VIDMA

FERRUM NETWORK

SMART CONTRACT AUDIT

Project: Ferrum Network
Date: November 1st, 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 vesting smart contract that can distribute any ERC20 tokens to a limited amount of addresses. The contract implements the AccesControl functionality with two roles, Default Admin and Vester. The Default Admin is actually an owner of the contract that has the ability to grant roles and set a signer, which is used to require that the creation of vesting pools will be made only through the Ferrum platform. The Vestor is allowed to create vesting pools, either Cliff or non-Cliff type.

The creation of the vesting pool and allocations of tokens itself happened in one function. It caused high gas usage and led to the limit of participants for that pool. The max amount of participants depends on the used network and gas price.

The auditor team found a bunch of issues, most of which did not significantly impact the contract's ability to operate. After a second review there are a few unresolved issues that can affect the contract's ability to operate. The most important is a critical issue "The Lack of check for compliance of arrays". Since the contract does not allow edit allocations after the creation, it can lead to the incorrect distribution of tokens, between participants, eg. some participants will have the ability to claim more tokens that had to be allocated for them, and others - less. This issue can be handled by a specific check on FrontEnd but otherwise, it will be possible to reproduce.

After the third review, Vidma audit team confirm that all known issues are resolved and the contracts are secure and operational. The changes are reflected in this version of the report accordingly.

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

Severity of the issue	Total found	Resolved	Not relevant	Unresolved
Critical	2 issues	1 issue	1 issue	0 issues
High	5 issues	4 issues	1 issue	0 issues
Medium	6 issues	5 issues	1 issue	0 issues
Low	27 issues	25 issues	2 issues	0 issues
Informational	14 issues	10 issues	4 issues	0 issues
Total	**54 issues**	**45 issues**	**9 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:

To set the codebase quality mark, 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 evaluation codebase quality 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.

Codebase quality: 88.30

Evaluating the **initial commit** and the **last commit with the fixes**, Vidma audit team set the following **codebase quality** mark.

Score

Based on the **overall result of the audit** and the state of the final reviewed commit, 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 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 audit was conducted from September 16, 2022 to September 23, 2022. The outcome is disclosed in this document.

The review of the fixes was conducted from October 3, 2022 to October 6, 2022.
The second review of fixes was done on October 13, 2022.
The third review of the fixes was done on October 24, 2022.
The final review of the latest commit and release tag was conducted on November 1, 2022.

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

- ironVest.sol (renamed to IronVest.sol);
- Library.sol (unused after fixes).

The source code was taken from the following **source**:
https://github.com/ferrumnet/linear-release-engine/releases/tag/v.0.5.0

Initial commit submitted for the audit (Merge branch 'release/0.5.0'):
https://github.com/ferrumnet/linear-release-engine/commit/bb255308053b2f09e403237ea8c1b277496f9dc6

Final Release tag reviewed by Vidma auditors:
https://github.com/ferrumnet/linear-release-engine/releases/tag/v1.0.0

Final commit hash audited:
https://github.com/ferrumnet/linear-release-engine/commit/5825dc3485395f795cbeb8e18731fd63d8a4b11e

As a reference to the contracts logic, business concept, and the expected behavior of the codebase, the Ferrum Network team has provided the following documentation:
https://docs.ferrumnetwork.io/ferrum-network-ecosystem/v/iron-vest/

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 relevant” 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

Lack of check for compliance of arrays

Critical | MC – 01 | Resolved

In the contract VestingHarvestContarct functions *addVesting* and *addCliffVesting* don't require the same length for arrays *_usersAddresses* and *_userAlloc*. It can bring Critical Issues, and wrong token allocations between beneficiaries, in the case when Vester made a mistake preparing data for *addVesting*.

Recommendation:

Consider adding important checks to prevent human mistakes when adding new vesting.

