Jupiter Token Security Analysis

by Pessimistic

Published: April 20, 2021

Abstract

In this report, we consider the security of token smart contract of Jupiter 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 Jupiter token contract. We performed our audit according to the procedure described below.

The audit showed only a few issues of low severity. They do not endanger the security of the project.

The project has no public documentation.

The token contract is deployed mainnet.

General recommendations

We recommend adding public documentation to the project.

Project overview

Project description

For the audit, we were provided with the code of Jupiter token contract. The contract has already been deployed to Ethereum mainnet at address 0x4b1e80cac91e2216eeb63e29b957eb91ae9c2be8.

The code has detailed comments but no public documentation.

The total LOC of audited sources is 84.

Token details:

Name:	Jupiter
Symbol:	JUP
Decimals:	18
Total supply:	1 000 000 000 000 000 000 000 000 000

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 smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

The audit showed no issues of medium severity.

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

No public documentation

The project has no public documentation. Since the contract's code is verified and is well-annotated, all the technical information required for integration can be easily accessed by developers. However, the lack of documentation can be confusing for users.

We strongly recommend adding public documentation to the project.

Code quality

- Consider emitting events with `emit` instruction at lines 53 and 68. Calling evens directly does not work with mentioned version of the compiler `0.6.0`.
- Consider specifying the exact version of Solidity compiler, i.e. `pragma solidity 0.6.12;` since the code will not compile with pragma version `0.6.0` or `0.7.0` and later.
- Consider declaring visibility of variables explicitly and setting it to `internal` at lines 21, 23, and 25.
- Consider declaring visibility of functions as `external` instead of `public` at lines 35, 39, 43, 51, 57, and 61.

Gas consumption

Consider declaring `totalSupply_` variable at line 25 as `immutable` to reduce gas consumption.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Daria Korepanova, Security Engineer
Boris Nikashin, Analyst

April 20, 2021