VIDMA

FERRUM NETWORK

SMART CONTRACT AUDIT

Project: Ferrum Network
Date: September 27th, 2022

TABLE OF CONTENTS

SUMMARY

Vidma is pleased to present this audit report outlining our assessment of code, smart contracts, and other important audit insights and suggestions for management, developers, and users.

An audited contract is a staking smart contract that distributes rewards. The owner has the ability to add extra tokens in the pool as rewards by checking the marginal rewards. The marginal reward is the number of tokens in the contract that are not already added to the reward or stake. There is a possibility to set early withdrawal rewards once a new reward portion is added by the user who added them. The owner can change the early withdrawable rewards once the marginal rewards are recalculation. In this case, the owner can reassign the early withdrawable rewards number.

During the audit process, the Vidma team found several issues. A detailed summary and the current state are displayed in the table below.

Severity of the issue	Total found	Resolved	Unresolved
Critical	0 issues	0 issues	0 issues
High	3 issues	3 issues	0 issues
Medium	2 issues	2 issues	0 issues
Low	6 issues	6 issues	0 issues
Informational	3 issues	3 issues	0 issues
Total	**14 issues**	**14 issues**	**0 issues**

After evaluating the findings in this report and the final state after fixes, the Vidma auditors can state that the contracts are fully operational and secure. Under the given circumstances, we set the following risk level:

Our auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. This approach helps us adequately and sequentially evaluate the quality of the code. Code style, optimization of the contracts, the number of issues, and risk level of the issues are all taken into consideration. The Vidma team has developed a transparent scoring system presented below.

Severity of the issue	Resolved	Unresolved
Critical	1	10
High	0.8	7
Medium	0.5	5
Low	0.2	0.5
Informational	0	0.1

Please note that the points are deducted out of 100 for each and every issue on the list of findings (according to the current status of the issue). Issues marked as "not valid" are not subject to point deduction.

Based on the **overall result of the audit**, the Vidma audit team grants the following score:

In addition to manual check and static analysis, the auditing team has conducted a number of integrated autotests to ensure the given codebase has an adequate performance and security level.

The test results and the coverage can be found in the accompanying section of this audit report.

Please be aware that this audit does not certify the definitive reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by the Vidma audit team. If the code is still under development, we highly recommend running one more audit once the code is finalized.

SCOPE OF WORK

FERRUM NETWORK

With the mission of breaking down barriers to mass adoption, Ferrum empowers the industry by reducing friction and bringing startups and established networks closer together.

Within the scope of this audit, two independent auditors thoroughly investigated the given codebase and analyzed the overall security and performance of the smart contracts.

The main part of the audit was conducted from August 1, 2022 to September 15, 2022. The outcome is disclosed in this document. The additional review of the fixes was conducted on Tuesday, Sep 27.

The scope of work for the given audit consists of the following contracts:

- FestakedWithReward;
- FestakedOptimized;
- Festaked.Library;
- SafeAmount.

The source code was taken from the following **source**:
https://github.com/ferrumnet/open-staking/blob/v2/v2/contracts/staking/legacy/FestakedWithReward.sol

Initial commit submitted for the audit:
b84eb4dc26546e3ac0b24893d5768e6f98882771

Last commit reviewed by the auditing team:
a6c0f3b6dd8d934518621199fd8da4a54145dc07

WORKFLOW OF THE AUDITING PROCESS

Vidma audit team uses the most sophisticated and contemporary methods and well-developed techniques to ensure contracts are free of vulnerabilities and security risks. The overall workflow consists of the following phases:

Phase 1: The research phase

Research

After the Audit kick-off, our security team conducts research on the contract's logic and expected behavior of the audited contracts.

Documentation reading

Vidma auditors do a deep dive into your tech documentation with the aim of discovering all the behavior patterns of your codebase and analyzing the potential audit and testing scenarios.

The outcome

At this point, the Vidma auditors are ready to kick off the process. We set the auditing strategies and methods and are prepared to conduct the first audit part.

Phase 2: Manual part of the audit

Manual check

During the manual phase of the audit, the Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract. The initial commit as stated in the agreement is taken into consideration.

Static analysis check

Static analysis tools are used to find any other vulnerabilities in smart contracts that were missed after a manual check.

The outcome

An interim report with the list of issues.

Phase 3: Testing part of the audit

Integration tests

Within the testing part, Vidma auditors run integration tests using the Truffle or Hardhat testing framework. The test coverage and the test results are inserted in the accompanying section of this audit report.

