Opium v2
Security Analysis

by Pessimistic

This report is public

January 26, 2022

Abstract 2
Disclaimer 2
Summary 2
General recommendations 2
Project overview 3
  Project description 3
  Codebase update 3
  Codebase update 2 3
Procedure 4
Manual analysis 5
  Critical issues 5
  Medium severity issues 5
    Tests issues 5
  Low severity issues 6
    Code quality 6
    Project management (fixed) 6
  Gas consumption (fixed) 6
  Notes 7
    Roles control 7

Abstract

In this report, we consider the security of smarts contracts of Opium 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 Opium v2 smart contracts. We performed our audit according to the procedure described below.

The audit showed issues with tests and a few issues of low severity.

The project has a documentation. The codebase is thoroughly covered with NatSpec comments. The overall code quality is good.

After the initial audit, the codebase was updated. A few issues were fixed, new tests were added to the project, a few old tests were removed.

After the recheck, another codebase update was performed. A few issues were fixed, new tests were added to the project.

General recommendations

We recommend adding more tests to improve code coverage.

Project overview

Project description

For the audit, we were provided with Opium v2 project on a public GitHub repository, commit 24b2d653bade302d522a5b588595e9728f6f9995.

The project has the documentation.

One test out of 71 does not pass constantly, another one does not pass inconsistently. The overall code coverage is 48.4%.

The total LOC of audited sources is 1540.

Codebase update

After the initial auidt, the codebase was updated. For the recheck, we were provided with commit 206884b078af7f6fd028eba011de585357d38c5b.

In this update, some issues were fixed but a new one was introduced. Also, new test were added to the project, some older tests were replaced. All 106 tests pass, the overall code coverage decreased to 40.8%.

Codebase update 2

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 3f89b8d9b6c9a3e67311fd379cfb81556a4b6bfb.

In this update, some issues were fixed. Also, new test were added to the project. All 111 tests pass, the overall code coverage increased to 68.67%.

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 codebase with automated tools: Slither and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
  - We manually analyze codebase 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 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 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.

Tests issues

One test constantly fails. Another test relies on external conditions and does not pass only in some circumstances. The overall code coverage is low.

Testing is crucial for code security and audit does not replace tests in any way. We highly recommend covering the codebase with tests and making sure that all tests pass and the code coverage is sufficient. We also recommend using strict conditions for tests so that they do not rely on external factors.

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 the future versions of the code. We recommend taking them into account.

Code quality

- There are several common patterns used to implement upgradeability. Usually, storage is a separate contract, and proxy inherits from the storage. In the current implementation, **Registry** contract does not need to inherit from **RegistryStorage** contract.

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

- The CEI (checks-effects-interactions) pattern is violated in **Core** contract at lines 388, 464, 511, 627, 700, and 707. We highly recommend following CEI pattern to increase the predictability of the code execution and protect from some types of re-entrancy attacks.

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

- Role codes in **LibRoles** contract are inconsistent with error codes in the comments in other contracts:
  - At lines 21 and 24 of **LibRoles** contract, roles `REGISTRY_MANAGER_ROLE` and `CORE_CONFIGURATION_UPDATER_ROLE` have codes `RL10` and `RL18` respectively. However, in **RegistryManager** contract, error codes for these roles are `M1` and `M2` respectively.
  - At lines 58–68 of **LibRoles** contract, roles have codes ranging from `RL11` to `RL17`. However, appropriate error codes mentioned at lines 26–33 of **Registry** contract and used in **RegistryStorage** contract have inconsistent numbers.

> *Comment from developers: We do not want to pair numbers in error codes to the numbers in the roles naming, as every file must have its own order and naming of the errors, we consider this inconsistency as non-important.*

Project management (fixed)

The project compiles with warnings. We recommend addressing the warnings raised by the compiler.

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

Gas consumption (fixed)

In **RegistryEntities** contract, `__gapFour` variable at line 28 does not affect the upgradeability of the contract, but occupies the second slot of storage. Consider removing it.

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

Notes

Roles control

There are many technical roles used in the project. Consider setting up a multisig for each of them.

This analysis was performed by Pessimistic:

Daria Korepanova, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

January 26, 2022