VIDMA

VRJAM

SMART CONTRACT AUDIT

Project: VRJAM
Date: November 8th, 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.

VRJAM is a multiplayer social gaming and live events platform built to empower global brands and premium content creators to create immersive ‘real-time’ consumer experiences inside virtual worlds.

The audited scope included the VestingWallet contract. It is the contract that handles the vesting of native coins and ERC20 tokens for a given beneficiary. The vesting contract configuration is set in the constructor and can’t be changed after. The vesting contract is eligible only for the one beneficiary who is also set in the contract.

Tokens can be transferred to the contract at any time. After the token is transferred to the vesting wallet balance, tokens start to be eligible for the vesting. Any token transferred to this contract will follow the vesting schedule as if they were locked from the beginning based on the start date set in the contract. Tokens are released linearly according to the number of seconds passed from the start date of the vesting.

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

Severity of the issue	Total found	Resolved	Not relevant	Unresolved
Critical	0 issues	0 issues	0 issues	0 issues
High	0 issues	0 issues	0 issues	0 issues
Medium	0 issues	0 issues	0 issues	0 issues
Low	3 issues	0 issues	1 issue	2 issues
Informational	3 issues	0 issues	1 issue	2 issues
Total	**6 issues**	0 issues	**2 issues**	**4 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. The founds issues are still left with unresolved status but they are not bearing any operational or security risks. 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: 98.80

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

VRJAM is a metaverse platform that empowers rightsholders to create and share digital content and virtual events with global audiences.

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 November 4, 2022 to November 8, 2022. The outcome is disclosed in this document.

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

- VestingWallet.

The source code was taken from the following **sources**:
https://github.com/vrjlive/VRJAM-Vesting

Initial commit submitted for the audit:
b6f2fbd8725eaf7ee87c2ac18bcfe8c9678f5f98

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 VRJAM 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

Inappropriate functions visibility

Low | ML – 01 | Unresolved

There are some functions that are never called in the contract itself with visibility ‘public’ so they can be marked as ‘external’.

Recommendation:

Consider changing visibility from public to external to safe gas usage on calling for functions *release()* and *release(address)*.

Cached storage value

Low | ML – 02 | Unresolved

In the function *_vestingSchedule()* there are 3 storage calls for the *_start* variable value and 2 for the *_duration* variable. Consider avoiding useless storage calls to decrease gas usage on the function call and on the deployment.

Recommendation:

Consider caching the result of *start()* and *duration()* and use a local variable for calculation:

```
uint256 start_ = start();
uint256 duration_ = duration();
```

Optimization by ternary operator

Low | TL – 01 | Not relevant

The *_vestingSchedule()* function can be optimized to return result of ternary operator that will avoid using of if-else branching:

```
if (timestamp < start()) {
return 0;
} else if (timestamp > start() + duration()) {
return totalAllocation;
} else {
return (totalAllocation * (timestamp - start())) / duration();
}
```

to:

```
return
timestamp < start() ? 0 : timestamp > start() + duration()
? totalAllocation
: (totalAllocation * (timestamp - start())) / duration();
```

Recommendation:

Consider using ternary operators.

Useless import

Informational | MI – 01 | Not relevant

VestingWaleet inherits the Context contract but its functionality is never used.

Recommendation:

Consider avoiding import and Inheritness of unused functionality to decrease contract bytecode.

Order of Layout

Informational | MI – 02 | Unresolved

The layout contract elements in the VestingWallet contract are not logically grouped.

Use the following order:

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

Recommendation:

Consider changing the layout order according to the solidity style guide documentation, that is switching events and state variables order.

Lack of NatSpec comments

Informational | MI – 03 | Unresolved

The VestingWallet is partly covered by the NatSpec comments but there is some code without required comments. For example, events and state variables. Also, tag @param for explanation function parameters is missed for each function.

