustively verifies every possible scenario for each rule we write. The key distinction lies in measurement: with fuzzing, we track how many execution paths were tested, while with FV, we achieve 100% of relevant path coverage for each rule. Instead, FV ensures our rules cover all possible variations and edge cases in the code. This mathematical approach makes FV inherently more comprehensive than fuzzing.

Contract Tested

This section highlights the smart contracts analyzed during the assessment and the corresponding number of rules verified for each contract.

Contract Name	Total Rules Verified
DebtShareToken	**22**
CollateralShareToken	**16**
SiloO	**41**
SiloConfig	**21**
LiquidationModule	**40**

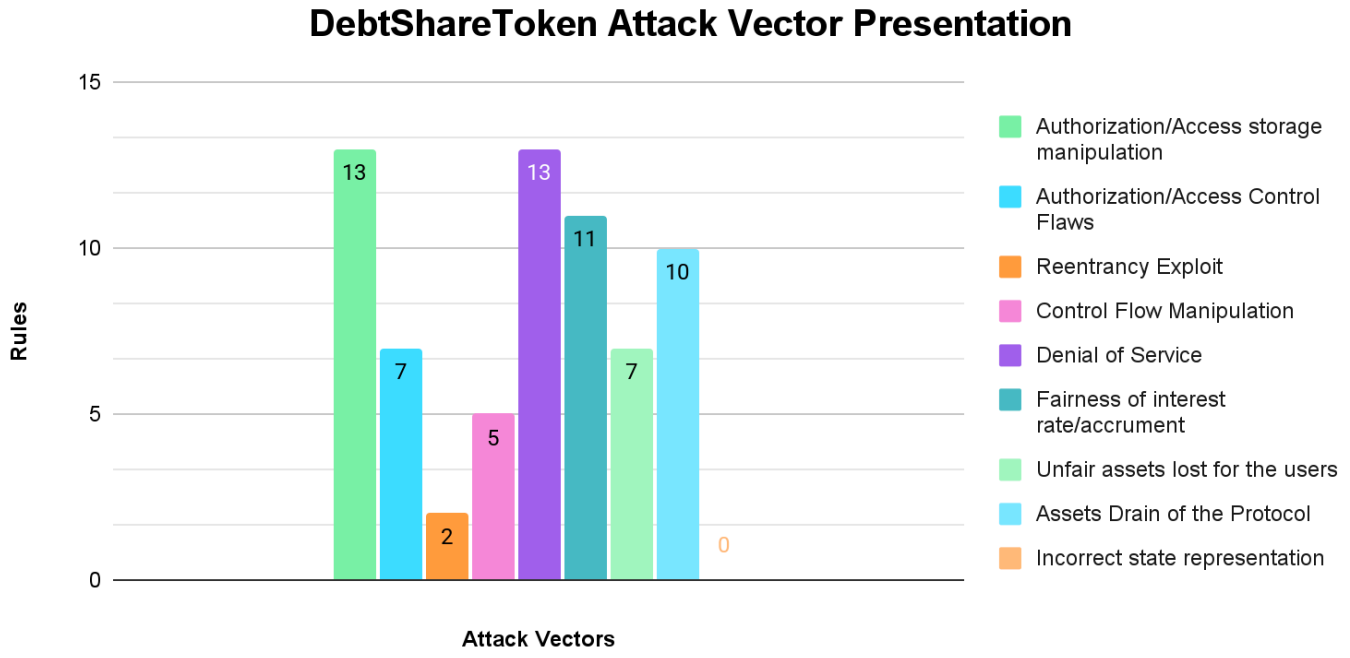
Attack Vector Analysis

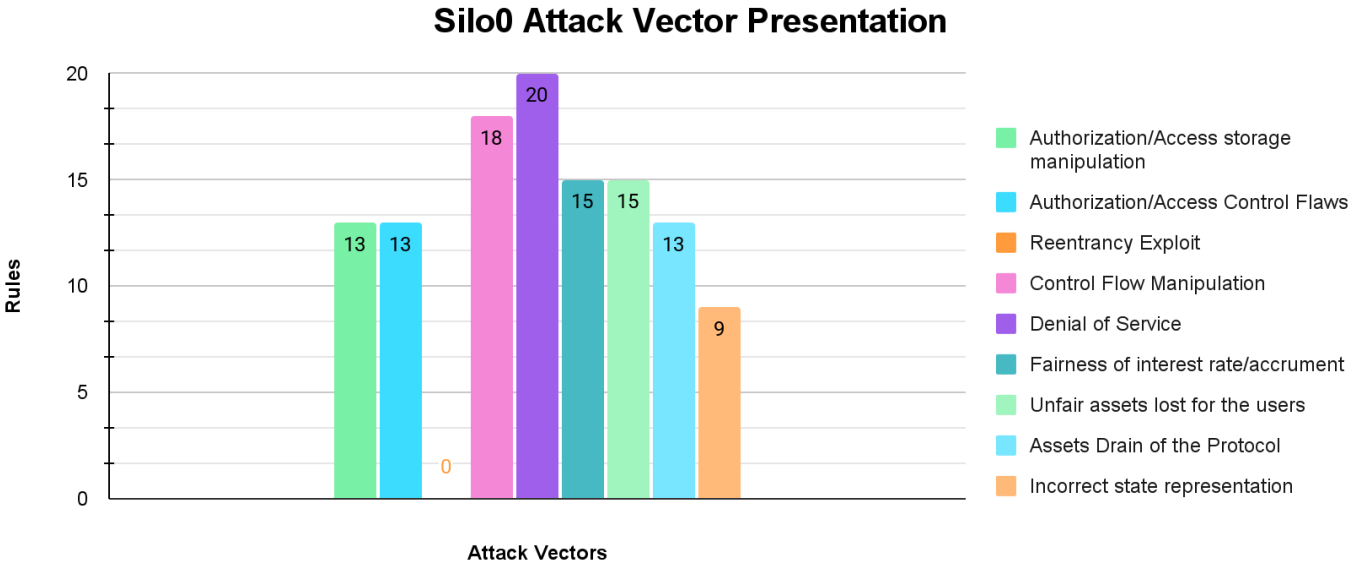
The table below summarizes the tested attack vectors and the number of rules verified for each type, highlighting the extensive efforts made to ensure robust smart contract security.

Sr. No	Identified Attack Vectors	Description	Rule Count
1	Authorization/Access storage manipulation	Focuses on unauthorized access and data manipulation risks	**28**
2	Authorization/Access Control Flaws	Highlights vulnerabilities in access control mechanisms	**18**
3	Reentrancy Exploit	Examines risks of reentrant calls in smart contracts	**9**
4	Control Flow Manipulation	Analyzes disruption of intended control flow	**38**
5	Denial of Service	Investigates potential service disruption attacks	**31**
6	Fairness of interest rate/accrument	Assesses fairness in financial terms and rates	**27**
7	Unfair assets lost for the users	Looks into unfair asset loss for users	**34**
8	Assets Drain of the Protocol	Evaluate risks of protocol draining	**25**
9	Incorrect state representation	Evaluate potential discrepancies in the contract's state	**16**

