DEFIMOON

be secure

# Smart Contract Audit Report

February, 2023

LEDGERFI

An Edge to Humanity

DEFIMOON PROJECT

Audit and
Development

CONTACTS

https://defimoon.org
audit@defimoon.org
defimoon_org
defimoonorg
defimoon
defimoonorg

DEFIMOON

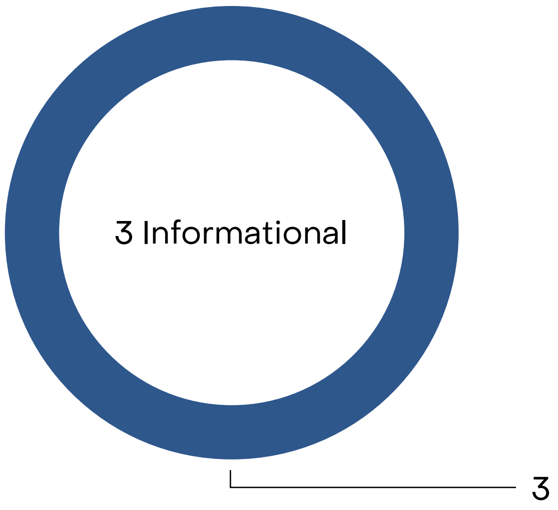
be secure

February 13th 2023

This reaudit report was prepared by Defimoon for LedgerFi

## Audit information

| | |
|---|---|
| Description | LedgerFi Token & Vesting contracts |
| Audited files | Project repo (Token, Vesting) |
| Timeline | 13th February 2023 |
| Audited by | Daniil Rashin |
| Approved by | Artur Makhnach, Kirill Minyaev |
| Languages | Solidity |
| Methods | Architecture Review, Unit Testing, Functional Testing, Manual Review |
| Specification | Repo's README file |
| Docs quality | N/A |
| Source code | Github commit 68b0a76 |
| Network | Not specified |
| Status | Passed |



| | | |
|---|---|---|
| 🟥 | High Risk | A fatal vulnerability that can cause the loss of all Tokens / Funds. |
| 🟧 | Medium Risk | A vulnerability that can cause the loss of some Tokens / Funds. |
| 🟩 | Low Risk | A vulnerability which can cause the loss of protocol functionality. |
| 🟦 | Informational | Non-security issues such as functionality, style, and convention. |

## Disclaimer

This audit is not financial, investment, or any other kind of advice and could be used for informational purposes only. This report is not a substitute for doing your own research and due diligence should always be paid in full to any project. Defimoon is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Defimoon has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Defimoon is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. Defimoon has no connection to the project other than the conduction of this audit and has no obligations other than to publish an objective report. Defimoon will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Defimoon assumes that the provided information and materials were not altered, suppressed, or misleading. This report is published by Defimoon, and Defimoon has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Defimoon. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

## Audit Information

Defimoon utilizes both manual and automated auditing approach to cover the most ground possible. We begin with generic static analysis automated tools to quickly assess the overall state of the contract. We then move to a comprehensive manual code analysis, which enables us to find security flaws that automated tools would miss. Finally, we conduct an extensive unit testing to make sure contract behaves as expected under stress conditions.

In our decision making process we rely on finding located via the manual code inspection and testing. If an automated tool raises a possible vulnerability, we always investigate it further manually to make a final verdict. All our tests are run in a special test environment which matches the "real world" situations and we utilize exact copies of the published or provided contracts.

While conducting the audit, the Defimoon security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Defimoon assesses the risks and assigns a risk level to each section together with an explanatory comment.

# Token Audit overview

## Related findings:

- **DFM-1 — Resolved**

Common parameter amount is treated like an **integer**, which is multiplied by 10**18 each time it occurs, though its type is uint256, it would be impossible to approach its maximum value.
More gas-effective transparent way is to pass these values as **wei**, so the multiplication is done off-chain. Thankfully, there are lots of tools, like **ethers.utils.parseEther**.

- **DFM-2 — Resolved**

A typo in both Token/ERC20Burnable.sol and Token/LedgerFiToken.sol does not break any logic, but may cost you a couple of minutes debugging when this code would be reused.

## Context.sol
**No major issues were found.**

OpenZeppelin-ish implementation.

## ERC20.sol
**No major issues were found.**

OpenZeppelin-ish implementation.

## ERC20Burnable.sol
**No major issues were found.**

OpenZeppelin-ish implementation.

## IERC20.sol
**No major issues were found.**

Standard interface for any token of that type.

## LedgerFiToken.sol
**No major issues were found.**

Simple token contract with slight additions.

## LedgerFiTokenStorage.sol
**No major issues were found.**

Storage contract with immutable variables.

