QuillAudits

AUDIT REPORT

April , 2025

For

[d]kloud

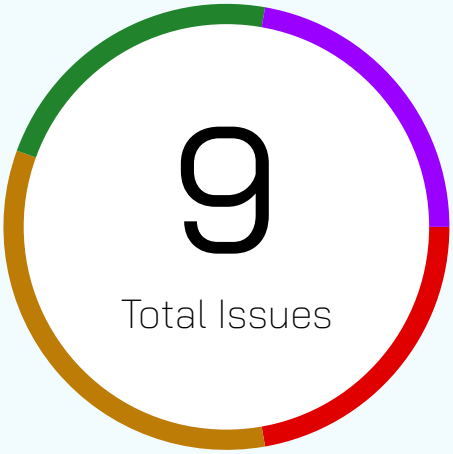
Table of Content

Executive Summary

Project name	dKloud
Project URL	https://www.dkloud.io
Overview	DKTVesting contract is an upgradeable Solidity smart contract designed to manage token vesting schedules for multiple beneficiaries. It allows the contract owner to create vesting schedules, allocate tokens to beneficiaries, and release tokens over time based on predefined rules (e.g., TGE release, cliff periods, and linear vesting).
Audit Scope	The scope of this Audit was to analyze the dKloud Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/dkloudio/DKT-Contract
Contracts in Scope	DKTVesting.sol DKT.sol
Branch	Dev
Commit Hash	2ec20ba1c9a2e016e9e3e9e19f3231e2b0693c7c
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	6th March 2025 - 17th March 2025

Updated Code Received	31st March 2025
Review 2	3rd March 2025 - 4th March 2025
Fixed In	https://github.com/dkloudio/DKT-Contract Branch: dev
Commit Hash	10e6f9147249d5d12da940896a4f7501f5397a94

Number of Issues per Severity



High	2 (22.22%)
Medium	3 (33.33%)
Low	2 (22.22%)
Informational	2 (22.22%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	**2**	**3**	**2**	**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.

High Severity Issues

Anybody can claim for anyone

Resolved

Path

DKTVesting.sol

Function

claim(address beneficiary, uint256 beneficiaryScheduleId)

Description

The claim() function is publicly accessible and allows any caller to execute claims on behalf of any beneficiary. This represents a significant access control vulnerability since there is no validation that the caller is either the beneficiary or an authorized representative.

Recommendation

Implement proper access controls by adding a check that msg.sender == beneficiary

amountInFinalPeriod is ignored in the calculation

Resolved

Path

DKTVesting.sol

Function

calculateTotalAmountEarnedToDate(ScheduleStruct memory _schedule

Description

The contract calculates a special amountInFinalPeriod value in createBeneficiarySchedule() to account for division remainders, but this value is never used when calculating claimable amounts. Instead, the function always uses amountPerPeriod even for the final period.
This bug permanently locks remainder tokens in the contract:

Any amounts not perfectly divisible by the period count will have remainders
These remainder tokens become permanently inaccessible to beneficiaries
Across all beneficiaries, this could represent a significant amount of locked tokens

```
// In createBeneficiarySchedule
amountPerPeriod = amountAfterTge / _schedule.totalVestingPeriods;
amountInFinalPeriod = (amountAfterTge / _schedule.totalVestingPeriods) +
(amountAfterTge % _schedule.totalVestingPeriods);

// In calculateTotalAmountEarnedToDate - the bug
// Calculate the amount earned to date for the current period
totalVestedAmount += _beneficiarySchedule.amountPerPeriod * currentReleasePeriod;
// Never uses amountInFinalPeriod for the final period
```

Recommendation

Modify calculateTotalAmountEarnedToDate() to properly handle the final period:

```
// Check if in the final period
if (currentReleasePeriod == _beneficiarySchedule.totalVestingPeriods) {
      // Use the special final period amount that includes the remainder
      totalVestedAmount += _beneficiarySchedule.amountInFinalPeriod;
} else {
      // Use the regular per-period amount for non-final periods
      totalVestedAmount += _beneficiarySchedule.amountPerPeriod *
currentReleasePeriod;
}
```

Medium Severity Issues

No functionality to revoke the schedule

Resolved

Path

DKTVesting.sol

Description

The contract contains fields and validation for revoked schedules but lacks any function to actually revoke a beneficiary's schedule. This creates a critical functional gap in the vesting management system.

