Tren Finance

SMART CONTRACTS REVIEW

zokyo

November 19th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passed a security audit.

ZOKYO AUDIT SCORING TREN FINANCE

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 3 Critical issues: 3 resolved = 0 points deducted
- 0 High issues: 0 points deducted
- 1 Medium issue: 1 acknowledged = - 4 points deducted
- 1 Low issue: 1 resolved = 0 points deducted
- 0 Informational issues: 0 points deducted

Thus, 100 - 4 = 96

TECHNICAL SUMMARY

This document outlines the overall security of the Tren Finance smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Tren Finance smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 3 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Tren Finance team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Tren Finance repository:
Repo: https://github.com/Tren-Finance/Tren-Single-Liquidity-Provider

Last commit - 9f87910defd2c498788b864491d295efc1c47ebe

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- SingleLiquidityProvider.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Tren Finance smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

The `SingleLiquidityProvider` contract allows users to deposit stablecoins, manage liquidity in a private vault, and earn rewards in the form of a secondary token. It interacts with an external vault to facilitate adding and removing liquidity, while using ERC20 tokens to represent shares. The contract provides functions for depositing, withdrawing, and claiming rewards, along with mechanisms to track and manage user requests. The rewards are distributed over time based on the amount of liquidity provided, with precision to calculate reward shares. The owner can manage certain contract parameters, including deposit suspension, TREN token configuration, and emergency withdrawals. The contract incorporates security measures such as reentrancy protection and checks for zero amounts.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Tren Finance team and the Tren Finance team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Missing updateRewardPerShare() Call in claimRewards()	Critical	Resolved
2	Pending Rewards Is Not Checked in _checkDepositRemoval()	Critical	Resolved
3	Hardcoded Token Decimals	Critical	Resolved
4	Risk of Centralisation	Medium	Acknowledged
5	Missing Revert in renounceOwnership	Low	Resolved

CRITICAL-1 | RESOLVED

Missing `updateRewardPerShare()` Call in `claimRewards()`

The `claimRewards()` function does not call `updateRewardPerShare()`, which is responsible for updating the reward calculations. This omission can result in the user receiving no rewards or less than they are entitled to, as the rewards calculation might not reflect the most recent state of the contract.

Recommendation:
Call the `updateRewardPerShare()` function before calculating the rewards in the `claimRewards()` function to ensure the rewards are accurately updated before transferring them to the user.

CRITICAL-2 | RESOLVED

Pending Rewards Is Not Checked in `_checkDepositRemoval()`

The `_checkDepositRemoval()` function only checks if the `depositedAmount`, `totalAmountToWithdraw`, and `rewards` are zero before removing the user deposit record. It does not verify that `pendingRewards` is also zero. As a result, if a user still has pending rewards, the function could inadvertently delete the user's deposit record, causing the user to lose their rewards.

Recommendation:
Modify the condition in `_checkDepositRemoval()` to also check that `pendingRewards` is zero. This ensures that the deposit record is only removed when all funds, including pending rewards, have been fully processed, preventing the loss of rewards.

CRITICAL-3 | RESOLVED

Hardcoded Token Decimals

The contract currently does not retrieve the decimals value dynamically from the token contract but instead relies on hardcoded decimals. This is problematic because different tokens can have different decimal places, and hardcoding the decimals could lead to inaccuracies or unexpected behavior, particularly when interacting with tokens that have non-standard decimals.

Recommendation:
Instead of hardcoding the decimals, the contract should retrieve the decimals value directly from the token contract by calling the `.decimals()` function. This would ensure the contract works correctly with any ERC20 token, regardless of its decimal precision.

MEDIUM-1 | ACKNOWLEDGED

Risk of Centralisation

The current implementation grants significant control to the owner through multiple functions that can alter the contract's state and behavior. This centralization places considerable trust in a single entity, increasing the risk of potential misuse.

If the owner's private key is compromised, an attacker could execute any function accessible to the owner, potentially leading to fund loss, contract manipulation, or service disruption.

Recommendation:
To enhance security and reduce the risk of a single point of failure, it is recommended to implement a multi-signature wallet for executing owner functions.

LOW-1 | RESOLVED

Missing Revert in `renounceOwnership`

The `renounceOwnership` function is overridden, but it does not include any logic to prevent ownership renouncement. If a user calls this function, it will remove ownership from the contract. This could cause a severe issue, as all `onlyOwner` restricted functions would no longer work, effectively locking the contract or making it ungovernable.

Recommendation:
The function should revert to prevent the contract from being misowned

	SingleLiquidityProvider.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Tren Finance team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Tren Finance team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo