QuillAudits

AUDIT REPORT

April , 2025

For

cryptokmbio

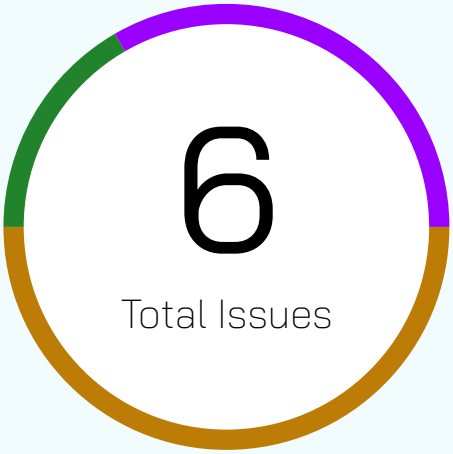
Table of Content

Executive Summary

Project name	KMBio
Project URL	https://kmbio.ai/en
Overview	The 'vesting' project is a smart contract designed to manage token vesting schedules for team members. It allows the contract owner to create vesting schedules for beneficiaries, specifying the amount of tokens and the vesting period. A unique feature is its ability to handle vesting with a cliff period, ensuring tokens are only accessible after a certain time.
Audit Scope	The scope of this Audit was to analyze the KMBio Vesting Smart Contracts for quality, security, and correctness.
Source Code link	Marketing: https://sepolia.basescan.org/address/0x93A57899BF9c7073A8B7994C4FDcFe3a19c40d05#code Development: https://sepolia.basescan.org/address/0xA382BC385FA37F71562f7A9c63516315a80C2204 Team: https://sepolia.basescan.org/address/0x5311aB62269c1Dc3656776146eD5353D9B6ae675#code Advisor: https://sepolia.basescan.org/address/0x14F7F16F97B8dBD8533eB96591531BcC9fD4Cc64#code
Contracts in Scope	Marketing,Development,Teams, Advisor
Language	Solidity
Blockchain	Base

Review 1	13th April 2025 - 18th April 2025
Updated Code Received	18th April 2025
Review 2	22nd April 2025
Fixed In	MarketingVesting - 0x633E7bFc17a30D2b18A9246a0116836Ea5Eca042 DevelopmentVesting - 0xB344C8D76Fbe2577334e51c7392A21199d47a869 AdvisorVesting - 0xb010239045BF47e71143Cf6F3CA991471544d6aC TeamVesting - 0x8321768986b20009d920b0bF6AC4368602F65ae7

Number of Issues per Severity



High	0 (0.00%)
Medium	3 (50.00%)
Low	1 (16.67%)
Informational	2 (33.33%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	0	**3**	**1**	**2**
Acknowledged	0	0	0	0
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] Implicit visibility level.

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools And Platforms Used For Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

Security vulnerabilities identified that must be resolved and are currently unresolved.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.

Medium Severity Issues

No Zero Address Validation in transferOwnership

Resolved

Path

DevelopmentVesting.sol,Vesting.sol,TeamVesting.sol,Advisorvesting.sol

Function

transferOwnership()

Description

The transferOwnership function lacks validation to prevent the owner from being set to the zero address (0x0). If the owner is accidentally set to the zero address, all owner-restricted functions would be permanently inaccessible

```
function transferOwnership(address _newOwner) external {
        require(
            msg.sender == owner, //@note no 0 address check and need 2 step process
for owner change.
            "VestingContract::transferOwnership: Only owner"
        );
        owner = _newOwner;
    }
```

Recommendation

Add a zero address check in the transferOwnership function.

Implement 2 step owner transfer

Resolved

Path

DevelopmentVesting.sol,Vesting.sol,TeamVesting.sol,Advisorvesting.sol

Function

transferOwnership()

Description

The current ownership transfer mechanism is single-step, meaning the owner is immediately changed. If the owner address is incorrectly specified (typo) or the receiving account is compromised or unable to access the address.

Recommendation

Implement a two-step ownership transfer process where the new owner must accept the ownership.

Unrestricted transferBalanceToOwner Function

Resolved

Path

DevelopmentVesting.sol,Vesting.sol,TeamVesting.sol,Advisorvesting.sol

Function

transferBalanceToOwner()

Description

The _transferBalanceToOwner function allows the owner to drain all tokens from the contract, including those that have been allocated to beneficiaries but not yet claimed. This completely undermines the purpose of the vesting contract. In case the owner decides to withdraw all tokens, beneficiaries who were expecting to receive tokens based on the vesting schedule would be unable to claim their rightful allocations.

Recommendation

Implement a time-locked withdrawal mechanism like this can be called after 30 days from the end of the vesting period.

Low Severity Issues

Missing end time validation in constructor

Resolved

Path

DevelopmentVesting.sol,Vesting.sol,TeamVesting.sol,Advisorvesting.sol

Function

constructor()

Description

The constructor should enforce a minimum duration between start and end time. Without this validation, the contract could be deployed with very short or practically 0 vesting periods.

Recommendation

Add validation to ensure end time is sufficiently later than start time.
15 min for DevelopementVesting and 30 days for remaining.

```
require(
    _end >= _start + 15 minutes,
    "VestingContract::constructor: End must be at least 15 minutes after start"
);
```

Informational Severity Issues

Unused Variable in createVestingSchedules.

Resolved

Path

DevelopmentVesting.sol,Vesting.sol,TeamVesting.sol,Advisorvesting.sol

Function

createVestingSchedules()

Description

The result variable in createVestingSchedules function is always set to true but never modified, making it redundant.

Recommendation

return true directly instead of this variable.

Incorrect error messages in Time Periods

Resolved

Path

DevelopmentVesting.sol

Function

_drawDown()

Description

The error message in _drawDown function mentions "30-day period" when the actual implementation uses 15 minutes.

```
// Ensure we're at or past the claim timestamp
        require(
            _getNow() >= claimTimestamp,
            "VestingContract::_drawDown: Cannot claim before the next 30-day
period" //@note comment should be updated with 15 minutes
        );
```

Recommendation

update it to 15 minutes.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of KMBio Contracts. We performed our audit according to the procedure described above.
Issues of medium, low and informational severity was found all the issues were resolved by KMBio team

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

April , 2025

For

cryptokmbio

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com