## Ownable.sol
**No major issues were found.**

OpenZeppelin-ish implementation.

# Vesting Audit overview

## Related findings:

- **DFM-3 — Resolved**
  function getCurrentTime() is not really needed as there is already block.timestamp for such needs, also precious gas would be spent on moving calldata back and forth just to return block.timestamp in the end.

- **DFM-4 — Resolved**
  function IERC20address(IERC20 token) does not have any input check, as well as the rest of the contract where this variable is used, so null-value in this case significantly breaks contract functionality.

- **DFM-5 — Resolved**
  function assignToken(address teamMemberAddress, uint256 amount) also needs more input validation, neither null-address nor duplications are appropriate values for membersAddress array.
  Looking at this line:
  teamMember[teamMemberAddress].totalTokensAssigned += amount * (10**18);
  I suppose it is allowed to assign tokens to a member more than once, so it would mess the data representation in the array.

- **DFM-6 — Acknowledged**
  With Ownable.sol already used by other contracts, VestingTeamStorage.sol has its own variant of Ownable features.

- **DFM-7 — Resolved**
  Require statement string typos. Does not affect contract functionality, but rather often affects user-experience.

- **DFM-8 — Acknowledged**

Project strongly relies on the library contracts in the same directory as core contracts. While improving developer experience by allowing to easily check imported sources, it dramatically affects maintainability. Installing OpenZeppelin contracts via yarn/npm would eliminate potential threats caused by replicating popular solutions. With help of well-known frameworks it is much easier to avoid most common threats.
In one particular place there is assembly code which does not really differ from the OZ version, but uses a 0x40 slot to store calldata and returndata.
It seems valid, but overcomplicated.
Please, consider using original open-source solutions when it is possible.

- **DFM-9**

Usage of **for loops** in view functions is acceptable, but not with state changing transactions. While fixing the {DFM-5}, there was added function memberExists(address member) private view returns (bool) iterating over the membersAddress array to find equal element. When called from assignToken opcodes of a view function are executed and gas spent. There are several ways to solve this issue provided in the **Findings** section.

## Address.sol

**No major issues were found.**

OpenZeppelin-ish implementation.

## Context.sol

**No major issues were found.**

OpenZeppelin-ish implementation.

## IERC20.sol

**No major issues were found.**

Standard interface for any token of that type.

## IERC20Permit.sol

**No major issues were found.**

OpenZeppelin implementation.

## Ownable.sol

**No major issues were found.**

OpenZeppelin-ish implementation.

## SafeERC20.sol

**No major issues were found.**

OpenZeppelin-ish implementation.

## VestingMaster.sol

**No major issues were found.**

Most issues refer to this contract either way, severity differs, but main project logic is consistent.
Gas optimisation possible.

## VestingTeam.sol

**No major issues were found.**

Skeleton contract to extend vesting strategies.

## VestingTeamStorage.sol

**No major issues were found.**

Main issue here is duplicating the Ownable.sol functionality {DFM-7}, also {DFM-6} is relevant.

## VestingTeamproxy.sol

**No major issues were found.**

Ownable feature is duplicated by the restricted modifier {DFM-7}. Also the injected assembly is always a potential vulnerability/complicated bug and it is always preferred to avoid it if possible. Also please keep in mind that fallback function in VestingTeamproxy.sol will not allow any value passed through as it does not have payable modifier.

## Summary of findings

According to the standard audit assessment, the audited solidity smart contracts are secure and ready for production, but there is one gas optimization opportunity.

| **ID** | **Description** | **Severity** | **Status** |
|---|---|---|---|
| DFM-1 | Redundant amount math | Medium Risk | Resolved |
| DFM-2 | Tricky typo | Informational | Resolved |
| DFM-3 | Redundant timestamp getter | Informational | Resolved |
| DFM-4 | Token assignment input validation | Medium Risk | Resolved |
| DFM-5 | Assigned token duplicates a member | Low Risk | Resolved |
| DFM-6 | Ownable code duplication | Informational | Acknowledged |
| DFM-7 | Another typos | Informational | Resolved |
| DFM-8 | Open-source adoption | Informational | Acknowledged |
| DFM-9 | For loop gas consumption | Informational | |

## Application security checklist

| | |
|---|---|
| Compiler errors | Passed |
| Possible delays in data delivery | Passed |
| Timestamp dependence | Passed |
| Integer Overflow and Underflow | Passed |
| Race Conditions and Reentrancy | Passed |
| DoS with Revert | Passed |
| DoS with block gas limit | Passed |
| Methods execution permissions | Passed |
| Private user data leaks | Passed |
| Malicious Events Log | Passed |
| Scoping and Declarations | Passed |
| Uninitialized storage pointers | Passed |
| Arithmetic accuracy | Passed |
| Design Logic | Passed |
| Cross-function race conditions | Passed |

