Custodia Security

SIR 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 SIR's smart contract, ensuring its proper implementation.

3. About SIR

SIR offers a new way to take leverage in DeFi: compounding exposure without the usual drag. Instead of funding or maintenance fees that slowly eat returns, SIR charges one fixed fee only when you open a position.

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: 12/07/2025 - 26/07/2025
Repository: SIR-trading/Core
Commit: e16dbb75925dc0132af10d2eb41033fc2025cf31

- src/*

6. Executive Summary

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

Findings Count

Severity	**Amount**
Critical	N/A
High	N/A
Medium	N/A
Low	3
Informational	1
Total Finding	**4**

Summary of Findings

ID	Title	Severity	Status
L-01	Missing dividend distribution on unstake and claim	Low	Acknowledged
L-02	Lack of permissionless fallback for distributing auction proceeds	Low	Acknowledged
L-03	Incorrect fee assumptions when starting a new auction	Low	Resolved
I-01	Vault fee withdrawal getter omission	Info	Acknowledged

7. Findings

7.1. Low Findings

[L-01] Missing dividend distribution on unstake and claim

Severity:
Low

Description:
The `unstake` and `claim` functions both rely on the current value of `stakingParams.cumulativeETHPerSIRx80` to compute the user's entitled dividends. However, they do not call `_distributeDividends()` beforehand to flush any idle ETH or WETH sitting in the contract. As a result, if there are undisbursed dividends—e.g. WETH sent directly to the contract or ETH from auction payouts—the next user to unstake or claim will not receive their fair share of rewards.

Users who unstake or claim while the contract holds unprocessed dividend funds (e.g., ETH or WETH) will permanently miss out on a portion of their rewards, especially if no one calls `collectFeesAndStartAuction` before them. This edge case breaks reward accounting and violates the fairness expectation that rewards should reflect all dividends currently held by the contract.

Recommendations:
Call `_distributeDividends()` at the start of both unstake and claim to ensure up-to-date dividend state before computing user entitlements.

[L-02] Lack of permissionless fallback for distributing auction proceeds

---

Severity:
Low

Description:
Currently, only the winning bidder is allowed to call `getAuctionLot()` to finalize the auction and trigger dividend distribution. This creates a dependency on the winner's timely action.
If the winner delays claiming their auction lot, the corresponding ETH dividends are withheld from stakers until the cooldown period (10 days) ends and a new auction starts via `collectFeesAndStartAuction`. In the meantime, users can unstake and exit the protocol, potentially receiving less than their fair share of dividends due to outdated `cumulativeETHPerSIRx80`.

If the winner delays claiming the lot, dividends are not distributed promptly, and users who unstake during that period may receive fewer ETH rewards than they are entitled to.

Recommendations:
Allow anyone (not just the winner) to distribute the bid winnings after the auction duration passes, and trigger dividend distribution.

[L-03] Incorrect fee assumptions when starting a new auction

Severity:
Low

Description:
The contract uses `vault.withdrawFees(token)` to determine whether a new auction can be started and emits the withdrawn amount as the `feesToBeAuctioned`. However, this check can fail to account for excess token balances already present in the contract. If the previous auction's winner was blacklisted or never claimed their lot, the `_payAuctionWinner` call will silently revert, and the unclaimed tokens will remain stuck in the contract. Since `withdrawFees()` returns zero in that case, a new auction cannot be initiated, despite tokens still being present and ready to be auctioned.

This can cause auctions to stall indefinitely if no new fees are accrued and the contract holds stale tokens from a previous unclaimed or failed auction payout. Additionally, when a new auction is eventually started, the `AuctionStarted` event may emit an inaccurate `feesToBeAuctioned` value, which reflects only the withdrawn amount and not the total lot being auctioned (which includes the stale balance).

Recommendations:
Instead of relying solely on the result of `withdrawFees()`, the logic should check the contract's actual token balance.

7.2. Informational Findings

[I-01] Vault fee withdrawal getter omission

Description:

The `withdrawFees(address token)` function computes fees owed to SIR stakers as the surplus of the contract's balance over the `totalReserves[token]`. However, the contract does not expose a public view function to query the withdrawable fees for a given token prior to execution. This creates unnecessary opacity and makes it harder to track available rewards for stakers.

Recommendations:

Introduce a new getter function that calculates and exposes the accumulated staker fees.