Spiral DAO Lockers Security Analysis

by Pessimistic

This report is public

October 16, 2023

Abstract

In this report, we consider the security of smart contracts of Spiral DAO Lockers 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 Spiral DAO Lockers smart contracts. We described the audit process in the section below.

The initial audit showed several issues of medium severity: Outdated documentation, Incorrect pending rewards calculation, Infinite lock duration and Lack of tests. Also, several low-severity issues were found.

After the initial audit, the developers provided a new version of the code. In that update, the developers added NatSpec comments and improved code style of the project. However, the overall code quality is mediocre. The project does not contain any major vulnerabilities. But it still contains medium-severity issues. In addition to this, a new low severity issue has been found.

General recommendations

We recommend fixing the mentioned issues and adding general documentation for the entire project in addition to NatSpec comments.

Project overview

Project description

For the audit, we were provided with Spiral DAO Lockers project on a private repository, commit af0d61ee53f2689f527a579ff348143a983b9702.

The scope of the audit included the following contracts:

- **LiquidLocker/Lockers/LiquidLockerMAV.sol**;
- **LiquidLocker/Lockers/LiquidLockerCRV.sol**;
- **LiquidLocker/LockerStaking.sol**;
- **LiquidLocker/LockerMaster.sol**;
- **LiquidLocker/Mocks/LiquidLockerMock.sol**;
- **LiquidLocker/Mocks/stLockerTokenMock.sol**;
- **libraries/SafeLocker.sol**.

The documentation for the project is outdated.

All 48 tests pass successfully. The code coverage cannot be computed correctly since not all files are covered with tests.

The total LOC of audited sources is 687.

Codebase update

After the initial audit, the developers provided us with a new commit c595dcb45a80965765a4d8ae3d5bd3e747031637. In that update, the developers slightly reorganized the project structure, added NatSpec comments to the functions and fixed some medium severity and code style issues. However, a new issue was introduced in the update (see L04).

All 65 tests pass successfully. The code coverage cannot be computed correctly since not all files are covered with tests.

Audit process

We started the audit on July 26, 2023 and finished on August 3, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. During the audit, we stayed in touch with them and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- The upgradeable contracts are initialized safely;
- The libraries are integrated correctly;
- The contracts with the external calls to arbitrary contracts are safe;
- The correctness of calculations in the staking contract;
- The overall safety of user funds (see M03 issue).

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts.

We asked the developers to run tests since it required a complex project setup. The code coverage could not be calculated correctly since not all files are covered with tests.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

On August 8, 2023, the developers provided us with an updated version of the code. In this update, they fixed some of the issues from our report, reorganized the project structure, and added NatSpec comments.

We reviewed the updated codebase and found one new low severity issue (see L04). We ran the aforementioned tools and manually verified the output.

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.

The audit showed no critical issues.

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. Outdated documentation

The project contains outdated documentation. We recommend updating it to make maintaining process easier.

The developers added NatSpec comments. However, we still recommend adding general documentation.

M02. Incorrect pending rewards calculation (fixed)

`pendingRewards` function of the **LockerStaking** contract calculates the amount of tokens that the user is able to claim. However, line 237 contains a logical issue. If the pool is updated in the same block when this function is called, then `poolInfo.lastRewardBlock` variable will be equal to the `block.number`. As a result, the user pending rewards will be zero even if the user has not claimed them.

The issue has been fixed in commit 1dede9529475977982f1278bf41aac8518db3e6a.

M03. Infinite lock duration (commented)

In the current implementation of the **LiquidLockerMAV** contract, every time a user locks some tokens, the lock period of all users is getting increased. As a result, any user is able to infinitely lock tokens of everyone. Note that the implementation of the **LockerMaster** contract could be changed in the future by the owners.

Following the initial audit, developers introduced a deadline for user locks. However, the lock duration for all of the tokens is extended each time any user locks their funds.

> *Comment from the developers: Users do not have locks, only the DAO has a lock, and it is constantly expanding in terms of time and quantity - it is a chosen strategy.*

M04. Lack of tests (commented)

The project tests are stored in a separate private repository. At the time of the audit, there are tests not for all of the files in the scope. To ensure project security, we strongly recommend expanding test coverage to cover more scenarios.

After the audit, the developers added a new liquid locker that inherits the same base contracts and tested this locker. As a result, the base contracts from the current scope of the audit are now covered with tests.

M05. **Project roles** (commented)

The project heavily relies on several crucial roles.

The `Service` role has the following powers:

- Disabling liquid lockers;
- Updating the unlock time, which is important for the `CRV` locker;
- Calling the `exec` function. This allows the `Service` to make an arbitrary call to any address in case of the `CRV` locker.

The `Owner` role has all the powers of the `Service` role. In addition to this, it is able to do the following:

- Enabling liquid lockers;
- Adding funds to the locker without minting tokens;
- Releasing tokens (if none are locking tokens, see M03 issue).

The owner of the proxy is able to update **LockerMaster** contract code to any code since the system is upgradeable.

The `Gov` role has the following powers:

- Setting **LockerMaster** and **RewardVault** contracts;
- Changing reward per block and maximum reward per block.

We recommend designing contracts in a trustless manner. If that is not possible, we recommend carefully reviewing role powers and considering setting up multisig wallets for the most crucial of them.

According to the developers, such a model of centralization forms the basis of a DAO. However, this might be subject to change in the future.

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 logic (fixed)

In the `lock` function of the **LockerMaster** contract, there is an inconsistency between valid `amount` argument values. According to line 143, `amount` should be greater than zero. However, lines 145-148 are executed only when `amount` is greater than zero. That behavior implies that the `amount` value equal to zero might be valid.

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

L02. Code logic (fixed)

The `release` function of the **LiquidLockerCRV** contract incorrectly calculates the actual amount of tokens that could be sent. If the sum of the amounts of the staked tokens and tokens on the contract is less than the provided `amount` argument, then `actualOutAmount` tokens will have an incorrect value.

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

L03. Unprotected initializer (addressed)

The `init` function of the **LockerMaster** contract is not protected by a role. It is expected that this function will be called only once after the contract is deployed. However, a malicious agent can frontrun the `init` function call and ruin the contract initialization. This is not a critical vulnerability since it could happen only prior to anyone using the contract. Hence, in that case, the attacked contract can be redeployed.

According to the developers, initialization is made via `updateAndCall` method. Hence, the initialization cannot be frontrunned.

L04. Missing argument (fixed)

The **SafeLocker** library contains a `lockedAmount` function intended to perform a `delegatecall` to the adapter. However, all existing adapter implementations require a `user` argument, which the library does not have. It is important to note that this issue does not lead to a vulnerability since `lockedAmount` function is never used. Nevertheless, we still recommend fixing it.

The issue has been fixed in commit ecb528ecce56b0668b6e4f0f6625912dbead8dee.

Notes

N01. Wrapped token should not be transferrable (commented)

When user stakes tokens in the **LockerStaking** contract, they receive `wrappedToken` tokens. These tokens should not be transferrable for the correct staking calculations. In the current implementation, it is not a problem since the **LockerTokenWrapper** contract will be used for the implementation of the `wrappedToken` tokens.

The developers decided not to use token wrappers in the next version of adapters.

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Nikita Kirillov, Security Researcher
Ivan Gladkikh, Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, Founder

October 16, 2023