Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.10.23, the SlowMist security team received the Haust Lab team's security audit application for Haust Token Vesting, 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

This is the Haust Token vesting contract system for managing the linear release of tokens. It is mainly used in scenarios where tokens are released linearly over time. The contract supports the creation of multiple vesting plans, each of which can set parameters such as beneficiaries, cliff periods, release periods, and supports revocation and withdrawal functions.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Cross-Function Reentrancy in revoke Function	Reentrancy Vulnerability	Low	Fixed
N2	Lack of Validation on _start Timestamp	Design Logic Audit	Low	Fixed
N3	Denial-of-Service Risk in Batch Vesting Schedule Creation	Denial of Service Vulnerability	Low	Fixed
N4	Unbounded Growth of vestingSchedulesIds Array	Others	Suggestion	Fixed
N5	Low-level call reminder	Unsafe External Call Audit	Suggestion	Acknowledged
N6	Mismatched NatSpec documentation in createVestingSchedulesBatch	Others	Information	Fixed
N7	Token Distribution Mechanism Inconsistent with AUDIT_README Documentation	Others	Information	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/Haust-Labs/vesting

commit: 3bba111c53dd01dec791bfff9e7f61aade1b4dfb

Fixed Version:

https://github.com/Haust-Labs/vesting

commit: e1e1fb86818e42787e89ffbec1b6aa01d2997798

Audit Scope:

```
./contracts
└── TokenVesting.sol
```

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

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

TokenVesting			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getVestingSchedulesCountByBeneficiary	External	-	-
getVestingIdAtIndex	External	-	-
getVestingScheduleByAddressAndIndex	External	-	-
getVestingSchedulesTotalAmount	External	-	-
getToken	External	-	-
createVestingSchedule	External	Can Modify State	onlyOwner
_createVestingSchedule	Internal	Can Modify State	-
createVestingSchedulesBatch	External	Can Modify State	onlyOwner
revoke	External	Can Modify State	onlyOwner onlyIfVestingScheduleNotRevoked
withdraw	External	Can Modify State	nonReentrant onlyOwner
withdrawOtherTokens	External	Can Modify State	nonReentrant onlyOwner

TokenVesting			
release	Public	Can Modify State	nonReentrant onlyIfVestingScheduleNotRevoked
getVestingSchedulesCount	Public	-	-
computeReleasableAmount	External	-	onlyIfVestingScheduleNotRevoked
getVestingSchedule	Public	-	-
getWithdrawableAmount	Public	-	-
computeNextVestingScheduleIdForHolder	External	-	-
getLastVestingScheduleForHolder	External	-	-
computeVestingScheduleIdForAddressAndIndex	Public	-	-
getCurrentTime	Internal	-	-
_computeReleasableAmount	Internal	-	-
_safeReleaseNative	Internal	Can Modify State	-
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [Low] Cross-Function Reentrancy in revoke Function

Category: Reentrancy Vulnerability

Content

The revoke function lacks the nonReentrant modifier and violates the Checks-Effects-Interactions (CEI) pattern. The critical state update (vestingSchedule.revoked = true) occurs after an external call to the release function, creating a cross-function reentrancy. When the owner (if implemented as a smart contract) is also a beneficiary and attempts to revoke a vesting schedule, the external call in _safeReleaseNative can trigger the contract's receive() function. An attacker can exploit this callback to re-enter the revoke function before the first call completes. Since revoked is not

set to true until after the external call, the modifier onlyIfVestingScheduleNotRevoked check passes on the second call, allowing the function to execute twice with inconsistent state. This leads vestingSchedulesTotalAmount to reduce to 0 when vestingSchedulesTotalAmount is decremented twice for the same unreleased amount.

Code location:

contracts/TokenVesting.sol#L265-L277

```
    function revoke(bytes32 vestingScheduleId) external onlyOwner
onlyIfVestingScheduleNotRevoked(vestingScheduleId) {
        VestingSchedule storage vestingSchedule = vestingSchedules[vestingScheduleId];
        require(vestingSchedule.revocable == true, "TokenVesting: vesting is not
revocable");
        uint256 vestedAmount = _computeReleasableAmount(vestingSchedule);
        if (vestedAmount > 0) {
            release(vestingScheduleId, vestedAmount);
        }
        uint256 unreleased = vestingSchedule.amountTotal - vestingSchedule.released;
        vestingSchedulesTotalAmount = vestingSchedulesTotalAmount - unreleased;
        vestingSchedule.revoked = true;

        emit Revoked(vestingScheduleId, vestingSchedule.beneficiary, unreleased);
    }
```

Solution

It's recommended to add the nonReentrant modifier to the revoke function and inline the release logic instead of calling the function.