The wrong calculation for value *nonClifVestingTime*

Critical | MC – 02 | Not relevant

The functions *addCliffVesting* of contract VestingHarvestContarct has a local variable *nonClifVestingTime* with wrong calculation (line 158).

```
nonClifVestingTime = SafeMath.add(SafeMath.sub(_vestingTime ,
_cliffVestingTime),_cliffPeriod)
```

Where:

- *SafeMath.sub(_vestingTime , _cliffVestingTime)* – duration of a non cliff period;
- *_cliffPeriod* - start of cliff period.

Based on the variable name it should be the same as *_vestingTime*, since it's used in the function *nonCliffClaimable* for checking the claimable amount.

Recommendation:

Consider using *_vestingTime* instead of *nonClifVestingTime*.

Wrong parameter in event *AddVesting*

High | MH – 01 | Resolved

In the contract VestingHarvestContarct there is an event *AddVesting* which has parameter *releaseRate*, it is supposed to be a value equal *_totalVesting* / *Vesting* duration, but it is always zero since changes in line 83 are not visible after the loop.

Recommendation:

Consider removing such parameters or fixing them, eg. *_totalVesting* / *Vesting* duration.

Absent check of zero address for the signer address

High | MH – 02 | Resolved

In the functions *addVesting* and *addCliffVesting* of contract VestingHarvestContarct absent check of zero address for the signer of the message.

Recommendation:

Consider adding a zero address check for the signer address, it will make your contract safer.

The contract doesn't fit the requirements for upgradeable

High | MH – 03 | Resolved

The contract VestingHarvestContarct doesn't fit the requirements for upgradable contracts.

When working with upgradeable contracts, there are a few minor caveats to keep in mind when writing your Solidity code. It's worth mentioning that these restrictions have their roots in how the Ethereum VM works, and apply to all projects that work with upgradeable contracts.

Recommendation:

Consider applying all of the requirements for upgradable contracts it will prevent implicit issues in your contract.

The contract is vulnerable for signature replay attack

High | MH – 04 | Resolved

The contract VestingHarvestContarct is vulnerable to a signature replay attack. There is no sure what you are signed on BE side, the functions that use signature don't reproduce signed messages and the same signature can be used for multiple transactions. This issue can be reproduced even on different networks with the same signature.

Recommendation:

Consider using the best market standards for signing messages, such as using domain separators, deadlines, and message type hash.

Inappropriate error message

High | TH – 01 | Not relevant

In the *addCliffVesting()* function there is a requirement that checks whether the cliff percentage is not less than 0.1% but in the error message this value should be not less than 0.5%:

```
    require(
        _cliffPercentage10000 >= 10,
        "Percentage:Percentage Should Be More Than  0.5%"
      );
```

It could lead to incorrect calculations by *cliff-*/*nonCliff-* claiming.

Recommendation:

Fix the condition or change the error message.

Re-Audit:

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

Unchecked call return value

Medium | MM – 01 | Resolved

There is an unchecked return value of the token transfer and transferFrom.

Recommendation:

Consider implementing a check of the return value for the transfer and *transferFrom* functions using safe transfer functions.
From OpenZeppelin library SafeERC20 in the contract VestingHarvestContarct.

Lack of check for *_cliffPeriod* in function *addCliffVesting*

Medium | MM – 02 | Resolved

The functions *addCliffVesting* of contract VestingHarvestContarct, absent check *cliffPeriod* > *block.timestamp*, which allows creating of vesting with already distributed tokens.

Recommendation:

Consider adding important checks to prevent human mistakes when adding new vesting.

Unnecessary rewrite to state variable

Medium | MM – 03 | Resolved

In the function *addVesting* of the contract VestingHarvestContarct initializing of a state variable *poolInfo* (line 84) is located in the loop which means that it will be rewritten in each iteration of the loop.

Same for function *addCliffVesting* where state variable *cliffPoolInfo* is also initialized in the loop.

Recommendation:

Consider moving the initialization of it outside the loop, it will bring saving gas during function execution.