The outcome

Second interim report with the list of new issues found during the testing part of the audit process.

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For simplicity in reviewing the findings in this report, Vidma auditors classify the findings in accordance with the severity level of the issues (from most critical to least critical).

All issues are marked as “Resolved” or “Unresolved”, depending on if they have been fixed by Ferrum Network or not. The issues with “Not Valid” status are left on the list of findings but are not eligible for the score points deduction.

The latest commit with the fixes reviewed by the auditors is indicated in the “Scope of Work” section of the report.

The Vidma team always provides a detailed description of the issues and recommendations on how to fix them.

Classification of found issues is graded according to 6 levels of severity described below:

Critical

The issue affects the contract in such a way that funds may be lost or allocated incorrectly, or the issue could result in a significant loss.

Example: Underflow/overflow, precisions, locked funds.

High

The issue significantly affects the ability of the contract to compile or operate. These are potential security or operational issues.

Example: Compilation errors, pausing/unpausing of some functionality, a random value, recursion, the logic that can use all gas from block (too many iterations in the loop), no limitations for locking period, cooldown, arithmetic errors which can cause underflow, etc.

Medium

The issue slightly impacts the contract's ability to operate by slightly hindering its intended behavior.

Example: Absence of emergency withdrawal of funds, using assert for parameter sanitization.

Low

The issue doesn't contain operational or security risks, but are more related to optimization of the codebase.

Example: Unused variables, inappropriate function visibility (public instead of external), useless importing of SCs, misuse or disuse of constant and immutable, absent indexing of parameters in events, absent events to track important state changes, absence of getters for important variables, usage of string as a key instead of a hash, etc.

Informational

Are classified as every point that increases onboarding time and code reading, as well as the issues which have no impact on the contract's ability to operate.

Example: Code style, NatSpec, typos, license, refactoring, naming convention (or unclear naming), layout order, functions order, lack of any type of documentation.

MANUAL REPORT

Compilation issues

High | MH – 01 | Resolved

Contracts don't compile because of the following problems in the codebase:

- Lack of import FestakedLib in the RewardAdder contract;
- In contract FestakedWithReward there is a typo in param in the natspec annotation for function *addMarginalReward withrawableAmount* -> *withdrawableAmount*

Recommendation:

Consider adding missing import and fixing the natspec.

Comparison with not initialized storage in the constructor

High | MH – 02 | Resolved

In the contract FestakedOptimized *stakingEnds_*, *withdrawStarts_*, and *withdrawEnds_* variables are validating. They are compared with the state not initialized variables *stakingStarts*, *stakingEnds*, and *withdrawStarts* which will lead to impossible pass the require correct.

Recommendation:

Consider renaming *stakingStarts*, *stakingEnds*, and *withdrawStarts* in lines 37-39 into *stakingStarts_*, *stakingEnds_*, and *withdrawStarts_*.

Re-Audit:

Still present empty storage comparison with *stakingEnds* in line 37. Consider using input param *stakingEnds_* instead.

stakingStarts can be set bigger than *stakingEnds*

High | MH – 03 | Resolved

In the contract FestakedOptimized there is a check if the *stakingEnds* is bigger than *stakingStarts*. But in a case, if the passed *stakingStarts* value will be less than the current block.timestamp it will be reassigned in line 45 and potentially can be set less than the *stakingEnds*.

Recommendation:

Consider adding require to check if the *stakingStarts* is >= than the current *block.timestamp* and remove the if-else statement in lines 44-48.

Possibility to set early rewards greater than total rewards for tokens with fee on transfer in *addReward()* method from FestakedLib

Medium | MM – 01 | Resolved

In method *addReward()* it's possible to increase early reward to be greater than total reward amount for tokens with fee on transfer. It can lead to improper early reward calculation.

Recommendation:

Consider to move require statement:

```
require(withdrawableAmount <= rewardAmount, "Festaking: withdrawable
amount must be less than or equal to the reward amount");
```

from line 96 to line 99 after actual reward amount calculation.

Re-Audit:

Move require from 118 line below line 120 *rewardAmount = payMe (from, rewardAmount, rewardTokenAddress)*;

Incorrect updating of *earlyWithdrawReward* value

Medium | MM – 02 | Resolved

In the FestakedWithReward *addMarginalReward()* there is the possibility to update the total value of the rewards based on how many reward tokens the contract has on its balance. The second possibility is to update the *earlyWithdrawReward* value here but there is no check if the *earlyWithdrawReward* won't exceed the max available number of the total rewards it may lead to incorrect calculation of the rewards in early withdrawing.

