SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: reBase
Date: 25 Apr, 2023

This report 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 can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for reBase
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token; Vesting
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://rebase.gg
Changelog	25.04.2023 - Initial Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by reBase (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.

Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/rebasegg/vesting-contract-eth
Commit	8f6eaf68f
Whitepaper	https://docs.rebase.gg/whitepaper-1.02
Functional Requirements	https://docs.rebase.gg/whitepaper-1.02/3.-rebase-echopath-layer
Contracts	File: ./contracts/MerkleTokenVesting.sol SHA3: ee4d44392dfc0c76c79c6b448c77e128f44d6e5d6f2e701facf9357d3e6fb4f6 File: ./contracts/Rebase.sol SHA3: 1ceb46f0bff019806791036b80215dac026a592b94e917cfbf07d9f750636afe File: ./contracts/abstract/MerkleDistributor.sol SHA3: 4c71d188aa4712a7d59082c1d4432d378b8abfeecafc00abe096d2ab81ae0e78 File: ./contracts/abstract/TokenVesting.sol SHA3: 2be4d15d0f7d1222ace878bb839fd4557774d86435953d008c5178cfb298c3b3 File: ./contracts/mocks/ERC20Mock.sol SHA3: f663ffeae512c0b7ef26e1c3f7235c9f85ff01a1f1ee4637233c5d4a4fa9bc30

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Executive Summary

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

Documentation quality

The total Documentation Quality score is **9** out of **10**.

- Functional requirements are partially provided.
  - Missing information about vesting schedule and claims.
  - Missing contract-specific use cases.
- Technical description is complete.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- Solidity Style Guide is followed.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and user interactions are covered with tests.
- Negative test cases are covered.
- Interactions by several users are tested.

Security score

As a result of the audit, the code contains **0** issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.9**. The system users should acknowledge all the risks summed up in the risks section of the report.

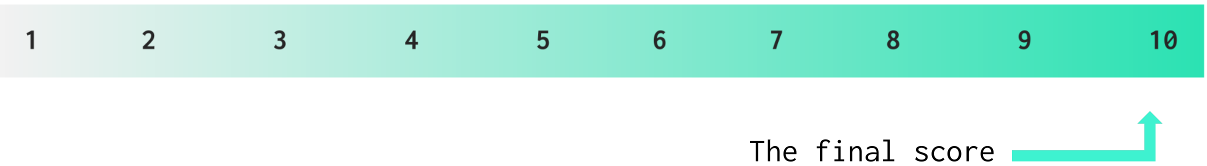

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
25 April 2023	0	0	0	0

Risks

- It is recommended that the ownership of the Vesting contract be renounced once all vesting schedules have been added, in order to mitigate the potential risk of manipulations to previously established schedules.
- For enhanced transparency and user confidence, the Merkle tree should be made publicly accessible, displaying the amount of funds that the contract owner is required to deposit into the contract.

System Overview

The system is composed by a vesting contract, MerkleTokenVesting.sol, and the contract of the ERC20 token being vested, Rebase.sol. MerkleTokenVesting inherits MerkleDistributor.sol and TokenVesting.sol, where some of the vesting functionalities reside.

- Rebase - Simple ERC-20 token that mints all initial supply to a deployer. Additional minting is not allowed.
  It has the following attributes:
  - Name: Rebase
  - Symbol: IRL
  - Decimals: 18
  - Total supply: 500m tokens.
- MerkleTokenVesting — Vesting contract that allows the configuration of periodic vesting with start tokens and cliff time.
- MerkleDistributor - Abstract contract that provides functions for adding Merkle roots, verifying Merkle proofs, and keeping track of claimed tokens using a bit map.
- MerkleTokenVesting - Abstract contract containing vesting functionalities and the external function for the users to claim vested tokens based on the vesting schedule.

Privileged roles

- The owner of MerkleTokenVesting can add new vesting schedules at any time.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed

Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Passed
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟧🟧🟧 High

No high severity issues were found.

🟨🟨 Medium

No medium severity issues were found.

🟦 Low

No low severity issues were found.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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 Consultant cannot guarantee the explicit security of the audited smart contracts.