Spiral DAO SDT Adapter Security Analysis

by Pessimistic

This report is public

September 8, 2023

Abstract

In this report, we consider the security of smart contracts of Spiral DAO SDT Adapter 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. A single 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, a security audit is not investment advice.

Summary

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

The initial audit showed five issues of medium severity: Insufficient documentation, Incorrect delegateTotal, Incorrect pending rewards calculation, Insufficient test coverage (was moved to the Notes section) and Project roles. Also, several low-severity issues were found.

The overall code quality is mediocre. The project does not contain any critical vulnerabilities. However, it has several medium severity issues. Additionally, the project has a limited number of tests and lacks comprehensive documentation.

After the initial audit, the developers provided an update. In that update, they either fixed or commented on all of the issues. As a result, the M04 issue was moved to the notes section.

General recommendations

We recommend adding comprehensive documentation and improving NatSpec coverage.

Project overview

Project description

For the audit, we were provided with Spiral DAO SDT Adapter project on a private GitHub repository, commit c595dcb45a80965765a4d8ae3d5bd3e747031637.

The scope of the audit included the following contracts:

- **LiquidLocker/Adapters/SDT.sol**;
- **LiquidLocker/Special/LockerMasterSDT.sol**;
- **LiquidLocker/Special/StakingSDT.sol**;
- dependencies of the aforementioned contracts.

The project contains NatSpec comments as the documentation. However, the audited module does not have them.

All 65 tests pass successfully. During the recheck, it was observed that the coverage cannot be calculated correctly (see N01 note).

The total LOC of audited sources is 573.

Codebase update

After the initial audit, we were provided with commit number 15c11ba1bca1f362791298a6003514b29c8a6700. In that update, the developers either commented or fixed all of the issues. In addition to that, they added new tests. As a result, all 73 tests pass.

Audit process

We started the audit on August 22, 2023, and finished on August 29, 2023.

We inspected the materials provided for the audit. We could not arrange an introduction meeting with the developers. However, during work, we stayed in touch with them. Based on the code and on the previously audited adapters from the same module, we conducted the following properties that should be checked:

- The libraries are integrated correctly;
- The contracts with external calls to arbitrary contracts are safe;
- The correctness of calculations in the staking contract;
- The overall safety of user funds;
- The user cannot delegate more tokens than they staked.

We manually analyzed all the contracts within the scope of the audit and checked their logic. In addition to this, 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 verified the output of the tools and shared Semgrep results with the developers.

We asked the developers to run the tests and calculate code coverage.

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

After the initial audit, we discussed the results with the developers. On September 4, 2023, the developers provided us with an updated version of the code. In this update, they either fixed or commented on all of the issues and added new tests.

We reviewed the correctness of all provided fixes. As a result, we decided to move the M04 issue to the N01 note and provided the reasoning in the comment there. Moreover, we removed the L06, which was identified as a false positive.

We ran the aforementioned tools and verified their 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. Insufficient documentation (commented)

The audited part of the project is not covered with NatSpec comments. We recommend adding them to make the maintenance process easier.

> *Comment from the developers:* *Acknowledged.*

M02. Incorrect delegateTotal (fixed)

In the **StakingSDT** contract, within the `delegate` function, there is an incorrect variable at line 299. This line uses the `delegateTotal` variable, which might not hold the expected value when a user's delegation has ended. At line 296, the `user.delegateTotal` variable is correctly set to zero, but the same adjustment is not applied to the delegateTotal variable. This discrepancy could potentially result in unintended behavior and the transaction being reverted.

The same issue is present at line 354 in the `delegateToPaladin` function.

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

M03. Incorrect pending rewards calculation (fixed)

The `pendingRewards` function of the **StakingSDT** contract calculates the amount of tokens that the user is able to claim. However, line 260 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's pending rewards will be zero even if the user has not claimed them.

The issue was fixed in commit 5ee48109af62fcc4e91653707375b7cd702e2a5a.

M05. Project roles (addressed)

The project heavily relies on several crucial roles.

The `Service` role has the following powers:

- Disabling the SDT adapter;
- Updating the unlock time;
- Calling the `exec` function. This allows the `Service` to make an arbitrary call to any address with a predetermined signature.

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

- Enabling the SDT adapter;
- Adding funds to the adapter without minting tokens;
- Releasing tokens.

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

The `Delegator` role is able to do the following:

- Boost rewards using locked tokens;
- Enable and disable the warden. That allows the `delegator` to give approval to any address;
- Pledge tokens to the Paladin protocol.

Note, that in the current implementation, the delegator is a **LockerStaking** smart contract.

The `Gov` role has the following powers:

- Setting **LockerMaster** and **RewardVault** contracts;
- Changing the reward per block and maximum reward per block;
- Setting Paladin warden and maximum delegation time.

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.

Comment from the developers:

- `Service` *role cannot call arbitrary data on any address. This role can only call* `execLock` *on adapter with hardcoded addresses and/or payload signatures;*
- `updateUnlockTime` *has been moved to* `Gov` *role;*
- `Owner` *role represented as protocol multi-signature wallet that consists of both team members and third-parties to ensure non-malicious actions. Also, this role is owner of* `ProxyAdmin`*;*
- `Delegator` *is designed to be contract, which address is setting by* `Gov`*;*
- `setPaladinWarden` *function called by* `Gov` *and permission to enable/disable warden is acts like 2FA to ensure state of initial setup;*
- `Gov` *is equivalent of* `Owner`*.*

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. Amount values for multiclaim (fixed)

In the **SDT** adapter, there is an option to perform a multiclaim action, allowing for the claiming of multiple tokens. However, as per lines 95-96, this action uses the same amount for all claims. In contrast, in `multiclaim` transactions of the **sdtFeeDistributor** contract, different claim values are typically involved. To enhance user experience and reduce gas costs, we recommend supporting distinct claim values within the multiclaim action.

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

L02. Code logic (commented)

A number of protocols implement an emergency withdrawal function where users acknowledge potential losses and retrieve available funds from the protocol. In **StakingSDT.sol**, an `emergencyWithdraw` function exists. However, it currently restricts the withdrawal of funds when users have any tokens delegated. It is possible to implement a withdrawal mechanism even in the case of delegation, utilizing a logic similar to the `unstake` function.

> *Comment from the developers: This function has been designed to be as simple as possible. In this case, partial time blocking is an acceptable inconvenience. On the other hand, if it will become more complex, the main gist will be lost.*

L03. Function visibility (fixed)

Consider declaring functions as `external` instead of `public` where possible in order 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. Incorrect payload size check (fixed)

In **SDT.sol**, at line 94, there is a verification of the payload size, which is later decoded into arguments. As per the requirement, the minimum payload size should be 192 bits. However, in practice, the minimum size for a correct payload is 224 bits.

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

L05. 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 the `lockedAmount` function is never used. Nevertheless, we still recommend fixing it.

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

Notes

N01. Incorrectly calculated test coverage

The project contains tests. However, we recommend reviewing the branch coverage of the audited files and the overall coverage of the **SDT.sol** file.

As per the developers' feedback, there are issues with code coverage in the project. The coverage library requires either disabling optimization or converting the code to YulIR. Disabling optimization is not a viable option because the contracts do not compile without optimization. On the other hand, the alternative approach of converting the code to YulIR leads to problems with achieving branch coverage.

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, Founder

September 8, 2023