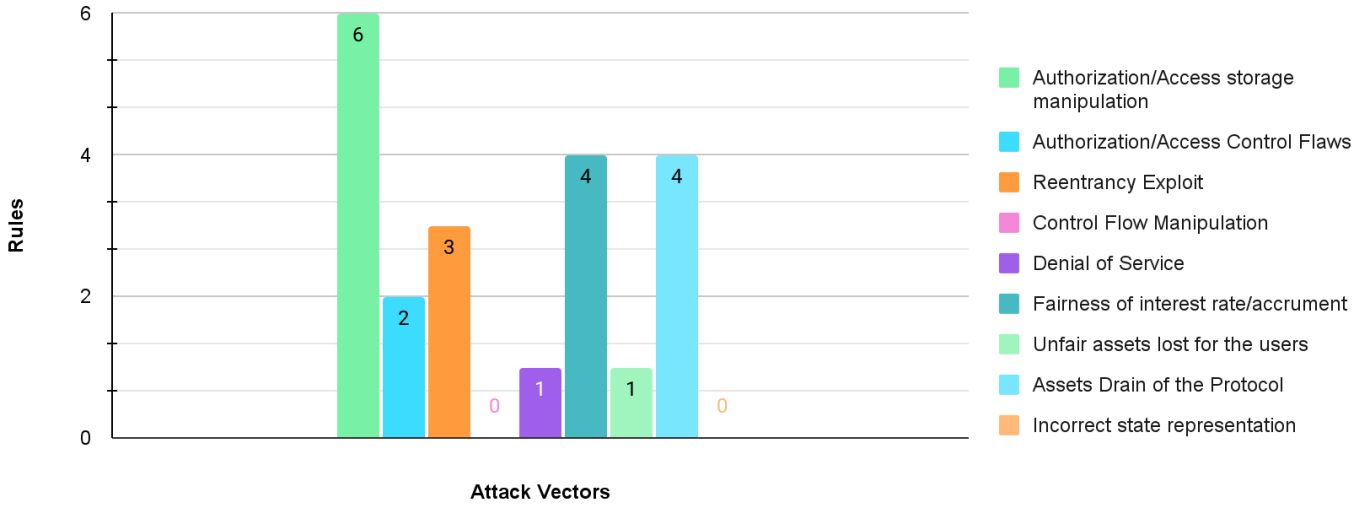
Property Coverage Analysis

A contract-wise bar graph illustrating the distribution of tested attack vectors, emphasizing the relevant vectors and their corresponding properties validated for each contract during the security assessment.

