azuro × PESSIMISTIC

SECURITY ANALYSIS

by Pessimistic

This report is public

August 15, 2024

ABSTRACT

In this report, we consider the security of smart contracts of Azuro RewardPoolV2 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 Azuro RewardPoolV2 smart contracts. We described the audit process in the section below.

The audit showed several issues of medium severity: No documentation, Project's roles, Asynchronous balances update. Also, two low-severity issues and several notes were found. All the tests passed.

After the initial audit, the codebase was updated. The developers fixed the Asynchronous balances update issue of medium severity and all low severity issues. The number of tests increased. All tests successfully passed.

We cannot guarantee the security of the **RewardPool** contract, as the project scope was limited. We checked the **RewardPoolV2** contract and only the migration functionality of the **RewardPool** contract.

GENERAL RECOMMENDATIONS

We recommend fixing the mentioned issues or providing comments. We also recommend implementing CI to run tests, calculate code coverage, and analyze code with linters and security tools.

PROJECT OVERVIEW

Project description

For the audit, we were provided with Azuro RewardPoolV2 project on a public GitHub repository, commit 97ae4fe256c257f156887acc81697ba7dcbe1244.

The scope of the audit included **RewardPoolV2.sol**, **RewardPool.sol** (only migration functionality).

The documentation for the project included only NatSpecs.

All 41 tests pass successfully. The code coverage is 100%.

The total LOC of audited sources is 326.

Codebase update #1

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 94482b7e7f2ae8a10fcb04ada3dedc07028aa435.

The developers fixed the medium severity issue, all low severity issues and one note.

The number of tests increased. All 42 tests passed. The code coverage is 100%.

AUDIT PROCESS

We started the audit on August 7 and finished on August 8, 2024.

We inspected the materials provided for the audit. Then, we contacted the developers in the chat for an introduction to the project.

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 migration process is safe;
- Whether the balance accounting is correct;
- Whether the influence of the owner is limited;
- Solidity standard checks.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts. We also sent the results to the developers in the text file.

We ran tests and calculated the code coverage.

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

We made the recheck from August 14 to August 15, 2024. We checked the fixes of the issues, re-ran tests and the code coverage.

Finally, we updated the report.

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. No documentation

The project has the detailed NatSpecs. However, it has no documentation. The documentation is a critical part that helps to improve security and reduce risks. It should explicitly explain the purpose and behavior of the contracts, their interactions, and key design choices. It is also essential for any further integrations.

M02. Project's roles

In the current implementation, the system depends on the owner role. 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.

The owner can:

- Change the address of the **RewardPoolV2** contract (which is used in the migration process) in the `RewardPool.changeRewardPoolV2` function. It may lead to sending unstaked tokens to the malicious contract;
- Change the delay for withdrawals in the `RewardPoolV2.changeWithdrawalDelay` function, which gives the possibility for censorship;
- Change the implementation of the **RewardPoolV2** and **RewardPool** contracts.

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

M03. Asynchronous balances update (fixed)

During the withdrawal request, the **RewardPoolV2** balances of the underlying token and wrapped tokens update at different times. The `totalSupply` updates during the withdrawal request, and the underlying token balance updates after a delay during the withdrawal.

As a result, the owner can mint extra wrapped tokens through the `recover` function. After withdrawing, the underlying token balance is less than the `totalSupply`, which may result in not all users being able to withdraw their part of the tokens.

> **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. Redundant code (fixed)

The **RewardPoolV2** contract has the following redundant code, which duplicates the code from the **ERC20WrapperUpgradeable** contract:

- The `ERC20WrapperStorageLocation` constant;
- The `__getERC20WrapperStorage` function.

This constant and function are used to get the address of the underlying token via `__getERC20WrapperStorage()._underlying` call in the **RewardPoolV2** contract. However, the underlying token address can be returned from the `public` `underlying` function of the **ERC20WrapperUpgradeable** contract.

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

L02. Multiple reads from storage (fixed)

The storage variable `rewardPoolV2` is read multiple times in the `migrateToV2` function of the **RewardPool** contract.

Consider reading it once to local variable to reduce gas consumption.

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

Notes

N01. Disable initializers (fixed)

Consider calling the `_disableInitializers` function in the **RewardPoolV2** constructor to avoid direct initialization of the implementation.

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

N02. Contracts consistency

Consider using ERC7201 in the **RewardPoolV2** for consistency with the **ERC20WrapperUpgradeable** contract.

This analysis was performed by Pessimistic:

Daria Korepanova, Senior Security Engineer
Evgeny Bokarev, Junior Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

August 15, 2024