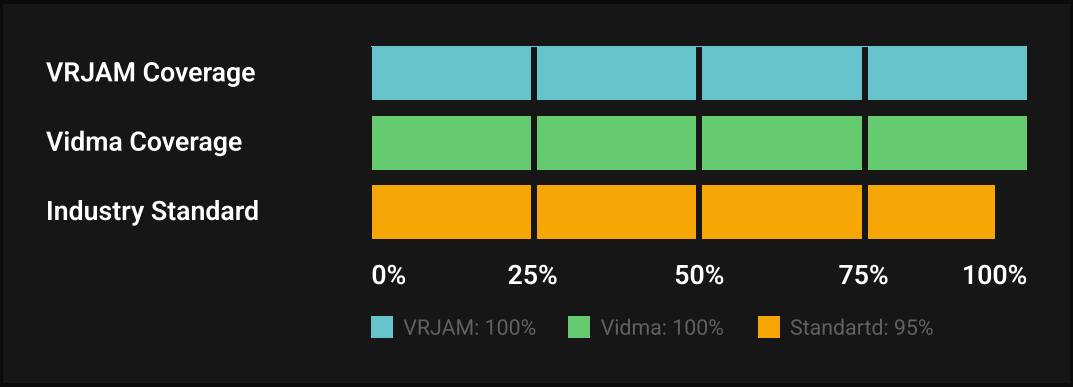
Recommendation:

Consider adding full NatSpec documentation.

TEST RESULTS

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

In this section, we provide both tests written by VRJAM 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 VRJAM repository. We write totally separate tests with code coverage of a minimum of 95% to meet industry standards.

Tests written by VRJAM

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
finance\	100.00	100.00	100.00	100.00
VestingWallet.sol	100.00	100.00	100.00	100.00
All Files	**100.00**	**100.00**	**100.00**	**100.00**

Test Results

```
Contract: VestingWallet
  ✓ rejects zero address for beneficiary (197ms)
  ✓ check vesting contract
  vesting schedule
    Eth vesting
      ✓ check vesting schedule (1278ms)
      ✓ execute vesting schedule (854ms)
    ERC20 vesting
      ✓ check vesting schedule (1243ms)
      ✓ execute vesting schedule (1578ms)
    ERC20 vesting [Seed Round]
      ✓ check vesting schedule (506ms)
      ✓ execute vesting schedule (679ms)
    ERC20 vesting [Public Sale]
      ✓ check vesting schedule (151ms)
      ✓ execute vesting schedule (230ms)

10 passing (8s)
```

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
finance\	100.00	100.00	100.00	100.00
VestingWallet.sol	100.00	100.00	100.00	100.00
All Files	**100.00**	**100.00**	**100.00**	**100.00**

Test Results

```
Contract: VestingWallet
  Deployment / Initialization
    ✓ should revert if strategy beneficiary address is zero address
    ✓ check beneficiary address
    ✓ check start timestamp
    ✓ check vesting duration
  Functions
    releasable (ETH)
      ✓ should return zero if contract balance is zero
      ✓ should return releasable ether amount correctly (57ms)
      ✓ should return releasable ether amount correctly after
        release (78ms)
    releasable (tokens)
      ✓ should return zero if contract balance is zero
      ✓ should return releasable tokens amount correctly (58ms)
      ✓ should return releasable ether tokens correctly after
        release (77ms)
    release (ETH)
      ✓ should emit event with zero amount argument if contract
        balance is zero
      ✓ should release ether correctly (74ms)
      ✓ should release ether twice correctly (84ms)
```

```
    release (tokens)
      ✓ should emit event with zero tokens amount argument if
        contract balance is zero
      ✓ should release tokens correctly (64ms)
      ✓ should release tokens twice correctly (98ms)
    released (ETH)
      ✓ should return released ether correctly (87ms)
    released (tokens)
      ✓ should return released tokens correctly (84ms)
    vestedAmount (ETH)
      ✓ should return zero if vesting is not started yet (61ms)
      ✓ should return zero if contract balance is zero
      ✓ should calculate vested ether correctly (45ms)
      ✓ should return full vested amount correctly
    vestedAmount (tokens)
      ✓ should return zero if vesting is not started yet (67ms)
      ✓ should return zero if contract balance is zero
      ✓ should calculate vested tokens correctly
      ✓ should return full vested amount correctly
  Reentrancy Test
    ✓ should fail reentrancy using delegate call (174ms)
  Test Cases
    ✓ release ether & tokens without vesting (155ms)
    ✓ release more ether & tokens without vesting by increasing
      contract balance (219ms)
    ✓ multiple vesting (few tokens + ether) (394ms)

30 passing (3s)
```

We are delighted to have a chance to work with the VRJAM 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