Incompleted condition in *cliffClaimable* and *nonCliffClaimable*

Medium | MM – 04 | Resolved

In the functions *cliffClaimable* and *nonCliffClaimable* of contract VestingHarvestContarct defined incompleted condition (line 194 and 223). In case when *block.timestamp* == *cliffVestingTime*, the function will return zero, instead of *remainingToBeClaimableNonCliff*.

Also valid in function claimable when *block.timestamp* == *vestingTime*.

Recommendation:

Consider changing the condition to *less equal* instead of *less*.

Re-Audit:

Not resolved for claimable.

Release rate value depends on the if-else branching

Medium | TM – 01 | Not relevant

There is a releaseRate calculation in the VestingHarvestContarct in *cliffClaimable()* and *nonCliffClaimable* functions (line 205, 234) but it should be calculated before if-else branching because there will be returned zero values before unlocking start and/or full unlock.

Recommendation:

Consider moving releaseRate calculation before if-else.

Re-Audit:

This issue is not relevant anymore since the *releaseRate* was removed in the latest version.

Invalid pool id emitted in *AddVesting* & *CliffAddVesting* events

Medium | TM – 02 | Resolved

In *addVesting()* and *addCliffVesting()* functions vesting pool data stores with pool id “i” and *vestingPoolSize* increases by 1, so after it event will be emitted with pool id “i + 1” which may confuse further usage.

Recommendation:

Move increasing *vestingPoolSize* variable after event emitting.

Unnecessary default initialization

Low | ML – 01 | Resolved

In contracts VestingHarvestContarct, state variable *vestingPoolSize* (line 12) and *cliff* (line 90) are initialized by default type value that can be avoided.

Recommendation:

Consider removing initialization by the default type value in the described cases.

Re-Audit:

Is not resolved for *cliff[]* in *addVesting* function.

Floating pragma ^0.8.12

Low | ML – 02 | Resolved

The current version of solc in contract VestingHarvestContarct and interface Vesting are ^0.8.12 and it is better to lock the pragma to a specific version.

Contracts should be deployed with the same compiler version and flag that they have been tested 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 to a specific version.

Re-Audit:

Pragma was changed to ^0.8.17 but should be 0.8.17.

Unnecessary using of SafeMath

Low | ML – 03 | Resolved

Using SafeMath is useless since from version 8.0.0+ underflow/overflow are checked under the hood.

Recommendation:

Consider using regular mathematical operators instead of library SafeMath, it will save gas and make your code more clear and readable.

Absence of indexing parameters in events

Low | ML – 04 | Resolved

All events from the contract VestingHarvestContarct didn't specify any indexing parameters. The indexed parameters for logged events will allow you to search for these events using the indexed parameters as filters.

Recommendation:

Consider indexing at least addresses of beneficiary and pool index.

Unnecessary check in functions

Low | ML – 05 | Resolved

In function *addCliffVesting* are unnecessary check *_vestingTime* > *_cliffPeriod* (line 165), since in this function we have check *_vestingTime* > *_cliffVestingTime*, and *_cliffVestingTime* > *_cliffPeriod* it will automatically mean that *_vestingTime* > *_cliffPeriod*.

Recommendation:

Consider removing useless requirements it will bring saving gas during function execution.

Unused local variables

Low | ML – 06 | Resolved

In the function *addVesting* of the contract VestingHarvestContarct are unused local variable *releaseRate* (line 83). It is supposed to be used in the initialization of *userInfo* (line 85), but there this value is calculated again.

Recommendation:

Consider removing unused local variables or reusing them if it is needed, it will bring saving gas during function execution.

Unnecessary return parameter in function *claimable()*

Low | ML – 07 | Resolved

In the function *claimable()* of the contract VestingHarvestContarct one of the return parameters is not initialized and always will return zero.

Recommendation:

Consider removing useless parameters, it will bring saving gas during function execution.

Definition of unused local variables

Low | ML – 08 | Resolved

In the function *claim*, *claimCliff* of the contract, VestingHarvestContarct are defined local variables *secDiff* and *releaseRate*, but they are not used anywhere.

Recommendation:

Consider removing unused local variables, it will bring saving gas during function execution.

Unnecessary state changes

Low | ML – 09 | Resolved

Functions *claim*, *claimCliff*, and *claimNonCliff* of the contract VestingHarvestContarct, change the state variables *userInfo*, *UserClifInfo*, *nonCliffClaimable*, by rewriting the whole object instead of changing only some parameters.

Recommendation:

Consider changing only parameters in the state variables, it will bring saving gas during function execution.

Check on zero claimable amount after transferring it to user

Low | ML – 10 | Resolved

The functions *claim* of contract VestingHarvestContarct makes a check on zero claimable amount after transferring it to the user.

Recommendation:

Consider using Checks Effects Interactions patterns for your functions, it will bring your contract more secure and save gas during function execution.

The repeated calculation of parameters

Low | ML – 11 | Resolved

Functions *claim*, *claimCliff*, and *claimNonCliff* of the contract VestingHarvestContarct, have the repeated calculation of value for parameter remaining of events *Claim*, *CliffClaim*, and *NonCliffClaim*:

- *SafeMath.sub(info.allocation,claimed)* – in event *Claim*;
- *SafeMath.sub(info.cliffAlloc,claimed)* – in event *CliffClaim*;
- *SafeMath.sub(info.nonCliffAlloc,claimed)* – in event *NonCliffClaim*.

Functions *addCliffVesting* of the contract VestingHarvestContarct 6 times calculate a value for *cliffAlloc* and 2 times for *remainingToBeClaimableNonCliff*:

- *SafeMath.div((SafeMath.mul(_userAlloc[i],_cliffPercentage)),100)* – as *cliffAlloc*;
- *SafeMath.sub(_userAlloc[i],SafeMath.div((SafeMath.mul(_userAlloc[i],_cliffPercentage)),100))* – as *remainingToBeClaimableNonCliff*;
- *SafeMath.add(SafeMath.sub(_vestingTime , _cliffVestingTime)* – as *nonClifVestingTime*;

Functions *cliffClaimable* of the contract VestingHarvestContarct 2 times calculate a value for *secondDiff*:

- *SafeMath.sub(block.timestamp , info.cliffLastWithdrawl)* – as *secondDiff.*

Functions *nonCliffClaimable* of the contract VestingHarvestContarct 2 times calculate a value for *secondDiff*:

- *SafeMath.sub(block.timestamp , info.nonCliffLastWithdrawl)* – as *secondDiff.*

Recommendation:

Consider creating local variables to not repeat calculations of repeated values, it will bring saving gas during function execution.

Visibility of function

Low | ML – 12 | Resolved

The functions *addVesting*, *claim*, *addCliffVesting*, *claimCliff*, and *claimNonCliff* of contract VestingHarvestContarct can be marked as external, it will save gas during function execution.

Recommendation:

Consider changing visibility to external if possible, it will bring saving gas during function execution.

Unnecessary condition check in *cliffClaimable* and *nonCliffClaimable*

Low | ML – 13 | Resolved

In the functions cliffClaimable and nonCliffClaimable of contract VestingHarvestContarct defined unnecessary condition(line 198 and 227). In case when condition `cliffVestingTime < block.timestamp` return false it automatically means that cliffVestingTime > block.timestamp.

Recommendation:

Consider deleting unnecessary conditions, it will save gas during function execution.

Local variable shadowing

Low | ML – 14 | Resolved

In the function *VerifyMessage* of contract VestingHarvestContarct there is shadowing of the state variable signer.

Recommendation:

Consider renaming the local variables that shadow another component.

Unnecessary import of contract draft-EIP712.sol

Low | ML – 15 | Resolved

In the file ironVest.sol imported draft-EIP712.sol from the openzeppelin library but it isn't used in the contract VestingHarvestContarct .

Recommendation:

Consider deleting, unused imports, it will save gas during deployment.

Lack of emergency withdraw

Low | ML – 16 | Resolved

Usually, a lot of people accidentally send tokens to a contract, and if the contract doesn't have the functionality to withdraw them they are locked on the contract forever.

