ZeroLend

SMART CONTRACTS REVIEW

zokyo

November 26th 2024 | v. 1.0

Security Audit Score

PASS

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

ZOKYO AUDIT SCORING ZEROLEND

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: 0 points

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 4 Medium issues: 3 resolved and 1 unresolved = - 10 points deducted
- 4 Low issues: 4 unresolved = - 20 points deducted
- 3 Informational issues: 3 unresolved = 0 points deducted

Thus, 100 - 10 - 20 = 70

TECHNICAL SUMMARY

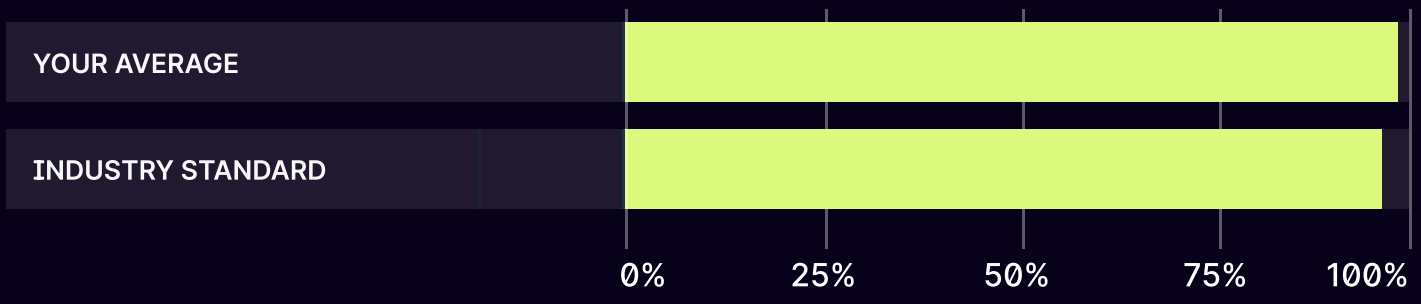
This document outlines the overall security of the ZeroLend smart contract/s evaluated by the Zokyo Security team.

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

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

Testable Code



100% of the code is testable, which is above the industry standard of 95%.

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 ZeroLend 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 ZeroLend repository:
Repo: https://github.com/zerolend/custom-atokens
Initial commit - d62cc50182ee5c530acc74738059b1565ba93081
Fixes: https://github.com/zerolend/custom-atokens/pull/3

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

- ./pendle/ATokenPendleLP.sol
- ./pendle/ATokenPendlePT.sol
- ./core/TokenEmissionsStrategy.sol
- ./aerodrome/ATokenAerodromeLP.sol
- ./mahaxyz/ATokenMahaStaker.sol
- ./interfaces/IAerodromeGauge.sol
- ./interfaces/IMahaStakingRewards.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 ZeroLend 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. Part of this work includes writing a test suite using the Foundry testing framework. 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.	03	Testing contract/s logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contract/s by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

ZeroLend's smart contracts implements customized AToken functionality tailored for distinct assets and reward mechanisms within DeFi ecosystems like Pendle, MahaDAO, and Aerodrome. Each contract adapts AToken to control reward accumulation and address specific asset requirements. For example, ATokenPendleLP and ATokenPendlePT manage Pendle LP and PT tokens, ensuring PENDLE rewards collection from Pendle Market and restricting minting near token expiry to maintain lifecycle security. ATokenMahaStaker integrates MahaDAO's dual-reward system, securing emissions through strict approval processes and an updated emissions manager, while ATokenAerodromeLP enhances Aerodrome LP tokens by interacting with the Aerodrome gauge for seamless reward collection.

The TokenEmissionsStrategy contract coordinates emissions across all these AToken implementations, leveraging an emission management system to allocate rewards efficiently. It restricts access to whitelisted addresses, communicates emissions data to the emission manager, and enforces time-based notification limits to prevent spamming, ensuring optimal reward distribution. Additionally, TokenEmissionsStrategy allows each aToken to notify the emission manager of available rewards through an on-chain mechanism, enhancing transparency and control over the incentive structure. Together, these contracts create a robust and efficient framework for managing rewards and emissions, offering a tailored and secure approach to asset handling in DeFi environments.

