Custodia Security

Size v1.8 Review

Conducted By: Ali Kalout, Ali Shehab

Contents

1. Disclaimer

A smart contract security review cannot ensure the absolute absence of vulnerabilities. This process is limited by time, resources, and expertise and aims to identify as many vulnerabilities as possible. We cannot guarantee complete security after the review, nor can we assure that the review will detect every issue in your smart contracts. We strongly recommend follow-up security reviews, bug bounty programs, and on-chain monitoring.

2. Introduction

Custodia conducted a security assessment of Size's smart contract following the implementation of v1.8, ensuring its proper implementation.

3. About Size

Size is a lending marketplace with unified liquidity across maturities.

Size is built on an order book model where offers are expressed as yield curves, allowing efficient and continuous pricing of fixed-rate products while maintaining unified liquidity.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1. Impact

- High: Results in a substantial loss of assets within the protocol or significantly impacts a group of users.
- Medium: Causes a minor loss of funds (such as value leakage) or affects a core functionality of the protocol.
- Low: Leads to any unexpected behavior in some of the protocol's functionalities, but is not critical.

4.2. Likelihood

- High: The attack path is feasible with reasonable assumptions that replicate on-chain conditions, and the cost of the attack is relatively low compared to the potential funds that can be stolen or lost.
- Medium: The attack vector is conditionally incentivized but still relatively likely.
- Low: The attack requires too many or highly unlikely assumptions, or it demands a significant stake by the attacker with little or no incentive.

4.3. Action required for severity levels

- Critical: Must fix as soon as possible
- High: Must fix
- Medium: Should fix
- Low: Could fix

5. Security Assessment Summary

Duration: 26/05/2025 - 29/05/2025
Repository: SizeCredit/size-solidity
Commit: daf1d1d8db21ae7c62df35fcef4f99ed0a914f69

- src/*
- script/*

6. Executive Summary

Throughout the security review, Ali Kalout and Ali Shehab engaged with Size's team to review Size. During this review, 4 issues were uncovered.

Findings Count

Severity	**Amount**
Critical	N/A
High	N/A
Medium	3
Low	2
Total Finding	**5**

Summary of Findings

ID	Title	Severity	Status
M-01	`getCollectionMarkets` assumes Base-specific markets exist — script will always revert on Ethereum	Medium	Resolved
M-02	Inconsistent APR source between validation and execution in `LiquidateWithReplacement`	Medium	Resolved
M-03	`reinitialize` relies on HTTP-fetched user state but cannot be safely paused — leads to potential migration inconsistencies	Medium	Resolved
L-01	`totalSupply` can be inflated if shares are sent directly to the vault	Low	Acknowledged
L-02	`callMarket` does not support payable calls, preventing ETH deposits into markets via factory	Low	Acknowledged

7. Findings

7.1. Medium Findings

[M-01] `getCollectionMarkets` assumes Base-specific markets exist — script will always revert on Ethereum

Severity:
Medium

Description:
The protocol is deployed only on Base and Ethereum mainnet, but the logic in `getCollectionMarkets()` implicitly assumes it's always running on Base by enforcing that exactly four collection markets exist with hardcoded collateral symbols:
`require(j == 4, "Invalid number of collection markets");`
This function filters markets based on whether their `underlyingCollateralToken.symbol()` matches one of:

- WETH
- cbBTC
- cbETH
- wstETH

However, these markets are part of the only collection that currently exists — and only on Base. The Ethereum deployment does not have any collection markets, meaning that `j` will always be `0`, and the script will always revert with:
`Invalid number of collection markets`

Recommendations:
Modify `getCollectionMarkets()` to conditionally apply the `require(j == 4)` check only when running on Base. For Ethereum, bypass the check and allow an empty or partial result.

[M-02] Inconsistent APR source between validation and execution in `LiquidateWithReplacement`

Severity:
Medium

Description:
In `validateLiquidateWithReplacement`, the borrow APR is fetched using `getUserDefinedBorrowOfferAPR`, which reads from the user's own curve and ignores the collection curve.
`state.getUserDefinedBorrowOfferAPR(params.borrower, tenor);`

However, in `executeLiquidateWithReplacement`, the borrow APR is calculated using `getBorrowOfferRatePerTenor`, which uses the `collectionId` and rateProvider, potentially returning a completely different value.

This mismatch creates a risk where a borrower passes validation using a favorable user-defined curve but ends up being charged a higher rate from the collection curve during execution. This breaks the expectation that validation guarantees the behavior of execution, and can lead to failed transactions or mispriced loans.

Recommendations:
Replace `getUserDefinedBorrowOfferAPR` with `getBorrowOfferAPR` in the validation function. This aligns validation with execution and ensures the same rate computation logic is used throughout.

[M-03] `reinitialize` relies on HTTP-fetched user state but cannot be safely paused — leads to potential migration inconsistencies

---

Severity:
Medium

Description:
The `reinitialize` function in `SizeFactory` performs a one-time setup that migrates a fixed set of users into a new collection. These users are fetched off-chain via an HTTP call and passed into the contract via a proposal submitted to a Safe.

This creates a critical timing assumption: the user list must remain unchanged between the moment it is fetched and when the upgrade is executed. However, due to the asynchronous nature of Safe workflows, users may join the collection in between — leading to inconsistencies and missed migrations.

Pausing the protocol to prevent further user changes would be a natural mitigation, but it is not possible here because the reinitialize function internally calls `buyCreditLimitOnBehalfOf` and `sellCreditLimitOnBehalfOf`, which revert if the protocol is paused.

As a result, the migration is exposed to a race condition: if a user joins after the HTTP snapshot but before the transaction is executed, they will be silently excluded from the new collection, with no way to recover them into the migration.

Recommendations:
Introduce an explicit mechanism to temporarily freeze collection enrollment without pausing the entire protocol.

7.2. Low Findings

[L-01] `totalSupply` can be inflated if shares are sent directly to the vault

---

Severity:
Low

Description:
The `totalSupply` function in `AaveAdapter` returns the `aToken.balanceOf(address(tokenVault))`, which reflects the total underlying held by the vault. However, this does not guarantee alignment with the sum of all user balances tracked via `sharesOf(...)` in the vault.

Since `sharesOf` is manually managed, a user (or contract) could transfer additional scaled aTokens to the vault address directly, inflating the totalSupply result without updating individual balances.

This issue is also present in `ERC4626Adapter`.

Recommendations:

Keep track of the `sum(sharesOf)` for the total supply, for more accurate accounting.

[L-02] `callMarket` does not support payable calls, preventing ETH deposits into markets via factory

---

Severity:
Low

Description:
The `callMarket` function in `SizeFactory` was introduced to allow users to batch interactions with multiple `Size` markets through the factory contract. It enables composing multicalls such as subscribing to rate providers, borrowing across different collaterals, or performing other protocol actions in a single transaction.

However, the current implementation of `callMarket` is not marked as payable.

This creates a silent limitation: users cannot call
`market.deposit{value: ...}(...)`
through the factory, even though the `deposit` function in the `ISize` implementation is explicitly marked as `payable`. The lack of the `payable` modifier on `callMarket` prevents any ETH from being forwarded to the market, causing such calls to revert.

Recommendations:

Mark `callMarket` as `payable` to allow forwarding ETH to the called market.