Recommendation:

Consider adding functionality for emergency withdrawal of not allocated tokens from the contract, it will allow withdrawing tokens that were sent to the contract by mistake.

Re-Audit:

Resolved but owner has possibility to withdraw all tokens that could be allocated to users. Maybe that funds should be limited to withdraw?

Unused internal function

Low | TL – 01 | Resolved

There is an unused internal function *poolInfoArrays* in the contract VestingHarvestContarct (line 104). Since internal functions can only be used internally or by derived contracts, there is no need for this function.

Recommendation:

Consider removing unused functionality, it will save gas during deployment.

Gas tracking in *addVesting()* / *addCliffVesting()*

Low | TL – 02 | Not relevant

Using of multiple for loops in contract function adding vestings: it has the danger of running into 'out of gas' errors if they are not kept under control.

Recommendation:

Consider adding a *gasleft()* tracking that if it returns true it will break the execution so the 'out of gas' error message will be avoided.

Not valid requirement in *claim()* function

Low | TL – 03 | Resolved

There is an unnecessary checking in the VestingHarvestContarct in *claim()* function (line 104). *startDate* value is equal to timestamp value when vesting was created, a user will claim this vesting this requirement will be never reverted because the current timestamp will be always greater than the start date.

Recommendation:

Consider removing this requirement.

Calculation for secondDiff in *claimable()* function is missed

Low | TL – 04 | Not relevant

Recommendation:

Add a calculation that will emit a value of how much time has passed between user withdrawals.

Re-Audit:

This issue is not relevant anymore since *secondDiff* was removed in latest version.

Not initialized returned value

Low | TL – 05 | Resolved

In the *poolInformation()* view function there is *isCliff* returned value but it is not initialized. It always returns the default false value.

Recommendation:

Initialize *isCliff* variable before if-else branching and use it like condition instead of *cliff[_poolId]* repeat:

```
isCliff = cliff[_poolId];

   if(isCliff) {...}
   else {...}
```

Re-Audit:

Resolved but added declaration that shadows existing variable. Remove `bool` type at line 603.

Not valid initializer requirement

Low | TL – 06 | Resolved

In the *initialize()* function there is requirement that checks whether the contract has not been initialized before (!initialized) but it was already checked in the initializer modifier.

Recommendation:

Remove useless requirement. Also *initialized* = *true*; should be removed from the end of the constructor body.

Rewriting the default values

Low | TL – 07 | Resolved

In the *claimable()*, *cliffClaimable()* and *nonCliffClaimable()* functions there are the last branches that rewriting variables values from zero to zero:

```
claimable(): else { claimable = 0; }
cliffClaimable(): else cliffClaimable = 0;
nonCliffClaimable(): else nonCliffClaimable = 0;
```

Recommendation:

Remove these branches to the gas economy, zero values were setted by default during variable declarations.

Unnecessary check in functions

Low | TL – 08 | Resolved

In function *addCliffVesting* are unnecessary check *_vestingEndTime* > *block.timestamp* (line 361), since in this function we have *checks block.timestamp* < *_cliffPeriodEndTime* < *_cliffVestingEndTime* < *_vestingEndTime* it will automatically mean that *_vestingEndTime* > *block.timestamp*.

Recommendation:

Consider removing useless requirements it will bring saving gas during function execution.

Visibility optimization

Low | TL – 09 | Resolved

Visibility of the *initialize()* and *setSigner()* functions could be optimized from public to external. It has no calls inside the contract code.

Recommendation:

Consider changing visibility of described functions.

Re-Audit:

Is not resolved for *poolInformation()*.

State variable visibility is not set

Low | TL – 10 | Resolved

The default visibility for *poolInfo*, *cliffPoolInfo* mappings are internal. Other possible visibility settings are public and private.

Recommendation:

Mark visibility for two mappings. If *poolInfo* should be internal, mark it internal, rename it with underscore to *mapping(uint256 => PoolInfo) internal _poolInfo* and move to the correct position (according to style guide). It is also relevant to private visibility.