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 ZeroLend team and the ZeroLend 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 call to _disableInitializers	Medium	Resolved
2	Approve is Incompatible with Non-Standard ERC20 Tokens	Medium	Resolved
3	Missing gap to Avoid Storage Collisions	Medium	Resolved
4	Unchecked transfers return value	Medium	Unresolved
5	Centralization Risk	Low	Unresolved
6	Lack of Two-Step Ownership Transfer	Low	Unresolved
7	Owner can renounce ownership	Low	Unresolved
8	Missing Emergency Exit for Rewards Distribution	Low	Unresolved
9	Lack of Detailed Event Emissions for Critical Functions	Informational	Unresolved
10	Time-based Emission Notification Manipulation	Informational	Unresolved
11	Floating Pragma	Informational	Unresolved

MEDIUM-1 | RESOLVED

Missing call to _disableInitializers

The `TokenEmissionsStrategy` contracts are using the `Initializable` module and doesn't have a constructor that calls `_disableInitializers()`.

An uninitialized contract can be taken over by an attacker. This applies to both a proxy and its implementation contract, which may impact the proxy. To prevent the implementation contract from being used, you should invoke the `_disableInitializers()` function in the constructor to automatically lock it when it is deployed, more information can be found here.

Recommendation:
Consider calling `_disableInitializers()` in the constructor.

MEDIUM-2 | RESOLVED

Approve is Incompatible with Non-Standard ERC20 Tokens

The `approve` function used in the contract is incompatible with some non-standard ERC20 tokens that do not correctly implement the EIP20 standard. For example, tokens like USDT on Ethereum return `void` instead of a success boolean, causing transactions to revert when interacting with these tokens. Additionally, some tokens require setting the allowance to `0` before a new `approve` call, and failing to do so results in the transaction reverting.

This incompatibility can lead to transaction failures when attempting to approve or interact with non-compliant tokens.

Recommendation:
To handle non-standard ERC20 tokens, it is recommended to use OpenZeppelin's `SafeERC20` library and specifically implement the `forceApprove()` function. This approach ensures proper handling of non-compliant tokens by supporting cases where the `approve` function behaves differently or requires additional steps like resetting the allowance to zero.

MEDIUM-3 | RESOLVED

Missing `gap` to Avoid Storage Collisions

The `TokenEmissionsStrategy` contract is intended to be an upgradeable smart contract, but do not have a __gap variable.

In upgradeable contracts, it's crucial to include a `_gap` to ensure that any additional storage variables added in future contract upgrades do not collide with existing storage variables. This is especially important when inheriting from multiple upgradeable contracts.

Recommendation:
Include a `_gap` as the last storage variable to `TokenEmissionsStrategy` contract to reserve space for future storage variables and prevent storage collisions. This is a common practice to ensure compatibility and avoid issues when upgrading the contract in the future.

MEDIUM-4 | UNRESOLVED

Unchecked transfers return value

In the `TokenEmissionsStrategy` smart contract, ERC20 token transfers are performed using the transfer and transferFrom methods. These methods return a boolean value indicating the success of the operation, as per the ERC20 standard. However, the contract does not check these return values, which can lead to scenarios where token transfers fail.

Recommendation:
Use `SafeERC20` library from `OpenZeppelin`, which handles these inconsistencies and ensures compatibility.

LOW-1 | UNRESOLVED

Centralization Risk

The smart contracts grant significant control to the contract owners through several functions. This centralization poses a substantial risk, as it places considerable trust and control in a single entity. If the owner's private key is compromised, it could lead to catastrophic disruptions or malicious misuse of the contract.

Recommendation:
Use a multi-signature wallet for executing Owner functions. This requires multiple authorized signatures to approve critical actions, reducing the risk of a single point of failure.

LOW-2 | UNRESOLVED

Lack of Two-Step Ownership Transfer

The `TokenEmissionsStrategy` contract does not implement a two-step process for transferring ownership. In its current state, ownership can be transferred in a single step, which can be risky as it could lead to accidental or malicious transfers of ownership without proper verification.

Recommendation:
Implement a two-step process for ownership transfer where the new owner must explicitly accept the ownership. It is advisable to use OpenZeppelin's Ownable2Step.

LOW-3 | UNRESOLVED

Owner can renounce ownership

The `Ownable` contracts includes a function named `renounceOwnership()` which can be used to remove the ownership of the contract.
If this function is called on the `TokenEmissionsStrategy` contract, it will result in the contract becoming disowned. This would subsequently break functions of the token that rely on `onlyOwner` modifier.

Recommendation:
override the function to disable its functionality, ensuring the contract cannot be disowned e.g.

LOW-4 | UNRESOLVED

Missing Emergency Exit for Rewards Distribution

Description:
- The TokenEmissionsStrategy contract allows the owner to perform an `emergencyWithdrawal()`, but there is no way to stop or pause rewards emissions in case of an emergency.

Scenario:
- If there is an issue with the emissions process or a vulnerability is discovered, there is no way to halt the distribution of rewards, potentially leading to loss of funds.