Recommendation:

Consider moving the setting of the *earlyWithdrawReward* into the *FestakedLib.addMarginalReward()* and add a check if the new value doesn't exceed the new total rewards amount.

Optimization in FestakedOptimized constructor

Low | ML – 01 | Resolved

- In the FestakedOptimized constructor *stakingStarts_* value is set and should be not less than the current *block.timestamp* what is shown in lines 42-46. It is possible to avoid if-else statement here and just check if the *stakingStarts_ >= block.timestamp* in the require in line 41. It will decrease gas usage on the deployment:

```
require(stakingStarts_ >=  block.timestamp, "Festaking:
incorrect start time");
```

- There are bunch of require statements in the constructor. It make sense to put all require into the begging of the constructor, to avoid extra gas fee in case if some of the required will revert.

Recommendation:

- Consider change the require and delete if-else.
- Consider put all require statements into the begging of the constructor.

Re-Audit:

New fixes didn't resolve the problem fully but triggered some new issues as empty storage comparison and the presence of if-else statements lead to the possibility of setting stakingStarts bigger than stakingEnds.

Functions visibility optimization

Low | ML – 02 | Resolved

There is function *withdraw()* in the contract FestakedWithReward that is never called in the contract itself with visibility 'public' so it can be marked as external.

Recommendation:

Consider changing visibility from public to external to safe gas usage on calling.

Require statement with tautology

Low | ML – 03 | Resolved

In the library FestakedLib function *addReward()* there is a comparison if the uint256 *withdrawableAmount* is >= 0 (line 95), which is useless and will always return true in terms of the definition of uint256 type. Another case in the contract FestakedOptimized.constructor() detected the same check for the *stakingCap_* >=0 while *stakingCap_* is uint256 type. In terms of uint256 type comparison to negative number is useless as negative numbers are not included in the range of values valid for uint256.

Recommendation:

Consider fixing the incorrect comparison by changing the value type or the comparison, or delete it.

Floating pragma

Low | ML – 04 | Resolved

The current version of solc in contracts FestakedWithReward, FestakedOptimized, and library FestakedLib are ^0.8.0 and it is better to lock the pragma to a specific version.

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation:

Consider locking pragma for all contracts to a specific version.

Lack of zero-check

Low | ML – 05 | Resolved

In the FestakedOptimized function *stakeFor()* there is no check if the 'staker" address isn't zero address.

Recommendation:

Consider adding a check for zero address.

Useless usage of SafeMath library

Low | ML – 06 | Resolved

In the FestakedLib SafeMath library is used. Starting from 0.8.x solc includes a check of arithmetic operations by default so SafeMath is unnecessary.

Recommendation:

Consider removing the SafeMath library to optimize the contract size and gas usage.

Lack of NatSpec annotations

Informational | MI – 01 | Resolved

Smart contracts FestakedLib, FestakedOptimized, and FestakedWithReward are not fully covered by NatSpec annotations.

Recommendation:

Consider covering by NatSpec all contract methods.

File name and library name do not match

Informational | MI – 02 | Resolved

The library name FestakedLib doesn't match its filename which is Festaked.Library

Recommendation:

Consider changing the name so it matches the filename.

Several contracts in one file

Informational | MI – 03 | Resolved

According to the solidity documentation, it is a good practice to separate each contract into a separate file. In the file FestakedWithReward.sol there are two contracts FestakedWithReward and RewardAdder.

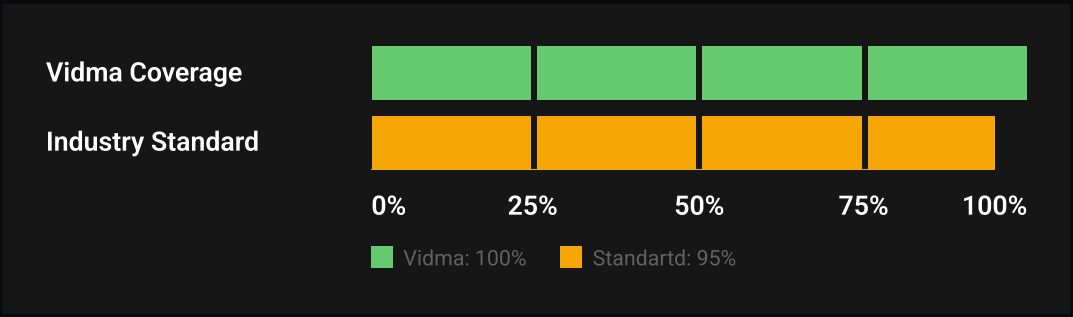
Recommendation:

Consider separating contracts into two different files.