If it is a public variable, just adding *public* will suffice.

Unnecessary checking

Low | TL – 11 | Resolved

In *claimable()* function the last condition of if-else branching could be removed because it will never return false value. It will save some gas.

Recommendation:

Remove *poolInfo[_poolId].vestingEndTime >= block.timestamp* checking.

Order of layout

Informational | MI – 01 | Resolved

The layout contract elements in VestingHarvestContarct are not logically grouped.

The contract elements should be grouped and ordered in the following way:

- Pragma statements;
- Import statements;
- Interfaces;
- Libraries;
- Contract.

Inside each contract, library or interface, use the following order:

- Library declarations (using directive);
- Constant variables;
- Type declarations;
- State variables;
- Events;
- Modifiers;
- Functions.

Ordering helps readers to navigate the code and find the elements more quickly.

Recommendation:

Consider changing the order of layout according to solidity documentation: Order of Layout.

Order of functions

Informational | MI – 02 | Resolved

The functions in contract VestingHarvestContarct are not grouped according to their visibility and order.

Functions should be grouped according to their visibility and ordered in the following way:

- constructor;
- receive function (if exists);
- fallback function (if exists);
- external;
- public;
- internal;
- private.

Ordering helps readers navigate across the code, identify which functions they can call and find the constructor and fallback definitions easier.

Recommendation:

Consider changing functions order according to solidity documentation: Order of Functions.

Code is not formatted

Informational | MI – 03 | Resolved

The code of the interface Vesting and the contract VestingHarvestContarct are not formatted, different spaces and tabs, and the chaotically located parts of code make your code confused and hard to navigate.

Recommendation:

Consider formatting your code according to Solidity Code Style it will make your code more clear and readable.

Absence of NatSpec docs

Informational | MI – 04 | Resolved

In the interface Vesting and in the contract VestingHarvestContarct absent any description of the functions and variables.

Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables, and more.

Recommendation:

Consider adding NatSpec documentation to your contracts it will make your code more clear and readable.

Re-Audit:

There were added simple comments, not natspec.

SafeMath is not used as a library for uint256

Informational | MI – 05 | Not relevant

Functions from library SafeMath are not attached to type uint256 through directive using.

Recommendation:

Consider using directive using to attach functions from library SafeMath, it will make your code clear and readable.

Re-Audit:

SafeMath was removed from the new version, so the issue is not relevant anymore.

Typos

Informational | MI – 06 | Resolved

There is a few typos in parameter and variable names.

In interface Vesting:

- struct *UserInfo*: *lastWithdrawl* -> *lastWithdrawal*, line 22;
- struct *UserClifInfo*: *UserClifInfo* -> *UserCliffInfo*, line 42;
- struct *UserClifInfo*: *tokensRelaseTime* -> *tokensReleaseTime*, line 46;
- struct *UserClifInfo*: *cliffRealeaseRatePerSec* -> *cliffReleaseRatePerSec*, line 48;
- struct *UserClifInfo*: *cliffLastWithdrawl* -> *cliffLastWithdrawal*, line 49;
- struct *UserNonClifInfo*: *struct name UserNonClifInfo* -> *UserNonCliffInfo*, line 52;
- struct *UserNonClifInfo*: *tokensRelaseTime* -> *tokensReleaseTime*, line 56;
- struct *UserNonClifInfo*: *nonCliffRealeaseRatePerSec* -> *nonCliffReleaseRatePerSec*, line 58;
- struct *UserNonClifInfo*: *nonCliffLastWithdrawl* -> *nonCliffLastWithdrawal*, line 59;

In the contract VestingHarvestContarct:

- contract name *VestingHarvestContarct* -> *VestingHarvestContract*, line 9;
- state variable *userClifInfo* -> *userCliffInfo*, line 35;
- state variable *userNonClifInfo* -> *userNonCliffInfo*, line 36;
- state variable *totalvesting* -> *totalVesting*, line 84;
- local variable *nonClifVestingTime* -> *nonCliffVestingTime*, line 200;

Recommendation:

Consider fixing all these typos in the contract parameters and variables.