Recommendation:
- Implement a mechanism to pause or stop emissions in case of an emergency. For example, a `pause()` function controlled by the owner that stops rewards emissions temporarily until the issue is resolved.

Lack of Detailed Event Emissions for Critical Functions

Description:

- Critical operations in TokenEmissionsStrategy contract such as notifying the `EmissionManager` and performing reward transfers do not emit detailed events. This makes it harder to track, audit, and monitor contract activities, particularly off-chain. In a situation where malicious activity occurs, it would be challenging to trace key operations without comprehensive logging.

Scenario:

1. A malicious actor abuses the contract's reward transfer function.
2. Without proper event logs, it becomes difficult to investigate the incident and determine the source of the vulnerability.

Recommendation:

- Add detailed events for all key operations, such as:
  - Emissions being notified.
  - Reward transfers.
  - Whitelisting of addresses

Time-based Emission Notification Manipulation

Description:

- The TokenEmissionsStrategy contract restricts emissions by ensuring that the `EmissionManager` can only be notified once per day (24 hours), using the `lastNotified[_reserve]` mapping to track the time of the last notification. However, there is a 10-minute buffer (`lastNotified[_reserve] + 1 days - 10 minutes`) that could allow a malicious actor to game the system by notifying emissions slightly earlier than intended.

Scenario:

1. A malicious actor calls `notifyEmissionManager()` just within the 10-minute buffer before the 24-hour period expires.
2. The attacker can consistently notify emissions slightly earlier each day, creating an emission schedule that does not align with the intended timeframe.

Recommendation:

- Remove the 10-minute buffer or reduce its size to ensure that emissions can only be notified after exactly 24 hours have passed.

  Replace:
  lastNotified[_reserve] + 1 days - 10 minutes <= block.timestamp
- With :

lastNotified[_reserve] + 1 days <= block.timestamp

Floating Pragma

The smart contracts are using a floating pragma version (`^0.8.0`). Contracts should be deployed using the same compiler version and settings as were used during development and testing. Locking the pragma version helps ensure that contracts are not inadvertently deployed with a different compiler version.

Recommendation:
Consider locking the pragma version to a specific, tested version to ensure consistent compilation and behavior of the smart contract.

	./pendle/ATokenPendleLP.sol **./pendle/ATokenPendlePT.sol** **./core/TokenEmissionsStrategy.sol** **./aerodrome/ATokenAerodromeLP.sol** **./mahaxyz/ATokenMahaStaker.sol** **./interfaces/IAerodromeGauge.sol** **./interfaces/IMahaStakingRewards.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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting ZeroLend in verifying the correctness of their contract/s code, our team was responsible for writing integration tests using the Foundry testing framework.

The tests were based on the functionality of the code, as well as a review of the ZeroLend contract/s requirements for details about issuance amounts and how the system handles these.

The code was tested on BASE fork and consisted of interacting with given addresses on BASE as well as test/mock contracts to interact with the protocols. Regarding implications of adding code to live marked, the team is advised to test the codebase further on the forked Mainnet when the Pendle LP market and other protocols launch in order to ensure the Robustness of the code and security. Currently they were not live on the BASE mainnet during the time of testing.

The team has ensured on an average 90% and above test coverage on the current codebase as given below.

Contract: ATokenAerodromeLp & TokenEmissionsStrategy
✓ check deposit
✓ check withdraw
✓ check notifyEmissionManager
✓ check whitelistNotifyEmissionManagerAndPerformTransfer
✓ check emergencyWithdraw
✓ check setEmissionsManager
Contract: ATokenMahaStaker
✓ check mint
✓ check burn
✓ check refreshRewards
✓ check setEmissionsManager
Contract: ATokenPendleLP
✓ check setEmissionsManager
✓ check mint
✓ check burn
Contract: ATokenPendlePT
✓ check mint

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
contracts/aerodrome/ATokenAerodromeLp.sol	100.00%	100.00%	100.00%	100.00%
contracts/core/TokenEmissionsStrategy.sol	100.00%	92.86%	90.00%	100.00%
contracts/mahastaker/ATokenMahaStaker.sol	100.00%	100.00%	85.71%	100.00%
contracts/pendle/ATokenPendleLP.sol	100.00%	66.67%	85.71%	100.00%
contracts/pendle/ATokenPendlePT.sol	100.00%	100.00%	100.00%	100.00%
All files	**100.00%**	**91.9%**	**92.2%**	**100.00%**

We are grateful for the opportunity to work with the ZeroLend 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 ZeroLend team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo