



# AUDIT REPORT

---

April , 2025

For



# Table of Content

Table of Content	02
Executive Summary	04
Number of Issues per Severity	06
Checked Vulnerabilities	07
Techniques & Methods	09
Types of Severity	11
Types of Issues	12
<b>■ High Severity Issues</b>	13
1. Anybody can claim for anyone	13
2. amountInFinalPeriod is ignored in the calculation	14
<b>■ Medium Severity Issues</b>	15
1. No functionality to revoke the schedule	15
2. Vesting can't be marked as true due to rounding errors	16
3. deactivatedSchedules is not initialized	17
<b>■ Low Severity Issues</b>	18
1. Initialize the deactivatedSchedules mapping	18
2. Centralization assumptions	19
<b>■ Informational Severity Issues</b>	20
1. nonReentrantPerCaller does not grant special functionality	20
2. whenNotPaused is not used in several functions	21

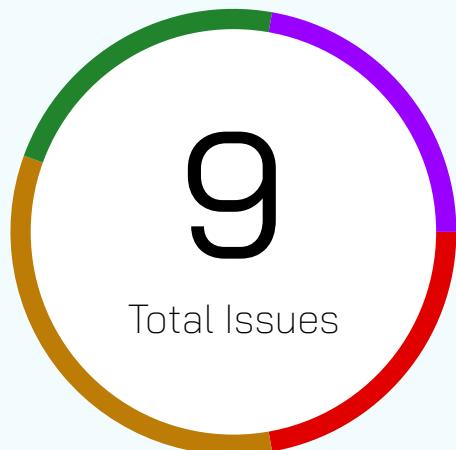
Functional Tests	22
Closing Summary & Disclaimer	23

# Executive Summary

<b>Project name</b>	dKloud
<b>Project URL</b>	<a href="https://www.dkloud.io">https://www.dkloud.io</a>
<b>Overview</b>	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).
<b>Audit Scope</b>	The scope of this Audit was to analyze the dKloud Smart Contracts for quality, security, and correctness.
<b>Source Code link</b>	<a href="https://github.com/dkloudio/DKT-Contract">https://github.com/dkloudio/DKT-Contract</a>
<b>Contracts in Scope</b>	DKTVesting.sol DKT.sol
<b>Branch</b>	Dev
<b>Commit Hash</b>	2ec20ba1c9a2e016e9e3e9e19f3231e2b0693c7c
<b>Language</b>	Solidity
<b>Blockchain</b>	Ethereum
<b>Method</b>	Manual Analysis, Functional Testing, Automated Testing
<b>Review 1</b>	6th March 2025 - 17th March 2025

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

# Number of Issues per Severity



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

Issues	Severity			
	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

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

Using inline assembly

Unsafe type inference

Style guide violation

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

<b>Open</b>  Security vulnerabilities identified that must be resolved and are currently unresolved.	<b>Resolved</b>  Security vulnerabilities identified that must be resolved and are currently unresolved.
<b>Acknowledged</b>  Vulnerabilities which have been acknowledged but are yet to be resolved.	<b>Partially Resolved</b>  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

DKTVesting.sol

### 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.

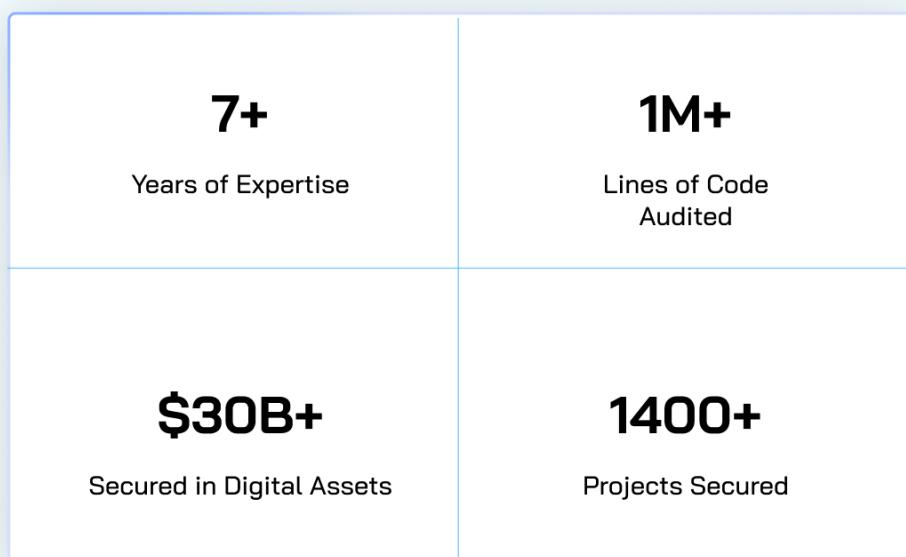
While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

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



Follow Our Journey



# AUDIT REPORT

---

April , 2025

For

[d]kloud



Canada, India, Singapore, UAE, UK

[www.quillaudits.com](http://www.quillaudits.com)    [audits@quillaudits.com](mailto:audits@quillaudits.com)