Wrong function naming convention

Informational | MI – 07 | Resolved

According to Solidity Style Guide, internal functions should begging from the underscore. In contract VestingHarvestContarct there is an internal function *pastpoolInfoArrays* which didn't fit the naming convention.

Recommendation:

Consider following the Solidity Style Guide, and naming conventions.

Wrong variable naming convention

Informational | MI – 08 | Not relevant

According to Solidity Style Guide, state and local variables should be in mixedCase style. In contract VestingHarvestContarct there is a local variable *Info* (line 100) in function *poolInfoArrays* which didn't fit naming convention (initial character in uppercase).

Recommendation:

Consider following the Solidity Style Guide, and naming conventions.

Re-Audit:

This issue is not relevant anymore since *poolInfoArrays()* function was removed.

Variable *vestingPoolSize* is used as a unique index for both types of vesting

Informational | MI – 09 | Not relevant

Since *vestingPoolSize* is used as incrementing key for both types of vesting there will never exist vesting with the same index.

Eg. Two times add *Vesting* and one time add *CliffVesting*, *poolInfo[0]* - first *Vesting*, *poolInfo[1]* - second *Vesting*, *cliffPoolInfo[0]* - didn't exist, *cliffPoolInfo[2]* - first *CliffVesting*.

Recommendation:

Make sure it is expected behavior.

Absence of precision for *cliffPercentage*

Informational | MI – 10 | Resolved

Since in the contract VestingHarvestContarct precision for *cliffPercentage* is not defined, the minimal cliff percentage will be 1%.

Recommendation:

Consider adding precision in case you need a minimal cliff percentage of less than 1%.

Re-Audit:

This issue is resolved. However, the fix of it caused the new issue: % should be more than 0.5% but we have expression % > 0.1%. This issue is added to this report.

Confused naming for the functions and variables

Informational | MI – 11 | Not relevant

In the contract VestingHarvestContarct there are functions and variables that realize the functionality of additional distributing, a kind of "boost" for the main allocation, but most of them consist of *cliff* in the name. It makes your code confused since in classic Vesting cliff means that in that period tokens are not distributed, it kind of a pause.

Recommendation:

Consider replacing *cliff* with the word "boost" etc., for this additional distribution period.

Unused variable

Informational | TI – 01 | Resolved

The prefix variable is unused in the *messageHash()* internal view function.

Recommendation:

Remove unused variable.

The license is not visible to the compiler

Informational | TI – 02 | Resolved

The MIT License in the start of code is invisible for solidity compiled. It is ignored and marked like ‘No license’.

Recommendation:

Change

```
/// SPDX-License-Identifier : MIT
```

line to

```
// SPDX-License-Identifier: MIT
```

Error message styles

Informational | TI – 03 | Resolved

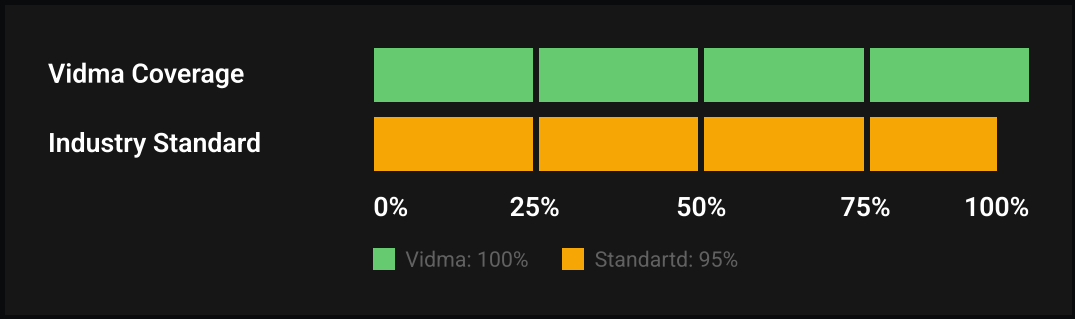
In *addCliffVesting()* function there is a requirement that checks cliff percentage and has the following error message: "Percentage :Percentage Should Be less Than 50%" that style differs from styles in other error messages.

