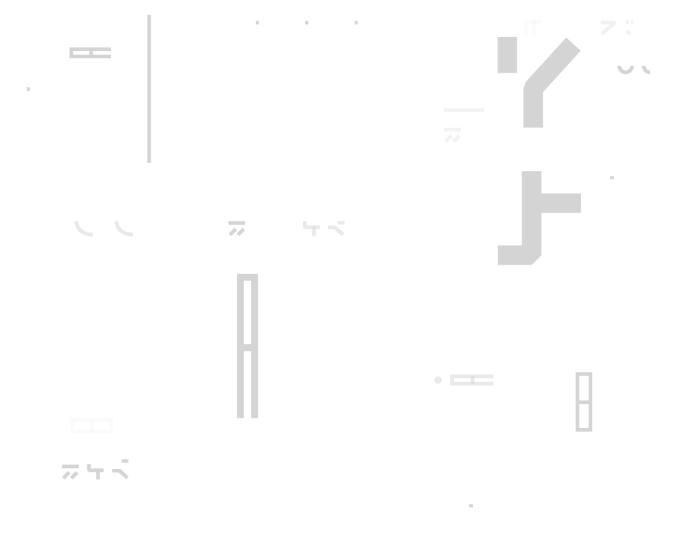


SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT



Customer: Liquidus

Date: December 20th, 2021



This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Liquidus.		
Approved by	Andrew Matiukhin CTO Hacken OU		
Туре	Staking		
Platform	Binance Smart Chain / Solidity		
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review		
Solidity Files	Farm_vesting.sol SingleTokenStake.sol		
md5 Hash	435f54808fb12f9585dd5055c4b0a162 6f0df7cfb6300da6d4237555151f41d5		
Technical Documentation	NO		
JS tests	NO		
Website	liquidus.finance		
Timeline	13 DECEMBER 2021 - 20 DECEMBER 2021		
Changelog	20 DECEMBER 2021 - INITIAL AUDIT		

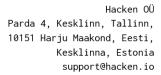




Table of contents

Introduction	4
Scope	4
Executive Summary	5
Severity Definitions	6
Audit overview	7
Conclusion	10
Disclaimers	11



Introduction

Hacken OÜ (Consultant) was contracted by Liquidus (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 contract and its code review conducted between December $13^{\rm th}$, 2021 - December $20^{\rm th}$, 2021.

Scope

The scope of the project is smart contracts in the solidity files:

Files:

Farm_vesting.sol SingleTokenStake.sol

md5 hash:

435f54808fb12f9585dd5055c4b0a162 6f0df7cfb6300da6d4237555151f41d5

Technical Documentation: No

JS tests: No Contracts:

Farm_vesting.sol SingleTokenStake.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
	DoS with (Unexpected) Throw
	DoS with Block Gas Limit
	 Transaction-Ordering Dependence
	Style guide violation
	Costly Loop
	■ ERC20 API violation
	Unchecked external call
	Unchecked math
	Unsafe type inference
	Implicit visibility level
	Deployment Consistency
	Repository Consistency
	■ Data Consistency



Functional review	•	Business Logics Review
	•	Functionality Checks
	•	Access Control & Authorization
	•	Escrow manipulation
	•	Token Supply manipulation
	•	Assets integrity
	•	User Balances manipulation
	•	Data Consistency manipulation
	•	Kill-Switch Mechanism
	1	

Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secured.



Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found 9 low severity issues.



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 can't lead to assets loss or data manipulations.	
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution	



Audit overview

Critical

No critical issues were found.

High

No high severity issues were found.

■ ■ Medium

No medium severity issues were found.

Low

1. State variables that could be declared immutable.

Constant state variables that are initialized in the constructor should be declared immutable to save gas.

Contract: Staking

Variables: reward, lpToken

Recommendation: Add the **immutable** attribute to state variables that never change and are initialized in the constructor.

2. Missing event for changing rewardPerBlock, vestingTime

Changing critical values should be followed by the event emitting for better tracking off-chain.

Contracts: Staking

Functions: setRewardPerBlock, updateVestingTime

Recommendation: Please emit events on the critical values changing.

Duplicated code

Some code, like rewards calculation, pending rewards, rewards debt are duplicated multiple times.

Contracts: Staking

Functions: pendingReward, updatePool, deposit, withdraw, harvest

Recommendation: Please put the calculation code into one function and call it from others when needed.

4. A public function that could be declared external.

public functions that are never called by the contract should be declared external to save gas.

Contracts: Staking



Functions: setRewardPerBlock, deposit, withdraw, harvest, emergencyWithdraw

Recommendation: Use the **external** attribute for functions never called from the contract.

5. State variables that could be declared immutable.

Constant state variables that are initialized in the constructor should be declared immutable to save gas.

Contract: CodiStake

Variables: stakedToken, PRECISION_FACTOR

Recommendation: Add the **immutable** attribute to state variables that never change and are initialized in the constructor.

6. A public function that could be declared external.

public functions that are never called by the contract should be declared external to save gas.

Contracts: CodiStake

Functions: harvest

Recommendation: Use the **external** attribute for functions never called from the contract.

7. Duplicated code

Some code, like rewards calculation, pending rewards, rewards debt are duplicated multiple times.

Contracts: CodiStake

Functions: deposit, pendingReward, harvest, withdraw,

Recommendation: Please put the calculation code into one function and call it from others when needed.

8. Excess require statement

The same "require" statement is placed on lines 972 and 973.

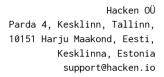
Contracts: CodiStake

Functions: recoverWrongTokens

Recommendation: Please remove excess require statement.

9. Missing event for changing bonusEndBlock, vestingTime

Changing critical values should be followed by the event emitting for better tracking off-chain.





Contracts: CodiStake

Functions: stopReward, updateVestingTime

Recommendation: Please emit events on the critical values changing.



Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found $\bf 9$ low severity issues.



Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to 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 as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. 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 can't guarantee the explicit security of the audited smart contracts.