TEST RESULTS

To verify the security of contracts and the performance, a number of integration tests were carried out using the Truffle testing framework.

In this section, we provide both tests written by Ferrum Network and tests written by Vidma auditors.



It is important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the Ferrum Network repository. We write totally separate tests with code coverage of a minimum of 95% to meet industry standards.

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
Festaked.Library.sol	100.00	90.63	100.00	100.00
FestakedOptimized.sol	100.00	92.86	100.00	100.00
FestakedWithReward.sol	100.00	100.00	100.00	100.00
All Files	**100.00**	**92.00**	**100.00**	**100.00**

Test Results

```
FestakedWithReward Test Cases
  FestakedWithReward Deployment Test Cases
    ✓ shouldn't deploy with zero address stake token (43ms)
    ✓ shouldn't deploy with zero address reward token (45ms)
    ✓ shouldn't deploy with staking start time setted to the zero
    ✓ shouldn't deploy if staking end time less than staking
      start time
    ✓ shouldn't deploy if start of the withdraw time less
      than staking end time
    ✓ shouldn't deploy if withdraw end time less than withdraw
      start time
    ✓ should deploy with correct name (383ms)
    ✓ should deploy with correct staking token
    ✓ should deploy with correct reward token
    ✓ should deploy with correct staking start time
    ✓ should deploy with correct staking end time
    ✓ should deploy with correct withdraw start time
    ✓ should deploy with correct withdraw end time
    ✓ should deploy with correct staking cap
```

```
    ✓ should deploy with correct address of reward setter
    ✓ should deploy with correct version of FestakedLib
  FestakedWithReward Adding Rewards Test Cases
    FestakedWithReward Add Reward Test Cases
      ✓ shouldn't add rewards if reward amount equal to the zero
      ✓ shouldn't add rewards if withdrawable amount greater than
        amount of the rewards
      ✓ should add rewards correctly (98ms)
      ✓ should add rewards for token with fee on transfer
        correctly (90ms)
    FestakedWithReward Add Marginal Reward Test Cases
      ✓ shouldn't allow to add marginal reward by not the reward
        setter
      ✓ should add marginal reward correctly (85ms)
      ✓ should allow to set new early withdraw rewards by add
        marginal reward method (62ms)
      ✓ should add marginal reward when reward token is the same
        as staking token correctly (217ms)
  FestakedWithReward Stake Test Cases
    ✓ shouldn't allow to stake before staking start time
    ✓ shouldn't allow to stake with zero amount
    ✓ shouldn't allow to stake if staking cap is reached (46ms)
    ✓ shouldn't allow to stake after staking end time
    ✓ should stake correctly (117ms)
    ✓ should stake if staking token with fee on transfer
      correctly (124ms)
    ✓ should stake for chosen address correctly (77ms)
    ✓ should stake when staking amount exceed staking cap
      correctly (78ms)
  FestakedWithReward Withdraw Test Cases
    ✓ shouldn't allow to withdraw before withdraw start time
    ✓ shouldn't allow to withdraw with zero amount
    ✓ shouldn't allow to withdraw from zero address
    ✓ shouldn't allow to withdraw if withdrawal amount is
      greater than user staked amount
    ✓ should perform early withdrawal if early withdraw reward
      equal to the zero correctly (177ms)
    ✓ should perform early withdrawal with early withdraw
      reward correctly (200ms)
    ✓ should perform withdrawal after close time correctly (170ms)

  39 passing (3s)
```

We are delighted to have a chance to work with the Ferrum Network team and contribute to your company's success by reviewing and certifying the security of your smart contracts.

The statements made in this document should be interpreted neither as investment or legal advice, nor should its authors be held accountable for decisions made based on this document.

Vidma is a security audit company helping crypto companies ensure their code and products operate safely and as intended, enabling founders to sleep soundly at night. We specialize in auditing DeFi protocols, layer one protocols, and marketplace solutions. Our team consists of experienced and internationally trained specialists. Our company is based in Ukraine, known for its strong engineering, cryptography, and cybersecurity culture.

Website: **vidma.io**
Email: **security@vidma.io**

VIDMA