Recommendation:

Add space after colon and remove extra space before percentage value: "Percentage : Percentage Should Be less Than 50%".

TEST RESULTS

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



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 the industry standards.

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	100.00	100.00	100.00
IronVest.sol	100.00	100.00	100.00	100.00
All Files	**100.00**	**100.00**	**100.00**	**100.00**

Test Results

```
Contract: IronVest
  Deployment / Initialization
    ✓ check name
    ✓ check admin role
    ✓ check vester rolefun
    ✓ check signer
    ✓ should fail is contract is already initialized (43ms)
  Functions
    addVesting
      ✓ should fail if caller is not the vester (58ms)
      ✓ should fail if array lengths of users and amounts are not
        equal (39ms)
      ✓ should fail if vesting time will be less than current
        time (38ms)
      ✓ should fail if signature has incorrect message (83ms)
      ✓ should add vesting correctly (206ms)
    addCliffVesting
      ✓ should fail if caller is not the vester
      ✓ should fail if vesting time will be less than current time
      ✓ should fail if vesting time will be less than cliff period
      ✓ should fail if vesting time will be less than cliff period
```

```
    ✓ should fail if vesting time will be less than cliff
      vesting time
    ✓ should fail if cliff period time will be less than
      current time
    ✓ should fail if cliff vesting time will be less than
      cliff period
    ✓ should fail if invalid message was provided (51ms)
    ✓ should fail if cliff percentage will be greater
      than 50% (41ms)
    ✓ should fail if array lengths of users and amounts are
      not equal
    ✓ should add cliff vesting correctly (229ms)
  claimable
    ✓ should fail if user has no allocation in current pool
    ✓ should get base claim amount by user
    ✓ should calculate claim amount (44ms)
    ✓ should get full claim amount
  cliffClaimable
    ✓ should fail if user has no allocation in current pool
    ✓ should get base claim amount
    ✓ should calculate claim amount & release rate by user (38ms)
    ✓ should get full claim amount & release rate by user
  nonCliffClaimable
    ✓ should fail if user has no allocation in current pool
    ✓ should get base claim amount
    ✓ should calculate claim amount
    ✓ should get full claim amount
  claim
    ✓ should claim base amount (55ms)
    ✓ should claim calculated amount correctly (229ms)
    ✓ should get full claim amount & release rate by user (64ms)
  claimCliff
    ✓ should fail if current time is less than cliff period
    ✓ should claim base amount correctly (63ms)
    ✓ should claim calculated amount correctly (137ms)
    ✓ should claim full amount correctly (86ms)
  claimNonCliff
    ✓ should fail if current time is less than cliff period
    ✓ should claim base amount correctly (56ms)
    ✓ should claim calculated amount correctly (140ms)
    ✓ should claim full amount correctly (92ms)
  signatureVerification
```

```
      ✓ should fail if incorrect signature is provided
      ✓ should return not that signer if not that address was
        extracted (50ms)
      ✓ should fail if message was already used
      ✓ should verify message correctly (46ms)
    setSigner
      ✓ should fail if caller is not admin
      ✓ should fail if signer has zero address
      ✓ should set signer correctly
    emergencyWithdraw
      ✓ should fail if caller is not the owner
      ✓ hould withdraw funds correctly (49ms)
  Reentrancy
    ✓ should fail add vesting (222ms)
    ✓ should fail add cliff vesting (276ms)
    ✓ should fail claim (183ms)
    ✓ should fail claim cliff (170ms)
    ✓ should fail claim nonCliff (170ms)
  Test Cases
    ✓ Simple vesting full flow (274ms)
    ✓ Cliff vesting full flow (276ms)
    ✓ Flow when one user should get all vested tokens (244ms)
    ✓ Case when cliff percentage is zero (193ms)

62 passing (5s)
```

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