Babylon Finance Security Analysis

by Pessimistic

This report is public

5 November, 2021

Abstract

In this report, we consider the security of smarts contracts of Babylon Finance 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. One 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, security audit is not an investment advice.

Summary

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

The audit showed multiple issues of medium severity, which are DoS with Out of Gas, three Bugs, ERC20 standard violation, and Overpowered role issue. It also showed many low-severity issues.

The project has sufficient documentation. Some functionality is not covered with tests.

After the initial audit, the code base was updated.
DoS with Out of Gas and ERC20 standard violation issues of medium severity and some of the Bugs were fixed, as well as most of low-severity issues.

General recommendations

We recommend addressing the issues that are still present in the code. Also consider introducing CI to run tests and calculate code coverage regularly.

Project overview

Project description

For the audit, we were provided with Babylon Finance project on a private GitHub repository, commit 628f72106cbb624ec448eb0f4e68decdd4143e8e.

The project has `README.md` file that describes the system well enough.

All 6 tests pass, the code coverage is 27%. However, `ProphetsArrival` contract is not tested at all.

The total LOC of audited sources is 265.

Code base update

After the initial audit, the code base was updated. For the recheck, we were provided with commit 98972999245fee35de390fefa9f748cac2ab7c66.

This update fixes most issues and adds a lot of tests. All 47 tests pass, resulting code coverage is 100%.

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 project's code base with automated tools: Slither and SmartCheck.
    - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
    - We manually analyze code base for security vulnerabilities.
    - We assess 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 often lead to the loss of funds or other catastrophic failures. The contracts should not be deployed before these issues are fixed. We highly recommend fixing them.

The audit showed no critical issues

Medium severity issues

Medium issues can influence project operation in 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.

DoS with Out of Gas error (fixed)

The `setGreatProphetsAttributes` function of `Prophets` contract performs 5k writes to storage, which on its own costs 100M gas. It exceeds block size limit on mist chains. Consider adding data in smaller batches.

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

Bugs

- Use built-in multipliers like `days` or `hours` to describe time intervals. Currently 8 hours are replaced by 8 seconds at line 27 of `ProphetsArrival` contract.

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

- The `mintProphet` function of `Prophets` contract assigns IDs starting with `1`, and can mint up to `NORMAL_PROPHETS` NFTs (i.e. 8000). However, other parts of the contract treat the token with id 8000 as a "Great Prophet".

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

- `claimLoot` function doesn't allow to claim loot for the last "Great Prophet" due to off-by-one error at line 120 of the `Prophets` contract.

ERC20 standard violation (fixed)

ERC-20 standard states:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, returned value of `transfer` call is not checked at line 122 of `Prophets` contract.

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

Low severity issues

Low severity issues don't directly affect project's operations. However, they might lead to various problems in the future versions of the code. We recommend taking them into account.

Code quality

- We recommend returning a struct instead of multiple return values, e.g., in `Prophets.getProphetAttributes` view function.

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

- The `Prophets._totalSupply` internal function is never called and should be removed to improve readability.

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

- `mintProphet` and `mintGreatProphet` functions of `Prophets` contract are payable for no apparent reason.

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

- Mark functions as `external` whenever possible. It improves readability and saves a bit of gas.

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

- Multiple addresses are hardcoded in `Prophets` and `ProphetsArrival` contracts. Consider using immutable variables and setting them via constructor instead.

  This issue has been partially fixed and only `BABYLON_TREASURY` address is hardcoded in the latest version of the code.

- We highly recommend following CEI (checks-effects-interactions) pattern since it prevents some types of re-entrancy attacks. The `mintProphet` function of `ProphetsArrival` contract violates this pattern. It doesn't pose a direct threat though, since the function is `nonReentrant` and doesn't call third-party contracts.

  This issue has been fixed and is not present in the latest version of the code. However, identical issue has been introduced in `mintGreat` function.

Gas consumption (fixed)

These issues have been fixed and are not present in the latest version of the code.

- `ProphetAttributes` struct is declared at lines 27-33 in `Prophets`. It occupies five slots in storage, however, one can tightly pack its fields to reduce storage footprint.
- We recommend directly access `bablLoot` field instead of reading the whole `ProphetAttributes` struct to memory at lines 116-118 in `Prophets.claimLoot`.

Misleading comment

Comments at lines 24-26 of `ProphetsArrival` contract incorrectly state dates as November rather than December. Comments in the updated version of the code incorrectly interpret each of four dates as Monday.

Notes

Overpowered role

When calling `ProphetsArrival.mintGreat` function to mint a NFT, the owner decides which bid (signature) to use for it. Therefore, users don't control which token they will receive, or if any at all in.

> *The auction happens off-chain and users don't bid for a specific prophet. The top 1000 prophets will be matched with the top 1000 bids.*

This analysis was performed by Pessimistic:
Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Irina Vikhareva, Project Manager

5 November, 2021