plasma finance

Quadrat Booster Security Analysis

by Pessimistic

This report is public

April 11, 2023

Abstract

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

The initial audit showed several issues of medium severity: Deposit prevention, Gas costs, Owner roles. Also, several low-severity issues were found.

The overall code quality is mediocre. The code structure is complex, which reduces the readability of the code.

After the initial audit, the codebase was updated. Most of the issues were fixed. However, a new critical issue was found.

After the first recheck, the codebase was updated. The critical issue C01 has been fixed.

General recommendations

We recommend using only one proxy pattern. Moreover, we recommend optimizing gas usage for the priority queue in addition to already implemented improvements (see M02).

Project overview

Project description

For the audit, we were provided with Quadrat Booster project on a private GitHub repository, commit c560b67c9f4b97526accb67b7f0b17183c405e58.

The scope of the audit includes everything.

The documentation for the project includes **README.md**.

All 26 tests pass successfully. The code coverage is 99.27%.

The total LOC of audited sources is 841.

Codebase update #1

After the audit, we were provided with commit 5c33c0a901b21c18638ca22feb08b4486850f9a0. In the update, the developers fixed most of the issues and implemented new tests. However, the developers introduced one new critical issue: C01.

All 28 tests pass successfully. The code coverage is 99.68%.

Codebase update #2

After the first recheck, we were provided with the code in the following zip archive: **quadratbooster_0e348fbc01b42d8de4cfb852b4fd5325574bfd6c_contracts.zip**, sha1sum **e915f7ae6f84ecfea2e738f9b03f399511d4f8c5**.

In the update, the developers fixed the critical issue (C01), added a new function, and implemented new tests.

All 29 tests pass successfully. The code coverage is 100%.

Audit process

We started the audit on March 7, 2023, and finished on March 15, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research and specified those parts of the code and logic that require additional attention during an audit:

- The logic of bonus reward calculation;
- Gas costs of cumulative reward logic;
- Possible attacks on the proxy logic.

We manually analyzed all the contracts within the scope of the audit and checked their logic.

We scanned the project with the static analyzer Slither and our private plugin with an extended set of rules and then manually verified their output.

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 an automated tool.

On March 20, the developers provided a new version of the code. We reviewed the fixes for the issues and inserted comments from the developers. We contacted the developers regarding the newly found critical issue.

Finally, we updated the report.

On April 6, the developers provided a new version of the code. We reviewed the fix of the critical issue and the newly added function.

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.

C01. Reward calculation for future blocks (fixed)

In **QuadratBoosterStorage** contract, the `updateCumulativeRewardByBlock` function allows anyone to update the cumulative reward until the given arbitrary block number (which is less than `toBlock`). With this, it is possible to update the cumulative reward for future blocks and break the reward calculation logic.

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

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. Deposit prevention (fixed)

In **QuadratBoosterStorage** contract, the `transferDeposit` function allows the user to transfer their deposit to another address. As a result, this function overwrites the recipient `DepositInfo` structure. That means this function allows sending a small number of locked tokens to an address in order to prevent them from staking funds.

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

M02. Gas costs (fixed)

The business logic of the boosting allows the user to receive bonus rewards during a specific period. In the current project, this logic is implemented with the use of a priority queue. Priority queue might contain various structures underneath, gaining different time and storage complexities. For the project, it is required to have better `dequeue` time complexity compared to the complexity of `enqueue` operation. This is due to the fact that the insertion occurs only once per deposit transaction, while removal occurs multiple times in the `_calculateCumulativeRewardPerShare` function. The project implementation uses linked lists, which is optimal for that task. However, a simple transaction with 500 removals from the queue and 500 slots written to the storage consumes approximately 18M gas. We recommend either significantly optimizing the code or updating all booster contract rewards on a regular basis.

The developers implemented a function for updating cumulative rewards up to a given block. Even though this fixes the issue, there might still be problems in the case when there are lots of deposits in one.

M03. Owner roles (addressed)

The system relies heavily on the following roles in the project:

- In **QuadratBoosterFactory** contract, the contract manager can:
    - upgrade the logic of booster contracts;
    - make some boosters immutable;
    - set the booster deployment fee.
- In **QuadratBooster** contract, the contract manager can:
    - transfer unlocked rewards to treasure;
    - update `fromBlock`, `toBlock`, `_blockBonus` parameters;
    - update `minimalWithdrawBlocks` parameter. As a result, the owner is able to lock user funds for an indefinite period by frontrunning the `deposit` transaction.

There are scenarios that can lead to undesirable consequences for the project and its users, e.g., if private keys for the **QuadratBoosterFactory** owner become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

In addition to this, we recommend reviewing the powers of **QuadratBooster** owner role.

The developers decided to use a multisig wallet for the factory manager address.

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)

In **QuadratBoosterStorage.sol**, calculations at lines 369-371 are redundant. According to line 353, `_blockNumber` is already not greater than `toBlock`.

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

L02. Missing check (fixed)

We recommend adding the `nonReentrant` modifier to `transferDeposit` function in order to mitigate possible reentrancy attacks using ERC777 token standard in the `transferDeposit` function of **QuadratBoosterStorage** contract.

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

L03. Missing check (fixed)

QuadratBoosterStorage contract heavily relies on the contract balance. In the case when `depositToken` equals to `rewardToken`, the contract logic will be broken. Hence, we recommend checking that the values are not the same.

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

L04. Missing event parameter names (fixed)

`BoostersUpgraded` and `BoostersUpgradedAndCalled` events do not have a name for the last `address` type parameter, which is the `implementation` address. We recommend not only adding a name for that parameter for the visual clarity, but also marking it as indexed.

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

L05. Missing reward update (fixed)

QuadratBoosterStorage contract owner is able to update `from` and `to` blocks for the staking. However, when these blocks are changed, `rewardPerBlock` variable is not updated. This is crucial, especially for the `setFromBlock` function, since it allows to decrease `rewardPerBlock` value, and, as a result, user tokens might be stuck on the contract if the allocated rewards are greater than the contract balance.

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

L06. Proxy patterns (commented)

The project uses two proxy patterns for the same logic. We recommend unifying code logic, as managing two proxy types might be confusing. Moreover, the `immutable` functionality for the implementation contracts works only for transparent proxies in the project.

According to the developers, they decided to use only the beacon proxy pattern in future versions of the code.

L07. Initialization of implementation contracts (fixed)

The common pattern for upgradeable contracts is to move all the contract initialization into `initialize` function. This allows to initialize the proxy storage. However, there is an attack vector that exploits `initialize` function by making a call directly to the logic contract. The current implementation is safe from this type of attack. Nevertheless, we recommend preventing logic contract initialization in order to make future versions of the code more secure.

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

Notes

N01. Supported tokens (addressed)

The project does not work with several types of tokens. It does not work with the rebasing tokens and with tokens that charge a transfer commission. In order to support tokens with the commission, the contract might check balances before and after transfers.

The developers added a check of the balances after transfers which will disable the usage of tokens with commissions.

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Security Engineer
Yhtyyar Sahatov, Junior Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, Founder

April 11, 2023