Babylon Security Analysis

by Pessimistic

This report is public

April 18, 2022

Abstract

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

Disclaimer

Summary

In this report, we considered the security of Babylon smart contracts. We performed our audit according to the procedure described below.

The initial audit showed an issue of medium severity: Access control. Also, several low-severity issues were found.

After the initial audit, the code base was updated. In this update, all the issues were fixed.

General recommendations

We recommend improving code coverage with tests.

Project overview

Project description

For the audit, we were provided with Babylon project on a private GitHub repository, commit 3f2f1a0d417457981e9955f7969b16efd1f38be7.

The scope of the audit included only **Heart.sol** and **RewardsDistributor.sol** files and their dependencies.

The documentation for the project includes the following links:

- https://www.babylon.finance/heart
- https://docs.babylon.finance/babl/heart
- https://medium.com/babylon-finance/the-heart-of-babylon-defi-3-0-6d16e35817f2
- https://docs.babylon.finance/babl/mining#rewards-per-participant

All 25 tests pass. The overall code coverage for the scope of the audit is 62%.

The total LOC of audited sources is 1449 without dependencies.

Codebase update

After the initial audit, the code base was updated. For the recheck, we were provided with commit 2625a08087dfc55b6a234156ef7bfccd393e4deb In this update, all of the issues were fixed. A few new functions were added, no new issues were found.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
    - We scan the project's codebase with the automated tool Slither.
    - We manually verify (reject or confirm) all the issues found by the tool.

- Manual audit
    - We manually analyze the codebase for security vulnerabilities.
    - We assess the overall project structure and quality.

- Report
    - We reflect all the gathered information in 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 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. Bug in access control (fixed)

The `sellWantedAssetToHeart` of **Heart** contract function allows anyone selling `wantedAssets` tokens to the **Heart** contract. As a result, a malicious user can force the contract to sell its entire stock of `assetForPurchases` tokens and perform a sandwich attack on that sell. Consider adding a check that allows calling `sellWantedAssetToHeart` function only from strategies.

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. Code quality (fixed)

In **RewardsDistributor** contract, the `cp > 0` part of condition at line 1180 is redundant since in case of `cp > 0` the first condition will evaluate to `true`.

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

L02. Code quality (fixed)

Consider declaring the following functions as `external` instead of `public` to improve code readability and optimize gas consumption:

1. `getRewards` function of **RewardsDistributor** contract.
2. `updateAssetToPurchase` function of **Heart** contract.

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

L03. Code quality (fixed)

In **RewardsDistributor** contract, the `_getSafeUserSharePerStrategy` function accepts an array with 14 elements as an argument. However, the function uses only three elemnts. Consider passing only those arguments that are expected to be used to improve code readability and optimize gas consumption.

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

L04. Code quality (fixed)

Consider using `!x` condition instead of `x == false` to improve code readability in **RewardsDistributor** contract at lines 829, 1439, 1446, 1449, 1503, and 1536.

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

L05. Code quality (fixed)

Multiple return values should have multiple `@return` blocks in NatSpec comments for `getPriorBalance` function of **RewardsDistributor** contract.

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

L06. Fuse integration issue (fixed)

The Compound documentation states that the `mint` function returns `0 on success, otherwise an Error code`. However, `_lendFusePool` function of **Heart** contract does not verify the returned values of `cToken.mint` calls on line 597. This might result in incorrect protocol statistics in case if the transaction rejection.

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

Notes

N01. Powered roles

The system depends heavily on the following roles with special powers:

1. The `Emergency` role can change the heart garden address and update the weights maximizing the share of distributed value to new address. The role can change these parameters at any moment and can front-run other transactions with these changes.

2. The `Keeper` role transfers the results from offline voting to the contract without any additional verification. Thus, the role can provide any results.

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

> *Comment from the developers: We confirm that Emergency role is actually a multisig (Gnosis Safe).*

This analysis was performed by Pessimistic:

Vladimir Tarasov, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Nikita Kirillov, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

April 18, 2022