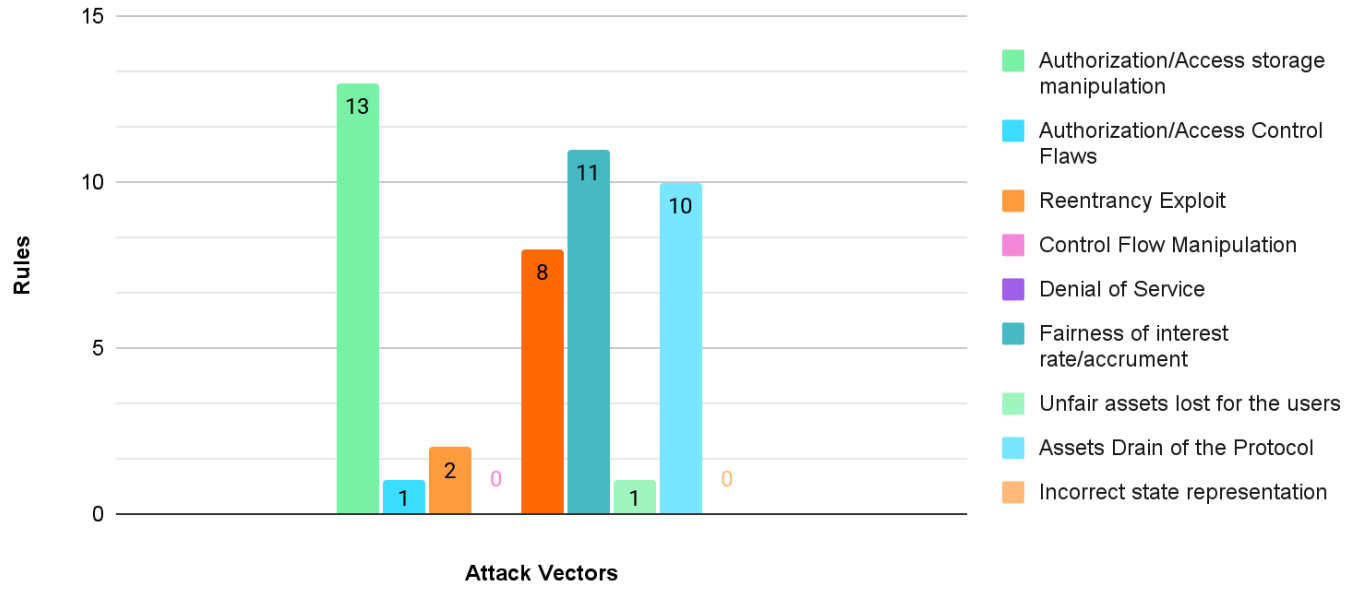




SiloConfig Attack Vector Presentation



CollateralShareToken Attack Vector Presentation

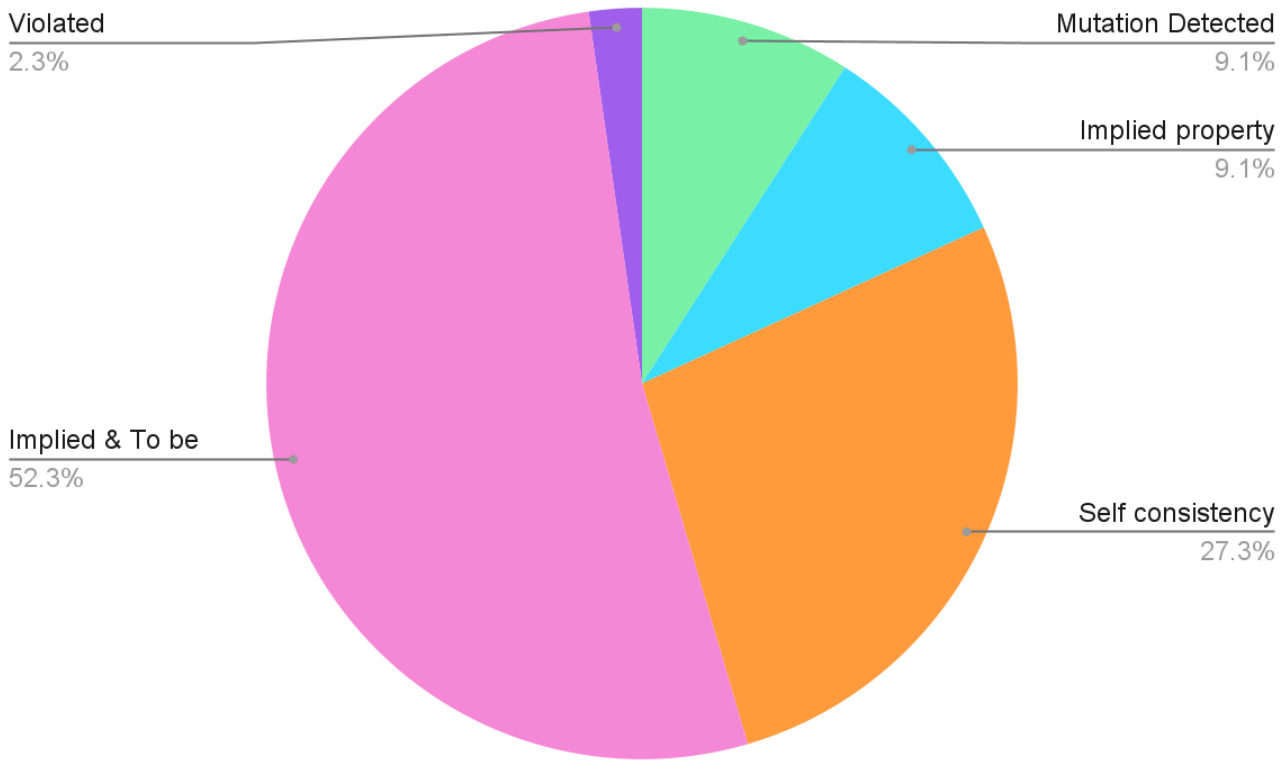


Additional Rules Analysis

The additional rules encountered during testing are presented and categorized into mutations, implied properties, self-consistency issues, and items requiring further verification. These rules represent aspects not captured in the graphs above and highlight key areas for potential improvements in the contract's functionality.

Below, we provide an overview of these rules and their implications:

- **Mutations**: Instances where state changes deviate from expected behavior.
- **Implied Properties**: Assumptions or constraints inferred but not explicitly defined in the contract.
- **Self-Consistency Issues**: Situations where internal contract logic contradicts itself.
- **Verification Required**: Rules or outcomes that require further validation to confirm their correctness.



Scope and Coverage

Our formal verification of Silo Contracts V2 used Certora's Prover to analyze security and verify correctness rules across all in-scope Solidity contracts. While it excluded front-end components, project infrastructure, key custody, and deployment parameter validation, our focus remained on core contract functionality.

Please note that a few more formal rules are not included in this report, as they were proven with an unreleased version of the Certora Prover. Once those rules are proven on a released version of the Certora Prover, we will add them to the next version of this document.