KINTO × PESSIMISTIC

SECURITY ANALYSIS

by Pessimistic

This report is public

September 18, 2024

ABSTRACT

In this report, we consider the security of smart contracts of Kinto Governance 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 Governance smart contracts. We described the audit process in the section below.

The audit showed several issues of medium severity: Discrepancy with the documentation, Misleading check function, Possible delegation. Also, several low-severity issues were found. All tests passed.

After the initial audit, the developers provided us with a new version of the code. In this update, they fixed Misleading check function, Possible delegation medium-severity, and most of the low-severity issues. Moreover, they partially fixed the Discrepancy with the documentation issue.

GENERAL RECOMMENDATIONS

We recommend fixing the remaining issue.

PROJECT OVERVIEW

Project description

For the audit, we were provided with Kinto Governance project on a public GitHub repository, commit 903b9790c820e9a3e7b18e8e7aaf2544b3ac55ea.

The scope of the audit included:

- **NioElection**;
- **NioGovernor**;
- **NioGuardians**;
- **BridgedKinto**;
- **BridgedToken**.

The documentation for the project included the link.

All 611 tests of the whole project pass successfully. The code coverage of the entire project is 90%.

The total LOC of audited sources is 579.

Codebase update

After the audit, the developers provided us with a new commit: 523c0337965e51d9c5dafaad57e055eec72b2e3a. In that update, they fixed M02, M03, medium-severity and L02, L03, L04, L05 low-severity issues. Moreover, they partially fixed M01.

All 617 tests pass successfully. The code coverage is 90.51%

AUDIT PROCESS

We started the audit on September 2 and finished on September 6, 2024.

We inspected the materials provided for the audit. And during the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

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:

- Whether the phase order can not be disturbed;
- Whether the code corresponds to the documentation;
- The standard Solidity issues.

We checked the following CI results: the number of passed tests, code coverage and the static analyzer Slither issues. We also scanned the project with the Semgrep tool.

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

After the initial audit, we discussed the results with the developers. On September 12, 2024, the developers provided us with an updated version of the code. In this update, they fixed most of the issues (see here for details) from our report.

We reviewed the updated codebase, ran the aforementioned tools, and updated the report accordingly.

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. Discrepancy with the documentation

The **NioElection** contract has the following code discrepancies with the Nios Election Process:

- **(fixed)** The part of the third point is not implemented in the code:

  > "If fewer candidates that seats are supported by pledged votes representing at least 0.5% of all Votable Tokens, the current Nio members whose seats are up for election may become candidates (as randomly selected out of their Cohort) until there are enough candidates".

  And if there are not enough nominees, the `electNios` function reverts at line 253 with `InsufficientEligibleCandidates` error. It may lead to an inability to complete the election process since the `getCurrentPhase` function will always return the `AwaitingElection` phase.

  > **After the fix, the election will not be stuck. However, note that random selection of the remaining Nio members does not occur in the contract.**

- It says in the second point:

  > "To the extent that there are more than six contenders, each eligible contender must be supported by pledged votes representing at least 0.5% of all Votable Tokens".

  But in the `voteForCandidate` function, each candidate has to get a minimum percentage of the vote at line 194, even if it is less than 6 candidates.

  > `Comment from the developers:`
  > **We will still require all candidates to receive at least 0.5% of the vote.**

M02. Misleading check function (fixed)

According to the first point of the Nios Election Process description:

> "Any DAO member may declare their candidacy as a Nio, provided that a current Nio candidate is not a candidate for a seat in the other cohort".

However, the `isElectedNio` function checks Nios in the current active election when it is called in the `submitCandidate` function. So, it only checks whether a candidate has already been submitted or not instead of whether it is a Nio in the other Cohort.

> **The issue has been fixed and is not present in the latest version of the code.**

M03. Possible delegation (fixed)

The **NioGuardians** token can not be delegated as the `delegate` function at lines 91-93 always reverts. However, the **Votes** contract also has the delegateBySig function, which does not revert in the **NioGuardians** contract.

The **NioGuardians** contract is inherited from **ERC721Votes** contract, which is inherited from the **Votes** contract.

> **The issue has been fixed and is not present in the latest version of the code.**

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. Gas Consumption

The `if` part from lines 62-67 can be called before the `super._update(from, to, amount)` call to decrease gas consumption in the `_update` function of the **BridgedToken** contract.

L02. Incorrect getter's returned value (fixed)

If a candidate is double submitted, the same address will be added to the `candidateList` array of the current election in the `submitCandidate` function of the **NioElection** contract. It leads to incorrect returned value of the `NioElection.getCandidates` function.

> The issue has been fixed and is not present in the latest version of the code.

L03. Typo (fixed)

The `weight` parameter of the `CandidateVoteCast` event could be named as `votes` to match the code's meaning in the **NioElection** contract.

> The issue has been fixed and is not present in the latest version of the code.

L04. Unused constant (fixed)

The `NOMINEE_VOTING_DURATION` constant at line 65 in the **NioElection** contract is not used.

> The issue has been fixed and is not present in the latest version of the code.

L05. Unused import (fixed)

The `"forge-std/console2.sol"` import at line 13 in the **NioElection** contract is not used.

> The issue has been fixed and is not present in the latest version of the code.

Notes

N01. Out of gas

The `NioElection.sortNomineesByVotes` function's gas consumption depends on the number of nominees. If the length of the `nomineeList` is high, the `sortNomineesByVotes` function can be out of gas theoretically. This leads to an inability to complete the election.

However, it is not clear whether it would be economically viable to launch this attack since a malicious one has to:

- Pay for the gas to submit the candidates;
- Collect 0.5% of the votes.

This analysis was performed by Pessimistic:

Daria Korepanova, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

September 18, 2024