The contract lacks a critical administrative function needed for proper management of vesting schedules:

No ability to revoke schedules for non-compliant beneficiaries
No way to handle exceptional circumstances requiring revocation

Recommendation

Implement a proper revocation function with appropriate access controls

Vesting can’t be marked as true due to rounding errors

Resolved

Path

DKTVesting.sol

Function

claim(address beneficiary, uint256 beneficiaryScheduleId)

Description

The logic to mark a vesting schedule as completed uses a strict equality check that will fail due to rounding errors in the vesting calculations.

Due to integer division in multiple places (TGE calculations, period calculations), claimedAmount will almost never be exactly equal to totalAmount, preventing schedules from being marked as completed

```
if (currentReleasePeriod >= beneficiarySchedule.totalVestingPeriods &&
      beneficiarySchedule.claimedAmount == totalAmount) {
      beneficiarySchedule.isCompleted = true;
      // ...
}
```

Recommendation

Replace the strict equality check with a "greater than or equal" check for claimed amounts:

```
if (currentReleasePeriod >= beneficiarySchedule.totalVestingPeriods &&
      beneficiarySchedule.claimedAmount >= totalAmount - DUST_THRESHOLD) {
      beneficiarySchedule.isCompleted = true;
      // ...
}
```

deactivatedSchedules is not initialized

Resolved

Path

DKTVesting.sol

Description

`deactivatedSchedules` mapping is used to track whether a vesting schedule is deactivated, but it is not initialized in the initialize function or anywhere else in the contract. This means that all schedules are considered active by default, as the default value for a bool in a mapping is false. This could lead to unintended behavior if the contract relies on this mapping to deactivate schedules.

Recommendation

Initialize the deactivatedSchedules mapping

Low Severity Issues

Initialize the deactivatedSchedules mapping

Resolved

Path

Add a little more tokens while creating vesting schedules as a margin of rounding errors

Function

createBeneficiarySchedule(address beneficiary, uint256 scheduleId, uint256 amount)

Description

The contract creates vesting schedules in such a way where a particular amount is exactly divisible among beneficiaries. However, real life scenarios will pose rounding errors over various places. This inherent assumption can be broken causing less amount to be available to beneficiaries.

Recommendation

Add a little more tokens while creating vesting schedules as a margin of rounding errors

Centralization assumptions

Resolved

Description

The contract implements a highly centralized administrative model where the contract owner has unrestricted control over critical functions without any checks and balances.

The owner has unilateral power to pause the contract, preventing beneficiaries from claiming their tokens, and can upgrade the contract implementation.

Recommendation

Ensure that owner is a multi sig wallet.

Informational Severity Issues

nonReentrantPerCaller does not grant special functionality

Resolved

Path

DKTVesting.sol

Description

The contract implements a custom nonReentrantPerCaller modifier instead of using OpenZeppelin's standard nonReentrant modifier. While the intention is to prevent multiple operations from the same beneficiary at the same time, the implementation doesn't provide any additional protection beyond what the standard modifier would offer.

Recommendation

Replace the custom nonReentrantPerCaller modifier with OpenZeppelin's standard nonReentrant modifier.

whenNotPaused is not used in several functions

Resolved

Path

DKT.sol

Function

mint , batchTransfer

Description

whenNotPaused modifier is only implemented in the _beforeTokenTransfer function, but it's missing from other critical functions that should be paused during emergency situations. The main functions impacted include:
1. mint - Token minting continues even when the contract is paused
2. batchTransfer - Batch transfers can still be executed during pause state

Recommendation

Use whenNotPaused modifier in mint and batchTransfer function

Functional Tests

Some of the tests performed are mentioned below:

- Test contract initializes correctly with valid token address
- Test claim by non-beneficiary address
- Test claim functionality with multiple beneficiary schedules
- Blacklisted users cannot send or receive tokens
- Total supply remains constant after all transfers
- Paused contract blocks all transfers
- unpausing restores functionality.

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 dKloud. We performed our audit according to the procedure described above.

Issues of High,Medium,Low and Informational severity were found, which the dKloud Team Resolved.

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

QuillAudits

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

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com