Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.12.17, the SlowMist security team received the KOLect team's security audit application for Daily Check In, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

KOLect is a research platform that scores crypto influencers based on verifiable market performance and on-chain signals. This audit pertains to the KOLect protocol's check-in contract, which allows users to earn points through daily on-chain check-ins.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	The storage of check-in variables can be optimized for gas efficiency	Gas Optimization Audit	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

Kolect_daily_check_in.sol

Hash(SHA256): 114f9439edf8fda5897839931331da05b4327ded26770e10b51fd4b0161bcae0

The main network address of the contract is as follows:

Contract Name	Contract Address	Chain
KolectDailyCheckIn	0xf5b88904C241fbB516d9B2ad8553c15E55e14307	Arbitrum One

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

KolectDailyCheckIn			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
_canCheckIn	Internal	-	-
checkIn	External	Can Modify State	whenNotPaused
canCheckIn	External	-	-
nextCheckInTime	External	-	-

KolectDailyCheckIn			
getPoints	External	-	-
getLastCheckIn	External	-	-
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
renounceOwnership	Public	-	onlyOwner

4.3 Vulnerability Summary

[N1] [Suggestion] The storage of check-in variables can be optimized for gas efficiency

Category: Gas Optimization Audit

Content

Within the checkIn function of the KolectDailyCheckIn contract, which is tasked with recording a user's last check-in time and incrementing their points, the current implementation utilizes two separate mappings (`lastCheckIn` and `points`) for data storage. This design results in each check-in operation triggering two distinct `SSTORE` write instructions (each writing to a different storage slot), consequently incurring higher gas costs. Considering that `block.timestamp` and point values generally do not necessitate the full 256-bit storage space, this segregated storage approach presents an opportunity for optimization.

Code location: Kolect_daily_check_in.sol#L60,L61

```
    function checkIn() external whenNotPaused {
        require(
            _canCheckIn(msg.sender, block.timestamp),
            "Kolect: can only check in once per day"
        );

        lastCheckIn[msg.sender] = block.timestamp;
        points[msg.sender] += 1;

        emit CheckedIn(msg.sender, block.timestamp, points[msg.sender]);
    }
```

Solution

It is recommended to define a `struct` to encapsulate these two variables, and then use a single mapping to store instances of this struct. This consolidation would merge the two separate writes into a single update to the same storage slot, thereby substantially reducing gas costs.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002512170001	SlowMist Security Team	2025.12.17 - 2025.12.17	Passed

Summary conclusion: The SlowMist security team uses a manual and the SlowMist team's analysis tool to audit the project. During the audit work, we found 1 suggestion. All the finding was acknowledged.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist