KINTO × PESSIMISTIC

SECURITY ANALYSIS

by Pessimistic

This report is public

February 5, 2025

ABSTRACT

In this report, we consider the security of smart contract of Kinto SealedBidTokenSale project. Our task is to find and describe security issues in the smart contracts of the platform.

DISCLAIMER

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

SUMMARY

In this report, we considered the security of Kinto SealedBidTokenSale smart contract. We described the audit process in the section below.

The audit showed one issue of medium severity: Owner's role. Also, two low-severity issues were found.

The overall code quality is good.

GENERAL RECOMMENDATIONS

We recommend fixing the mentioned issues.

PROJECT OVERVIEW

Project description

For the audit, we were provided with Kinto SealedBidTokenSale project on a public GitHub repository, commit 157028bb98b3a31dcd6c5922faefcfcf30495d26.

The scope of the audit included the **src/apps/SealedBidTokenSale.sol** file.

The documentation for the project included the following link.

All 42 tests pass successfully. The code coverage is 100%.

The total LOC of audited sources is 99.

AUDIT PROCESS

We started and finished the audit on February 4, 2025.

We inspected the materials provided for the audit. We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Standard Solidity issues;
- Compliance with the documentation;
- Using Merkle tree logic in the tokens sale;
- The impact of the owner's role.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules.

We checked the tests and the code coverage in CI on the GitHub.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the audit, the developers acknowledged all the issues and decided to not implement the fixes.

MANUAL ANALYSIS

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Owner's role

In the current implementation, the system depends heavily on the owner's role. The owner can:

- Modify the Merkle root. This may result in the inability to retrieve sale tokens;
- Complete the sale at any time. This means that if they never end the sale, the users deposits will be locked in the contract;
- Withdraw user deposits via the `withdrawProceeds` function to the treasury. If the contract balance of `USDC` is insufficient, users cannot claim `USDC` via the `claimTokens` function.

Also, if the contract balance of sale tokens is insufficient, users will not be able to get their deposits back. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if owner's private keys become compromised.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Error naming

The `CapNotReached` error name does not fully match the code logic at lines 196, 249, and 265 since the revert can occur if the sale has not ended even when the minimum cap is reached.

L02. Gas consumption

Consider fixing the following points to reduce gas consumption:

- The `nonReentrant` modifier can be removed from all functions where it is used;
- The `MerkleProof.verify` function can be replaced with the verifyCalldata function to avoid copying the `proof` parameter from `calldata` to `memory`.

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Daria Korepanova, Senior Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

February 5, 2025