SOLIDIFIED

Audit Report for Vortex - September 02 2021

Summary

Audit Report prepared by Solidified covering the Vortex smart contracts.

Process and Delivery

Two (2) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief on 30 August 2021.

Audited Files

The source code has been supplied in the form of a GitHub repository:

https://github.com/BlockzeroLabs/vortex-contracts

Commit number: `9687b422bf42e7f1e9baab83b9cbf8c2dcedb41f`

The scope of the audit was limited to the following files:

```
contracts/
├── Portal.sol
└── interfaces
    └── IMigrator.sol
```

Intended Behavior

The smart contracts implement a staking solution with rewards in several reward tokens that can be claimed.

SOLIDIFIED

Audit Report for Vortex - September 02 2021

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	Medium	-
Test Coverage	High	-

Coverage report:

```
----------------------|----------|----------|----------|----------|----------------|
File                  |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------------|----------|----------|----------|----------|----------------|
 contracts/           |    92.93 |    57.69 |    78.57 |    92.93 |                |
  Portal.sol          |    92.93 |    57.69 |    78.57 |    92.93 |... 191,192,208 |
 contracts/interfaces/ |      100 |      100 |      100 |      100 |                |
  IMigrator.sol       |      100 |      100 |      100 |      100 |                |
----------------------|----------|----------|----------|----------|----------------|
All files             |    92.93 |    57.69 |    78.57 |    92.93 |                |
----------------------|----------|----------|----------|----------|----------------|
```

SOLIDIFIED

Audit Report for Vortex - September 02 2021

Issues Found

Solidified found that the Vortex contracts contain no critical issue, 1 major issues, 2 minor issue in addition to 3 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Users can stake, harvest, and withdraw their stake in the same transactions	Major	Pending
2	Reward tokens are assumed to have 18 decimals	Minor	Resolved
3	Missing input check for array length mismatch	Minor	Resolved
4	Potential gas optimization in loops	Note	Resolved
5	Consider improved error handling for withdraw()	Note	Resolved
6	Consider using external instead of public	Note	Resolved

SOLIDIFIED

Audit Report for Vortex - September 02 2021

Critical Issues

No critical issues were found.

Major Issues

1. Users can stake, harvest, and withdraw their stake in the same transactions

---

Users can claim rewards tokens without leaving any staking tokens in the contract by staking tokens, harvesting, and withdrawing their stake. This could be used, for example, in conjunction with a flashloan to game the reward system.

Recommendation
Consider introducing a minimum number of blocks for staking or a timelimit before which withdrawals are not possible.

Minor Issues

2. Reward tokens are assumed to have 18 decimals

The contract uses hardcoded value `1e18` as the denominator for fixed-point arithmetic operations. This means that the contract can only be used with tokens that have 18 decimals.

Recommendation
Consider using the token's real `decimal` value or make sure the contract is only used with an 18-decimal token.

3. Missing input check for array length mismatch

The function `addReward()` takes an array of rewards as a parameter. This should be of the same length as the rewardsToken. However, this precondition is never checked.

Recommendation
Consider adding the following statement:

```
require (rewards.length == rewardsToken.length);
```

Informational Notes

4. Potential gas optimization in loops

The smart contract includes quite a few functions with costly loop operations. Whilst these are contained in terms of iterations and do not represent a gas limit risk, they are quite costly.

A simple gas optimization could be applied. Iteration over arrays in the codebase contains a check on `.length` of a non-memory array inside the condition of the for loops.
As a result, this procedure consumes a comparatively larger amount of gas for every iteration.

Recommendation
Consider using a local variable with the length of the required array in the condition of the for loop instead of the state variable directly.
This will optimize gas usage.

5. Consider improved error handling for withdraw()

The function `withdraw()` in the allows users to pass an `amount` argument.
The function does implement a check to ensure that the argument passed is greater than 0. However, it doesn't ensure if the `amount` argument is less than or equal to the user's actual staked balance. This check will be enforced implicitly through underflow protection, however, the resulting error message would be confusing.

Recommendation
Consider including an input validation which ensures that the `amount` argument passed to this function must be less than or equal to the caller's actual staked balance.

This ensures that the above-mentioned scenario is handled effectively and the function doesn't lead to an unknown revert.

6. Consider using external instead of public

Functions that are never called from within the contract and are only supposed to be called from outside the contract can be assigned an `external` instead of `public` visibility.

Recommendation

Consider changing visibility where appropriate.

Audit Report for Vortex - September 02 2021

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Vortex or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.