ALMANAK

SMART CONTRACTS REVIEW

zokyo

September 26th 2025 | v. 1.0

Security Audit Score

PASS

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

ZOKYO AUDIT SCORING ALMANAK

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
- 0 Medium issues: 0 points deducted
- 5 Low issues: 5 resolved = 0 points deducted
- 5 Informational issues: 3 acknowledged and 2 resolved = 0 points deducted

Thus, the score is 100

TECHNICAL SUMMARY

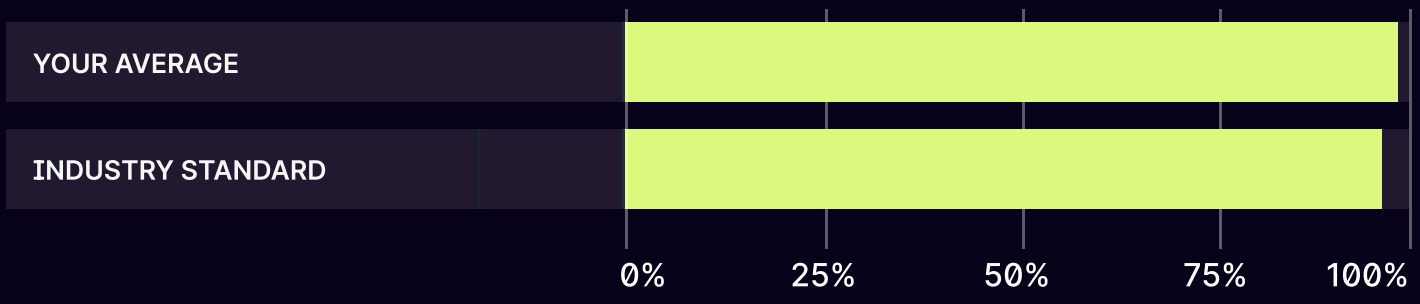
This document outlines the overall security of the Almanak smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Almanak 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



98.67% 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 Almanak 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 5

Executive Summary 7

Structure and Organization of the Document 8

Complete Analysis 9

Code Coverage and Test Results for all files written by Zokyo Security 16

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Almanak repository:
Repo: https://github.com/almanak-co/contracts

Last commit - 13b33175322ff7efce52dd1915f4aef068f49716

Contracts under the scope:

- contracts/AlmanakToken.sol
- contracts/AlmanakTokenL2.sol
- contracts/AlmanakOFTAdapter.sol
- contracts/AlmanakAirdrop.sol
- contracts/VotingEscrowALMANAK.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 Almanak 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.
02	Cross-comparison with other, similar smart contract/s by industry leaders.
03	Fuzz tested the contract/s to detect logic issues via randomized inputs.
04	Thorough manual review of the codebase line by line.

Executive Summary

The protocol consists of a series of smart contracts that together define the lifecycle, utility, and governance of the ALMANAK token. At its core is AlmanakToken.sol, an ERC20-compliant contract that incorporates ERC20Permit functionality, enabling gasless approvals thought signatures. The contract is Ownable, granting administrative privileges to a designated account, and it mints a fixed token supply at initialization.

Extending this base, AlmanakTokenL2.sol implements the LayerZero Omnichain Fungible Token (OFT) standard. This design allows tokens to move across chains using LayerZero messaging, while maintaining compatibility with ERC20Permit. Unlike the main contract, this implementation does not mint an initial supply, instead relying on tokens bridged from other deployments. Complementing it, AlmanakOFTAdapter.sol acts as an adapter that wraps the base ERC20 token into an OFT-compatible form. It manages token flows during bridging by locking them when users transfer out and releasing them when tokens arrive from external networks.
For initial distribution, AlmanakAirdrop.sol provides a gas-efficient system based on Merkle proofs. Eligible participants can claim tokens during a defined period by presenting valid proofs. The contract can be paused in emergency situations.

The governance layer is implemented throught VotingEscrowAlmanak.sol, which follows a vote-escrowed token model. Users may lock their tokens for durations ranging from one week up to two years. Voting power is tied to the lock duration and decreases linearly over time. Each user maintains only a single locking position, and the contract stores checkpoints to allow queries of historical voting power.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Almanak team and the Almanak team is aware of it, but they have chosen to not solve it. 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	Airdrop tokens sent by error to the contract will get locked	Low	Resolved
2	Unclaimed variable can get manipulated	Low	Resolved
3	Airdrop duration can be set really close to `0`	Low	Resolved
4	Missing safety checks	Low	Resolved
5	Using not locked solidity versions can lead to unexpected behaviors	Low	Resolved
6	Owner is in control of the whole token supply	Informational	Acknowledged
7	merkleRoot is immutable	Informational	Acknowledged
8	Unused internal functions	Informational	Resolved
9	VotingEscrow contract allows external contracts to create/modify locks	Informational	Acknowledged
10	Unused variable and internal functions	Informational	Acknowledged

LOW-1 | RESOLVED

Airdrop tokens sent by error to the contract will get locked

Description:
The `recoverWrongToken` within `AlmanakAirdrop.sol` is used to recover any token from the contract except from the airdrop token. However, someone can send any airdrop token after claiming them.

Impact:
Tokens sent to the contract will get locked. Owner will not be able to withdraw them via `revokeUnclaimedTokens` if this function has already been called once.

Recommendation:
Allow the owner to withdraw any type of token from the contract to avoid tokens getting locked.
Add an `amount` parameter to select the amount to withdraw.

LOW-2 | RESOLVED

Unclaimed variable can get manipulated

Description:
The getAirdropStats() function in the AlmanakAirdrop.sol contract is expected to return the unclaimed token amount, among other values. However, to calculate this amount, it relies on token.balanceOf(address(this)).

The same situation applies for the `getUnclaimedAmount()` function.

Impact:
This calculation may not reflect the actual unclaimed amount, because anyone could claim tokens and then send them back to the contract, artificially inflating balanceOf.

Recommendation:
Consider returning the total amount - the already claimed amount instead of relying in balanceOf.

LOW-3 | RESOLVED

Airdrop duration can be set really close to 0.

Description:
The constructor within `AlmanakAirdrop.sol` reverts if claimDuration is set to 0. However, there is not a minimum duration.

Impact:
If the airdrop duration is set to 1, the deployment will not revert, but in practice, it will be almost the same as setting it to 0.

Recommendation:
Define a minimum claim duration and check that the used duration is greater or equal to the minimum.

LOW-4 | RESOLVED

Missing safety checks

Description:
The constructor of VotingEscrowALMANAK.sol does not validate the _token and _owner parameters. If either of these values is set to the zero address (address(0)), the contract may be deployed in a broken state, making it impossible to interact safely with the escrow or properly enforce ownership.

This is present across the rest of contracts as well.

Impact:
If _token is address(0), all ERC20 interactions will fail, effectively bricking the contract.

If _owner is address(0), ownership functionality from Ownable will be unusable, and critical administrative actions (e.g., upgrades, parameter tuning) may not be possible.

Recommendation:
Add explicit checks in the constructor to ensure _token and _owner are not the zero address.

LOW-5 | RESOLVED

Using not locked solidity versions can lead to unexpected behaviors

Description:
Contracts in the codebase are using unlocked Solidity pragma statements such as pragma solidity ^0.8.28;. This allows the compiler to use any version greater than or equal to 0.8.28. While this provides flexibility, it can introduce risks if newer compiler versions contain undiscovered bugs, behavior changes, or incompatibilities.

Impact:
Future Solidity compiler versions could introduce breaking changes, security vulnerabilities, or unintended behavior.
Builds may become non-deterministic if developers use different compiler versions.
Auditability and reproducibility of the contracts are reduce

Recommendation:
Lock the Solidity compiler to a specific, stable version to ensure deterministic builds, avoid unexpected behavior, and improve long-term maintainability and auditability of the contracts. Check every contract which is not using a locked solidity version.

INFORMATIONAL-1 | ACKNOWLEDGED

Owner is in control of the whole token supply

Description:
The address set as `owner` while deploying AlmanakToken is in full control of the whole total supply as he is receiving the whole total minted supply.

INFORMATIONAL-2 | ACKNOWLEDGED

merkleRoot is immutable

Description:
The merkleRoot used within AlmanakAirdrop is immutable. This means that once the contract is deployed, the root cannot be modified, and any mistake in its generation (such as incorrect addresses, amounts, or formatting) would render the airdrop unusable or unfair.

One possible improvement is to add a setter function that allows the contract owner (or another authorized role) to modify the merkleRoot in case an error is discovered and the root needs to be updated.

INFORMATIONAL-3 | RESOLVED

Unused internal functions

Description:
The internal functions _findBlockEpoch and _findUserBlockEpoch are defined in VotingEscrowALMANAK.sol but are not referenced anywhere within the contract.

