Teahouse

SMART CONTRACTS REVIEW

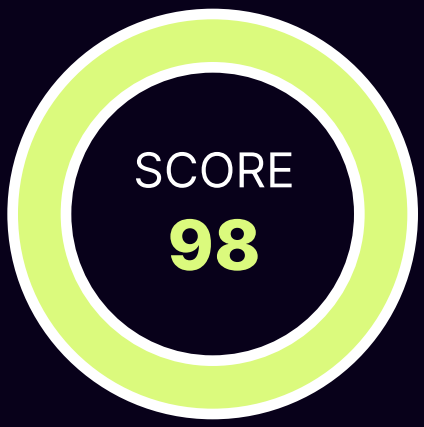
zokyo

October 25th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.



ZOKYO AUDIT SCORING TEAHOUSE 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:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 2 Medium issues: 1 resolved and 1 acknowledged = - 2 points deducted
- 0 Low issues: 0 points deducted
- 2 Informational issues: 2 resolved = 0 points deducted

Thus, 100 - 2 = 98

TECHNICAL SUMMARY

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

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

Contract Status

There were 0 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 Teahouse 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 Teahouse Finance repository:
Repo: https://github.com/TeahouseFinance/TeaVaultAlgebraV1Point9
Initial commit - 1c664099e47669134ff5d72dafb54e9abcf93f95
Last commit - 8647b7fed5271ebbd3f4eda3ffb06997712b4dcb

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

- ./contracts/interface/ITeaVaultAlgebraV1Point9Factory.sol
- ./contracts/interface/ISwapRelayer.sol
- ./contracts/interface/IAlgebraPoolFactory.sol
- ./contracts/interface/IAlgebraPool.sol
- ./contracts/interface/ITeaVaultAlgebraV1Point9.sol
- ./contracts/test/MockToken.sol
- ./contracts/SwapRelayer.sol
- ./contracts/library/VaultUtils.sol
- ./contracts/TeaVaultAlgebraV1Point9Factory.sol
- ./contracts/TeaVaultAlgebraV1Point9.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 Teahouse 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/s by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

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 Teahouse Finance team and the Teahouse 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	Insufficient Access Controls on collectPositionSwapFee Will Allow Anyone to Call This Function	Medium	Resolved
2	Users Could be Frontrun Should They Swap with an Approved swapRouter via the SwapRelayer Contract Due to a Lack of Token Transfer	Medium	Acknowledged
3	Deadline Checks Can be Done After Function Implementation for Better Code Readability	Informational	Resolved
4	Dead code contained in TeaVaultAlgebraV1Point9	Informational	Resolved

MEDIUM-1 | RESOLVED

Insufficient Access Controls on collectPositionSwapFee Will Allow Anyone to Call This Function

Description:

According to the nat spec for the TeaVaultAlgebraV1Point9 contract, the fund manager should only be allowed to call `collectPositionSwapFee` however, there appears to be no access controls placed on the function implementation which may allow anyone to call this function.

Recommendation:
Add the `onlyManager` modifier to the `collectPositionSwapFee` function for the TeaVaultV1Point9 contract.

MEDIUM-2 | ACKNOWLEDGED

Users Could be Frontrun Should They Swap with an Approved swapRouter via the SwapRelayer Contract Due to a Lack of Token Transfer

Description:

The swap relayer contract is responsible for facilitating swaps against whitelisted swap routers which is correct however, the `swap` function does not transfer funds to the contract and expects external entities to first transfer in order to swap. Should an EOA or contract integrator attempt to make a swap by first transferring to the contract, they will get frontrun as the balance of the contract is transferred to the msg.sender.

Recommendation:
It's recommended that the external entity makes a call to approval, where the swapRelayer will make a pull from the user to facilitate a swap. Once a transfer to the contract has happened, the contract can make steps to swap funds and return the amount out to the user.

Comment: We are acknowledging this finding so users know that if this contract is used by users or contract integrators, they will be susceptible to frontrunning.

INFORMATIONAL-1 | RESOLVED

Deadline Checks Can be Done After Function Implementation for Better Code Readability

Description:

The TeaVaultAlgebraV1Point9 contract uses a deadline checker to assert that transactions cannot be maliciously executed by validators; however, the check can be done after the execution of the implementation as the implementation may take additional time to execute.

Recommendation:
It's recommended that the `checkDeadline` modifier in the TeaVaultAlgebraV1Point9 Contract is modified to the following which first executes the implementation then makes a deadline check:

```
modifier checkDeadline(uint256 _deadline) {
    _;
    _checkDeadline(_deadline);
}
```

INFORMATIONAL-2 | RESOLVED

Dead code contained in TeaVaultAlgebraV1Point9

Description:

There exists code in the TeaVaultAlgebraV1Point9 contract `_abs()` which is never executed and is an internal function. Additional unnecessary code may contribute to the size of the contract which will require an unnecessary expenditure of gas to deploy.

Recommendation:
It's recommended that the `_abs()` function is removed from the aforementioned contract.

	./contracts/interface/ITeaVaultAlgebraV1Point9Factory.sol **./contracts/interface/ISwapRelayer.sol** **./contracts/interface/IAlgebraPoolFactory.sol** **./contracts/interface/IAlgebraPool.sol** **./contracts/interface/ITeaVaultAlgebraV1Point9.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

	./contracts/test/MockToken.sol **./contracts/SwapRelayer.sol** **./contracts/library/VaultUtils.sol** **./contracts/TeaVaultAlgebraV1Point9Factory.sol** **./contracts/TeaVaultAlgebraV1Point9.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 Teahouse 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 Teahouse Finance team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo