PROTOFIRE

SMART CONTRACT AUDIT REPORT

SparkDex Distributor

Version: 1.0

Protofire
December, 2024

Table Of Contents

The SparkDex Distributor report

The audit report was prepared for the **SparkDex Team** and includes following Github repository:

https://github.com/SparkDEX/sparkdex-contracts audited commit 2d882cd2ddb893437111313478166670447e9006

The code repository does not contain unit tests.

Documentation was not provided.

Report Update - 1

The **SparkDex Team** team has updated its code in accordance with the issues identified in the first review.

The updated code was reviewed at the following commit: https://github.com/SparkDEX/sparkdex-contracts/commit/445668837076ae9b372389b2fe3776ce54532b4e

Procedure

The audit includes the following procedures:

- code analysis by the Slither static analyzer, followed by manual analysis and selection of problems
- manual code analysis, including logic check
- checking the business logic for compliance with the documentation (or white paper)

In addition, during the audit, we also provide recommendations on optimizing smart contracts and using best practices.

Summary of audit findings

Severity	Count
HIGH	0
MEDIUM	0
LOW	2
INFORMATIONAL	1
TOTAL	**3**

Severity classification

High severity issues

High severity issues can lead to a direct or indirect loss of funds by both users and owners, a serious violation of the logic of the contract, including through the intervention of third parties. Such issues require immediate correction.

Medium severity issues

Medium severity issues may cause contract functionality to fail or behave incorrectly. Also, medium severity issues can cause financial damage. Such issues require increased attention.

Low severity issues

Low severity issues do not have a major security impact. It is recommended that such issues be taken into account.

Informational severity issues

These issues provide general guidelines for writing code as well as best practices.

Smart contract TokenDistributor [SDD-01]

The TokenDistributor contract is used for distributing tokens among users. Distribution is conducted on a daily basis. After a distribution is created, recipients have a specified time to claim their tokens. If a user does not claim their tokens within the allotted time, those tokens will be returned to the owner.

The contract supports the distribution of up to five tokens simultaneously.

ID: **SDD01-01**	Severity: **Low**	Status: **Fixed**

Gas savings

1. The `require(address(_token) != address(0)` statement on L141 looks redundant because in case of zero-address the function `withdrawEmergencyToken()` will fail automatically on transfer attempt.
2. In the `withdrawExpiredToken()` function, lines 128-129 can be swapped so that the `uint256 amount = totalExpired[_token]` variable is declared first, and then the `require()` check is performed using the `amount` variable. This will avoid redundant reads from the contract storage and will save gas.

ID: **SDD01-02**	Severity: **Info**	Status: **Fixed**

Limit bufferTime

The `bufferTime` global variable should have an upper limit in the constructor. If this variable does not have an upper limit, functions `setRecipients()` and `claim()` may have issues with a gas block limit in case of a lot of unprocessed indexes.

Smart contract AddressSet [SDD-02]

The AddressSet library provides functionality for storing and interacting with arrays of addresses.

ID: **SDD02-01**	Severity: **Low**	Status: **Fixed**

Gas savings

In the `clear` function in a cycle there are multiple reads of the `_state.list.length` global (storage) variable.
Consider using local variable for the array length.

ID: **SDD02-02**	Severity: **Info**	Status: **Not relevant**

Custom implementation

This contract has an open source solution from the OpenZeppelin team. It is better to use open source and battle tested solutions instead of custom implementations.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.1.0/contracts/utils/structs/EnumerableSet.sol

The issue is no longer relevant.

Conclusion

During the audit 2 low severity issues were found.

Fixing issues and covering them with tests is strongly advised.

Conclusion Update. All issues have been fixed in the updated code.

Disclaimer

Note that smart contract audit provided by Protofire is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, Protofire recommends proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Smart contract audit provided by Protofire shall not be used as investment or financial advice.