Impact:
While this does not introduce a functional vulnerability, unused functions can increase the complexity of the codebase, make audits less efficient, and potentially confuse future maintainers regarding their intended purpose.

Recommendation:
If these functions are not required, consider removing them to reduce code complexity. If they are intended for future functionality, add comments clarifying their purpose and expected usage.

INFORMATIONAL-4 | ACKNOWLEDGED

VotingEscrow contract allows external contracts to create/modify locks

Description:
The original Curve's VotingEscrow contract does not allow contracts to create/modify locks (the operation is forbidden in create_lock, increase_amount and increase_unlock_time). However, Almanak's contract allows this feature.

Recommendation:
If allowing contracts to create/modify locks is not desired to be allowed, as in Curve's contract, implement a check to forbid these type of operations.

INFORMATIONAL-5 | RESOLVED

Unused variable and internal functions

Description:
The MAX_WITHDRAWAL_PENALTY variable has been declared in the VotingEscrowALMANAK contract but never used. Also internal function _findUserBlockEpoch and _findBlockEpoch are never invoked anywhere in the contract

Recommendation:
Consider removing these if they are not meant to be used in future.

	contracts/AlmanakToken.sol contracts/AlmanakTokenL2.sol contracts/AlmanakOFTAdapter.sol contracts/AlmanakAirdrop.sol contracts/VotingEscrowALMANAK.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 are written by Zokyo Secured team

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

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

Ran 25 tests for .audit/test/AlmanakAirdropTest.t.sol:AlmanakAirdropFuzzTest
[PASS] testFuzz_Claim(uint256,uint256,uint256,uint256,address) (runs: 10014, μ: 1868381, ~: 1868888)
[PASS] testFuzz_ClaimReverts(uint256,uint256,uint256,uint256,address) (runs: 10014, μ: 1792798, ~: 1793369)
[PASS] testFuzz_ClaimTwice(uint256,uint256,uint256,uint256,address) (runs: 10014, μ: 1871942, ~: 1872449)
[PASS] testFuzz_ConstructorReverts(address,bytes32,uint256,uint256,uint256) (runs: 10014, μ: 1192347, ~: 1719586)
[PASS] testFuzz_ConstructorValid(uint256,uint256,uint256) (runs: 10014, μ: 1754836, ~: 1755404)
[PASS] testFuzz_OFTInitialize(string,string,address) (runs: 10014, μ: 9626469, ~: 9614056)
[PASS] testFuzz_RevokeRepeatRevert(address,uint256) (runs: 10010, μ: 1816852, ~: 1817521)
[PASS] testFuzz_RevokeReverts(address,uint256,address) (runs: 10014, μ: 1784776, ~: 1785447)
[PASS] testFuzz_RevokeSuccess(address,uint256) (runs: 10010, μ: 1816743, ~: 1816115)
[PASS] testFuzz_canClaim_false_afterDeadline(uint64,address,uint256) (runs: 10014, μ: 1782873, ~: 1782234)
[PASS] testFuzz_canClaim_false_beforeStart(uint64,uint256) (runs: 10014, μ: 1782950, ~: 1782299)
[PASS] testFuzz_canClaim_false_paused(uint256,address,uint256) (runs: 10014, μ: 1813461, ~: 1813427)
[PASS] testFuzz_canClaim_false_zeroAmount(uint64) (runs: 10014, μ: 1780578, ~: 1780578)
[PASS] testFuzz_canClaim_true_validConditions(uint64) (runs: 10014, μ: 1783725, ~: 1783725)
[PASS] testFuzz_getClaimableAmount_returnsAmount_whenValid(uint256,uint256) (runs: 10014, μ: 1789836, ~: 1789808)
[PASS] testFuzz_getClaimableAmount_zero_afterDeadline(uint256,address,uint256) (runs: 10014, μ: 1787421, ~: 1786818)
[PASS] testFuzz_getClaimableAmount_zero_beforeStart(uint256,address,uint256) (runs: 10014, μ: 1786457, ~: 1786414)
[PASS] testFuzz_getClaimableAmount_zero_hasClaimed(uint256,uint256) (runs: 10014, μ: 1863306, ~: 1863278)

