HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: LunaFi Technologies Ltd
Date: April 18, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for LunaFi Technologies Ltd.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC-20 token; Staking
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	www.lunafi.io
Timeline	24.03.2022 - 14.04.2022
Changelog	28.03.2022 - Initial Review 07.04.2022 - Revising 18.04.2022 - Revising

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by LunaFi Technologies Ltd (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

The second review was conducted on April 07th, 2022.

The third review was conducted on April 14th, 2022.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/Luna-Fi/lunafi-smart-contracts
Commit:
6ff9a601e9bc72326b8e89f69daaa3105ed6afb0
Technical Documentation: Yes
JS tests: Yes
Contracts:
./contracts/LFIToken.sol
./contracts/VLFI.sol
./contracts/claimToken.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency ▪ Kill-Switch Mechanism

Executive Summary

The score measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided brief functional requirements and technical documentation. The total Documentation Quality score is **6** out of **10**.

Code quality

The total Code Quality score is **5** out of **10**. The code is partially commented. Unit tests have medium coverage. The contracts have the following coverage: LFIToken.sol 23.26%, VLFI.sol 43.97%, claimToken.sol 75%, vesting.sol 94.19%.

After the second review, the code is found to be well-commented, and unit tests were improved. The total Code Quality score is **7** out of **10**

Architecture quality

The architecture quality score is **10** out of **10**. The project has clear and clean architecture.

Security score

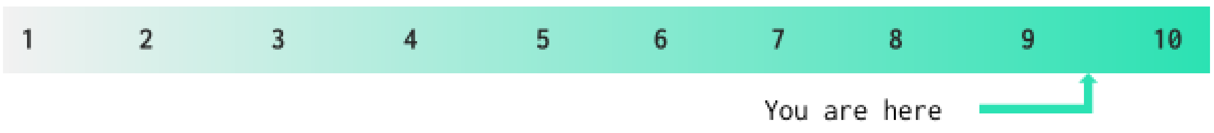
As a result of the audit, security engineers found **4** high, **1** medium, and **3** low severity issues. Security score is **0** out of **10**. All found issues are displayed in the “Issues overview” section.

After the second review, the code contains **1** high and **1** low severity issue. The security score is **5** out of **10**.

After the third review, the code contains **no** issue. The security score is **10** out of **10**.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.3**



Notices

1. The staking contract implementation may be updated by the contract owner. The current audit covers only implementation from the scope.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

1. **Vesting funds are not guaranteed.**

   The vesting contract creates all the vesting schedules by calling `createVestingSchedule` function from the constructor, but the `createVestingSchedule` function logic does not guarantee that the contract has enough funds for vesting payment.

   Contracts: vesting.sol

   Function: createVestingSchedule

   Recommendation: Add a statement to the `createVestingSchedule` function to check if the contract has enough funds. Create a function to add a batch of predefined vestings schedules after the contract deployment.

   Status: Fixed

2. **Staking liquidity funds safety is not guaranteed by the contract logic.**

   The VLFI.sol contract has the `transferToTreasury` function, which allows an account with the manager role to withdraw staking liquidity funds to any account. It is expected to be possible to withdraw less than `maxWithdrawalLiquidity` percent of the `liquidity` value, but the `transferToTreasury` function does not update the `liquidity` value.

   An account with a manager role may call the function till all the funds are not withdrawn.

   Contracts: VLFI.sol

   Function: transferToTreasury

   Recommendation: Update `liquidity` value inside `transferToTreasury` function.

   Status: Fixed

3. **Admin may burn users' funds.**

   The contracts have functions that allow the admin to burn tokens from any account, not only from the admin account.

   Contracts: claimToken.sol

   Function: burn

   Recommendation: Update the function's logic to allow burning funds only from the admin account or burn tokens only when a user provides approval for this.

Status: Fixed

4. **Admin may mint an unlimited amount of tokens.**

   The contract has a function that allows the admin to mint any amount of tokens to any account.

   Contracts: claimToken.sol

   Function: mint

   Recommendation: Update the function logic to allow minting tokens before `totalSupply` limit or specify unlimited mint behavior in public documentation.

   Status: Fixed

🟩🟩 Medium

1. **Redundant function argument.**

   The function has two arguments. `value` argument is the value of the permitted amount of tokens that may be spent, and `LFIamount` is the number of tokens that should be staked.

   There is no sense in permitting using more tokens than it should be staked.

   Contracts: VLFI.sol

   Function: permitAndStake

   Recommendation: Remove one of the arguments.

   Status: Fixed

🟦 Low

1. **Redundant use of SafeMath library.**

   SafeMath is not needed starting with Solidity 0.8. The compiler has built-in overflow checking.

   Contracts: LFIToken.sol

   Recommendation: Do not use SafeMath in the contract.

   Status: Fixed

2. **Use of magic number.**

   The vesting contracts interact with the token contract. The tokens' contract address is passed to the constructor, but the local `decimals` variable does not get value from the contract. Accidentally vesting contract may operate with a wrong decimal value, which may lead to the funds lost.

   Contracts: vesting.sol

   Function: constructor

Recommendation: Get `decimals` value from the token contract external function.

Status: Fixed

3. **Unexpected output.**

   According to the vesting contract logic, itis possible to create multiple vesting schedules for one address, but the `getVestingScheduleByAddress` function logic suspects that one address may have only one assigned vesting.

   Contracts: vesting.sol

   Function: getVestingScheduleByAddress

   Recommendation: Update the `getVestingScheduleByAddress` function logic, or add a statement to check to `createVestingSchedule` if the vesting with the same recipient exists.

   Status: Fixed

4. **Redundant functions.**

   The LFIToken has unused internal functions: `_mint`, `_burn`.

   Contracts: LFIToken.sol

   Function: _mint, _burn

   Recommendation: Remove unused functions.

   Status: Fixed

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.