SIR-Trading Security Review

Auditors

armormadeofwoe, Security Researcher

Amar Fares, Security Researcher

May 3rd, 2025

1 Executive Summary

Over the course of 10 days in total, SIR-Trading engaged with Syzygy to review SIR-Trading.

Summary

Project Name	SIR-Trading
Repository	Core
Commit	a25347f
Type of Project	DeFi, Leveraged Trading
Review Timeline	April 24th, 2025 to May 3rd, 2025

Total Issues

Severity	Count
Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	2

Contents

2 Introduction

SIR brings a fresh approach to leveraged investing in DeFi, offering compounding returns without the usual drawbacks. Unlike traditional approaches to leverage, SIR does away with maintenance fees and removes volatility decay.

3 Disclaimer

Smart contract audits are constrained by time, resources, and expertise. They combine automated analysis with expert manual review to detect vulnerabilities. The Syzygy team makes all efforts to find as many vulnerabilities as they can in the given time period, but cannot guarantee that all issues are found.

4 Findings

4.1 Medium Risk

4.1.1 Front-running allowance changes of APE results in double spending

Context: APE.sol#L87-L93

Description:

The protocol's native **APE** token uses an approve method which is vulnerable to double spending attacks:

```
    function approve(address spender, uint256 amount) external returns (bool) {
        allowance[msg.sender][spender] = amount;

        emit Approval(msg.sender, spender, amount);

        return true;
    }
```

If a user front-runs the owner's call to approve a new amount by spending the previous one, the user will be able to spend more tokens once again after the approval tx passes.

Recommendation: Change the function or add a separate function for when users want to increase an allowance:

```
+   function increaseAllowance(address spender, uint256 amount) external
 ↪  returns (bool) {
+       uint256 currentAllowance = allowance[msg.sender][spender];
+       allowance[msg.sender][spender] = currentAllowance + amount;
+
+       emit Approval(msg.sender, spender, amount);

+       return true;
+   }
```

SIR-Trading: Fixed in 1763614.

4.1.2 No bidding minstep

Context: Staker.sol#L433-L463

Description:

During the auctioning process, users can make bids on certain token auctions and all proceeds from the auctions are then distributed as dividends to stakers of the protocol:

```
    function bid(address token, uint96 amount) external {
        unchecked {
            SirStructs.Auction memory auction = _auctions[token];

            // Unchecked because time stamps cannot overflow
            if (block.timestamp >= auction.startTime +
             ↪  SystemConstants.AUCTION_DURATION) revert NoAuction();

            // Transfer the bid to the contract
 >>         _WETH.transferFrom(msg.sender, address(this), amount);

            if (msg.sender == auction.bidder) {
                // If the bidder is the current winner, we just increase the
                 ↪  bid
                totalWinningBids += amount;
>>              amount += auction.bid;
            } else {
                // Return the previous bid to the previous bidder
                totalWinningBids += amount - auction.bid;
                _WETH.transfer(auction.bidder, auction.bid);
            }
            ...
        }
    }
```

Currently, there is no minstep increase requirement on auction bidding. Due to this, a user can fron-run another user's bid with `amount + 1 wei` which will revert the bid for the original bidder due to `BidTooLow()`.

Recommendation: Introduce a % of current bid as minstep for bids.

SIR-Trading: Fixed in 421f18f.

4.2 Low Risk

4.2.1 Storing contributions in a mapping will save gas

Context: `Contributors.sol#L9-L63`

Description:

The way contributors are fetched in the system is by using an `if/else if` lad-

der. This is inefficient, poorly scalabe and maintainable.

Recommendation:

A better way (both for efficiency and gas savings) would be to store the contributions in a mapping and fetch them from it.

SIR-Trading: Fixed in 2625a97.

4.2.2 Gas optimization during auction bidding

Context: `Staker.sol#L433-L463`

Description:

During bidding on auctions, a check is made for whether `amount <= auction.bid` and reverts if it is. The check can be put earlier in the flow in order to save on gas instead of towards the end like currently done.

```
function bid(address token, uint96 amount) external {
    unchecked {
        SirStructs.Auction memory auction = _auctions[token];

        // Unchecked because time stamps cannot overflow
        if (block.timestamp >= auction.startTime +
        ↪  SystemConstants.AUCTION_DURATION) revert NoAuction();

        // Transfer the bid to the contract
        _WETH.transferFrom(msg.sender, address(this), amount);

        if (msg.sender == auction.bidder) {
            // If the bidder is the current winner, we just increase the bid
            totalWinningBids += amount;
            amount += auction.bid;
        } else {
            // Return the previous bid to the previous bidder
            totalWinningBids += amount - auction.bid;
            _WETH.transfer(auction.bidder, auction.bid);
        }

        /** If the bidder is not the current winner, we check if the bid is
        ↪  higher.
            Null bids are no possible because auction.bid >=0 always.
        */
>>      if (amount <= auction.bid) revert BidTooLow();

        // Update bidder & bid
        _auctions[token] = SirStructs.Auction({bidder: msg.sender, bid: amount,
        ↪  startTime: auction.startTime});

        emit BidReceived(msg.sender, token, auction.bid, amount);
    }
}
```

Recommendation:

The check should instead be put between the auction start time check and WETH transfer taking place. A little refactoring of the code would also be needed for when the bidder is also the current winner.

SIR-Trading: Acknowledged.