[PASS] testFuzz_getClaimableAmount_zero_invalidProof_amountMismatch(uint256,uint256) (runs: 10014, μ: 1790071, ~: 1790039)
[PASS] testFuzz_getClaimableAmount_zero_paused(uint256,address,uint256) (runs: 10014, μ: 1798214, ~: 1798180)
[PASS] testFuzz_recoverWrongToken(uint8,address) (runs: 10014, μ: 2705827, ~: 2703301)
[PASS] testFuzz_timeUntilClaimStart(uint256) (runs: 10014, μ: 1779998, ~: 1780466)
[PASS] testFuzz_timeUntilDeadline(uint256) (runs: 10014, μ: 1780203, ~: 1780669)
[PASS] test_getAirdropStats() (gas: 1784520)
[PASS] test_getClaimWindow() (gas: 1772726)
Suite result: ok. 25 passed; 0 failed; 0 skipped; finished in 43.04s (95.33s CPU time)

Ran 20 tests for .audit/test/VotingEscrowAlmanak_FuzzTest.t.sol:VotingEscrowALMANAK_FuzzTest
[PASS] testFuzz_CreateLock(uint256,uint256,address) (runs: 10014, μ: 330305, ~: 355842)
[PASS] testFuzz_CreateLock(uint256,uint256,uint256,uint256,address,address) (runs: 10014, μ: 1550603, ~: 985370)
[PASS] testFuzz_CreateLockRandMultiple(uint256[5],uint256[5],uint256[5],bytes32) (runs: 10014, μ: 2414617, ~: 2366412)
[PASS] testFuzz_CreateLockReverts(uint256,uint256,address) (runs: 10014, μ: 84262, ~: 86223)
[PASS] testFuzz_DepositWithdrawRandMultiple(uint256[5],uint256[5],uint256[5],bytes32) (runs: 10014, μ: 2054170, ~: 2119833)
[PASS] test_createLock_Revert_LockAlreadyExists() (gas: 339436)
[PASS] test_createLock_Revert_TooLong() (gas: 22776)
[PASS] test_createLock_Revert_UnlockNotInFuture() (gas: 22395)
[PASS] test_createLock_Revert_ZeroAmount() (gas: 22477)
[PASS] test_decay_Monotonic() (gas: 402075)
[PASS] test_increaseAmount_Revert_Expired() (gas: 339289)
[PASS] test_increaseAmount_Revert_NoLock() (gas: 21252)
[PASS] test_increaseAmount_Revert_Zero() (gas: 338577)
[PASS] test_increaseUnlockTime_Revert_Expired() (gas: 340029)
[PASS] test_increaseUnlockTime_Revert_NoLock() (gas: 22411)
[PASS] test_increaseUnlockTime_Revert_NotGreater() (gas: 339010)
[PASS] test_increaseUnlockTime_Revert_TooLong() (gas: 339981)
[PASS] test_multiUser_OverlappingLocks_WeeklyRolls() (gas: 1137683)
[PASS] test_withdraw_Revert_NoLock() (gas: 20657)
[PASS] test_withdraw_Succeeds() (gas: 654311)
Suite result: ok. 20 passed; 0 failed; 0 skipped; finished in 43.04s (79.90s CPU time)

Ran 3 tests for .audit/test/VotingEscrowALMANAKTestBSearch.t.sol:VotingEscrowALMANAKTestBSearch
[PASS] testFuzz_FindBlockEpoch(uint256,uint256,uint256) (runs: 10014, μ: 7294929, ~: 7213839)
[PASS] testFuzz_FindUserBlockEpoch(address,uint256,uint256,uint256) (runs: 10014, μ: 7419496, ~: 7325224)
[PASS] testFuzz_SupplyAt_AllBranches(uint256,uint32,int16,int16,uint32) (runs: 10014, μ: 71984, ~: 75518)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 45.63s (89.39s CPU time)

Ran 3 test suites in 45.64s (131.71s CPU time): 48 tests passed, 0 failed, 0 skipped (48 total tests)

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	%UNCOVERED LINES
AlmanakAirdrop.sol	100.00% (86/86)	100.00% (20/20)	100.00% (13/13)	100.00% (80/80)	
AlmanakToken.sol	100.00% (1/1)	(0/0)	100.00% (1/1)	100.00% (2/2)	
VotingEscrowALMANAK.sol	98.13% (210/214)	92.16% (47/51)	100.00% (16/16)	97.91% (187/191)	290,294,317,320
All Files	**98.67%**	**94.37%**	**100%**	**98.5%**	

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

zokyo