## Detailed Audit Information

### Contract Programming

| | |
|---|---|
| Solidity version not specified | Passed |
| Solidity version too old | Passed |
| Integer overflow/underflow | Passed |
| Function input parameters lack of check | Passed |
| Function input parameters check bypass | Passed |
| Function access control lacks management | Passed |
| Critical operation lacks event log | Passed |
| Human/contract checks bypass | Passed |
| Random number generation/use vulnerability | Passed |
| Fallback function misuse | Passed |
| Race condition | Passed |
| Logical vulnerability | Passed |
| Other programming issues | Passed |

### Code Specification

| | |
|---|---|
| Visibility not explicitly declared | Passed |
| Variable storage location not explicitly declared | Passed |
| Use keywords/functions to be deprecated | Passed |
| Other code specification issues | Passed |

### Gas Optimization

| | |
|---|---|
| Assert () misuse | Passed |
| High consumption ‘for/while’ loop | Not Passed |
| High consumption ‘storage’ storage | Passed |
| “Out of Gas” Attack | Passed |
| Public function could be external | Passed |

# Findings

## DFM-1 «Redundant amount math»

**Severity:** Medium Risk

**Status:** Resolved

**Description:**
Amount value is treated as integer in every project's function, though token has decimals. Assuming that every amount we receive is in **wei** we can save some gas and eliminate possible miscalculations.

**Recommendation:**
Get rid of 10**18 when working with such values.

## DFM-2 «Tricky typo»

**Severity:** Informational

**Status:** Resolved

**Description:**
Both Token/ERC20Burnable.sol and Token/LedgerFiToken.sol has function brunDirect which may be missed while reusing ERC20Burnable.sol.

**Recommendation:**

Fix a typo.

## DFM-3 «Redundant timestamp getter»

**Severity:** Informational

**Status:** Resolved

**Description:**
Dedicated function returning block.timestamp increases gas usage and is unneeded.

**Recommendation:**
Use block.timestamp instead.

## DFM-4 «Token assignment input validation»

**Severity:** Medium Risk

**Status:** Resolved

**Description:**
When setting LF token to vesting contract, zero address may be set so that that particular vesting contract won't be able to act correctly.

**Recommendation:**
Add input validation.

## DFM-5 «Assigned token duplicates a member»

**Severity:** Low Risk

**Status:** Resolved

**Description:**
When assigning token to a member via vesting contract user address is added to array each time function is called.

**Recommendation:**
Either prohibit function from assigning to a user several times or make sure user is not already in array.

## DFM-6 «Ownable code duplication»

**Severity:** Informational

**Status:** Acknowledged

**Description:**
VestingTeamStorage.sol and VestingTeamProxy.sol have their own Ownable versions.

**Recommendation:**
It is always better to reuse whats already working, so the desired functionality may be achieved just by inheriting Ownable.sol

## DFM-7 «Another typos»

**Severity:** Informational

**Status:** Resolved

**Description:**
There are typos in require message string, which may affect user-experience.

**Recommendation:**
Fix the typos.

## DFM-8 « Open-source adoption»

**Severity:** Informational

**Status:** Acknowledged

**Description:**
All of the helper contracts (Address, Ownable, ERC20 etc.) are already written and tested by many developers. So when building up the project on top of such contracts you don't need to worry if it works correctly, also you may avoid some threats by simple package manager update.

**Recommendation:**
Use well-known and already tested solutions instead of rewriting the same code again.

## DFM-9 «For loop gas consumption»

**Severity:** Informational

**Description:**
Calling memberExists function from assignToken would increase gas consumption with each member added. It is hard to reach "out of gas" with such function, nevertheless it is much more efficient to check the member with constant complexity.

**Recommendation:**
Basically memberExists can be replaced with mapping(address => bool) with the same name. If removing the member from array is needed, another mapping(address => uint) memberID may be added to store indexes of membersAddress thus removing an item from array may be done with simple assignment of the last member to the one being deleted and members.Address.pop().

## Adherence to Best Practices

1. Use OpenZeppelin solutions.
2. Consider amount values as wei.
3. Reuse the code instead of duplication.

## Methodology

### Manual Code Review

We prefer to work with a transparent process and make our reviews a collaborative effort. The goal of our security audits is to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

### Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, review open issue tickets, and investigate details other than the implementation.

### Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system to make a final decision.

### Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

## Appendix A — Finding Statuses

| | |
|---|---|
| Resolved | Contracts were modified to permanently resolve the finding |
| Mitigated | The finding was resolved by other methods such as revoking contract ownership or updating the code to minimize the effect of the finding |
| Acknowledged | Project team is made aware of the finding |
| Open | The finding was not addressed |