HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: KAIF DAO Platform
Date: January 06, 2023

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 KAIF DAO Platform
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token; Vesting
Platform	EVM
Language	Solidity
Methodology	Link
Changelog	07.12.2022 - Initial Review 14.12.2022 - Second Review 06.01.2023 - Third Review

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by KAIF DAO Platform (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 is smart contracts in the repository:

Initial review scope

Repository	https://gitlab.com/kaif-cabinet/kaif-smart-contract/
Commit	56ddd932843b754b5d8cdd222ecea3435f11286a
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/Vesting.sol SHA3:078ed60266c2802e8cb9790aa6b0934596d3a33bf7ebe643a9b55984a4ce9eee File: ./contracts/Token.sol SHA3:ac33ad31f47fb65268c9a142c73300b13ea74c9e4d16ccad2c7af5a4da437d4c File: ./contracts/interface/ITokenVesting.sol SHA3:ef39aaa1755821c47f20c4de96d4b330dd46b98e9073ca3f55197cbbf1ede5d1

Second review scope

Repository	https://gitlab.com/kaif-cabinet/kaif-smart-contract/
Commit	9686cadb4c4d3839f81f06ed3325fd58bc804c23
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/Vesting.sol SHA3:190d6b47fb1872408b216237cf13f10677693b1f484f97bc48f63f3f73c03317 File: ./contracts/Token.sol SHA3:d9a9d7190127e6caf3555dce91e0f13b9c2ca4cdc038b24470705079ce428ba4 File: ./contracts/interface/ITokenVesting.sol SHA3:ef39aaa1755821c47f20c4de96d4b330dd46b98e9073ca3f55197cbbf1ede5d1

Third review scope

Repository	https://gitlab.com/kaif-cabinet/kaif-smart-contract/

Commit	5fb5a1780b8cbda1c5c4ce9441d0dc3b679012f9
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/Vesting.sol SHA3:053968881f5d57b27bc3e350203d88a5c50afaf0d162e3ddb490ea594ece7c1d File: ./contracts/Token.sol SHA3:d9a9d7190127e6caf3555dce91e0f13b9c2ca4cdc038b24470705079ce428ba4 File: ./contracts/interface/ITokenVesting.sol SHA3:ef39aaa1755821c47f20c4de96d4b330dd46b98e9073ca3f55197cbbf1ede5d1

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 assets 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 the code quality

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

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 **10** out of **10**.

- Functional description is provided.
- Technical description is provided.

Code quality

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

- The development environment is configured.
- The code follows the official Solidity style guides.

Test coverage

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

- The testing environment is set up.
- Deployment and basic user interactions are covered with tests.
- Positive and negative cases are covered.
- Interactions by several users are tested thoroughly.

Security score

As a result of the audit, the code does not contain 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: **10**.

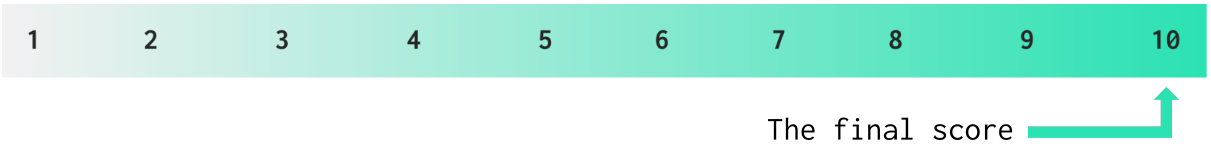

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
7 December 2022	4	0	0	0
14 December 2022	0	0	0	0
06 January 2023	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more 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.	Passed
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.	Passed
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.	Passed
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.	Not Relevant
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.	Passed
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.	Not Relevant

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

System Overview

KAIF DAO Platform is a vesting system with the following contracts:

- *Token* — ERC-20 token that mints all initial supply to a specified address. Additional minting is not allowed. Contains a custom method for setting the start of the vesting TGE (Token Generation Event). It has the following attributes:
  - Name: specified as constructor parameter during deployment
  - Symbol: specified as constructor parameter during deployment
  - Decimals: 18
  - Total supply: 809.710.000 tokens.
- *Vesting* - a vesting contract for managing token generation events, participants and destinations. It is focused on multisig wallets.
- *ITokenVesting* - an interface that exposes the "setStartAt" method from the *Vesting* contract.

Privileged roles

- DEFAULT_ADMIN_ROLE: Contract deployer.
  - Can set a public round vest for custom addresses and amounts.
  - Can set a seed round vest for custom addresses and amounts.
  - Can set a private round one vest for custom addresses and amounts.
  - Can set a private round two vest for custom addresses and amounts.
  - Can set a marketing vest for a custom address and amount.
  - Can set a main team vest for custom addresses, amounts and percentages.
  - Can set a foundation vest for custom addresses and amounts.
  - Can withdraw all withdrawable tokens (token balance not meant to be claimed by any vest).
- MULTISIG_ROLE: EOA wallet that represents the participant of vesting.
  - Can add a vesting schedule for additional users other than the original vesting founders.
- STARTER_ROLE: The Token address.
  - Can set the "startAt" variable to the current block timestamp.

Risks

No potential risks were found.

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

No medium severity issues were found.

Low

1. Unindexed Events

Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

Paths: ./contracts/Vesting.sol : event Claimed(), event VestingCreated(), event BatchVestingCreated()

Recommendation: Use the “indexed” keyword to at least one of the event parameters.

Status: Fixed (Revised commit: 9686cad)

2. Style Guide Violation

The project should follow the official guidelines.

Inside each contract, library or interface, use the following order:

1. Type declarations
2. State variables
3. Events
4. Modifiers
5. Functions

Functions should be grouped according to their visibility and ordered:

1. constructor
2. receive function (if exists)
3. fallback function (if exists)
4. external
5. public
6. internal
7. private

Scientific notation in the form of 2e10 is recommended to aid readability if using literals with too many digits, underscores can be used to separate the digits of a numeric literal as well.

Path: ./contracts/Vesting.sol

Recommendation: Follow the official Solidity guidelines.

Status: Fixed (Revised commit: 9686cad)

3. **Missing Zero Address Validation**

   Address parameters are being used without checking against the possibility of 0x0.

   This can lead to unwanted external calls to 0x0.

   Paths: ./contracts/Vesting.sol ./contracts/Token.sol

   Recommendation: Implement zero address checks.

   Status: Fixed (Revised commit: 9686cad)

4. **Reading State Variables in a Loop**

   Reading a state variable or an attribute of it may be costly, in terms of Gas fees.

   Path: ./contracts/Vesting.sol : setMainTeamVestFor(), setAdditionalTeamVestFor(), claim(), getVestedAmount(), _batchVestFor()

   Recommendation: Save the state variable or its attribute into a local variable and perform updates after the loop.

   Status: Fixed (Revised commit: 9686cad)

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.