Status

Fixed

[N2] [Low] Lack of Validation on _start Timestamp

Category: Design Logic Audit

Content

The _createVestingSchedule function does not validate the reasonableness of the _start timestamp parameter. This allows the creation of vesting schedules with start times set to unreasonably far in the past or distant future, which may lead to unintended behavior, user errors, or exploitation scenarios. Without proper validation, the following problematic scenarios can occur:

If an owner makes a mistake (e.g., a "fat-finger" error or script bug) and sets _start to a time older than the _duration (e.g., _start = block.timestamp - 5 years for a 4-year duration), the vesting plan will be created successfully. However, the _computeReleasableAmount function will immediately see that currentTime >= vestingSchedule.start + vestingSchedule.duration is true, allowing the beneficiary to instantly release 100% of the supposedly "vested" tokens. This defeats the entire purpose of a vesting contract.

Code location:

contracts/TokenVesting.sol#L201-L232

```
    function _createVestingSchedule(
        …
     ) internal {
        require(
            _token.balanceOf(address(this)) - vestingSchedulesTotalAmount >= _amount,
            "TokenVesting: cannot create vesting schedule because not sufficient
tokens"
        );
        require(_beneficiary != address(0x0), "TokenVesting: beneficiary address
should be non-zero");
        require(_duration > 0, "TokenVesting: duration must be > 0");
        require(_amount > 0, "TokenVesting: amount must be > 0");
        require(_slicePeriodSeconds >= 1, "TokenVesting: slicePeriodSeconds must be
>= 1");
        require(_duration >= _cliff, "TokenVesting: no vesting possible with cliff
duration");
        ...
    }
```

Solution

It's recommended to add validation to ensure the _start timestamp falls within a reasonable range relative to the current block timestamp or >= block.timestamp.

Status

Fixed

[N3] [Low] Denial-of-Service Risk in Batch Vesting Schedule Creation

Category: Denial of Service Vulnerability

Content

The createVestingSchedulesBatch function lacks a maximum array length limit, which may cause transactions to fail

due to block gas limit constraints. When processing a large number of beneficiaries, the cumulative gas consumption from multiple _createVestingSchedule calls can exceed the block gas limit, resulting in transaction failure and wasted gas fees. Additionally, since the batch operation is atomic, if any individual schedule creation fails during iteration (e.g., due to insufficient tokens or invalid parameters), the entire batch transaction reverts, causing all previously successful operations within that transaction to be rolled back.

Code location:

contracts/TokenVesting.sol#L242-L259

```
    function createVestingSchedulesBatch(
        ...
    ) external onlyOwner {
        ...
        for (uint256 i = 0; i < _beneficiaries.length; i++) {
            _createVestingSchedule(
                _beneficiaries[i], _start, _cliff, _duration, _slicePeriodSeconds,
isRevocable, _amounts[i]
            );
        }
    }
```

Solution

It’s recommended to implement a maximum batch size limit to prevent gas exhaustion.

Status

Fixed

[N4] [Suggestion] Unbounded Growth of vestingSchedulesIds Array

Category: Others

Content

The vestingSchedulesIds array appends a new ID every time a schedule is created, but it never removes IDs, even after a schedule is completed or revoked. This leads to two issues: 1) Permanent storage bloat on-chain, slightly increasing gas costs for push operations over time. 2) A severe performance bottleneck for any off-chain client (like a dApp frontend) that tries to iterate this array using getVestingIdAtIndex(i).

Code location:

contracts/TokenVesting.sol#L227

```
vestingSchedulesIds.push(vestingScheduleId);
```

Solution

It’s recommended to add a separate counter to track active vesting schedules.

Status

Fixed

[N5] [Suggestion] Low-level call reminder

Category: Unsafe External Call Audit

Content

The native token transfers use the low-level calls, but it does not limit the amount of gas used to transfer native tokens.

Code location:

contracts/TokenVesting.sol#L441

```
(bool success,) = payable(to).call{value: amount}("");
```

Solution

When using low-level calls, it is recommended to limit the amount of gas used.

Status

Acknowledged

[N6] [Information] Mismatched NatSpec documentation in createVestingSchedulesBatch

Category: Others

Content

The NatSpec (documentation comment) for the createVestingSchedulesBatch function is incorrect and does not match the function's actual implementation. The @param comments for _cliff, _duration, and _slicePeriodSeconds incorrectly state that these parameters are arrays (e.g., array of cliff durations in seconds). However, the actual function signature accepts these parameters as single uint256 values. Only _beneficiaries and _amounts are array parameters.

Code location:

contracts/TokenVesting.sol#L234-L259

```
/**
  * @notice Creates multiple vesting schedules in a single transaction.
  * @param _beneficiaries array of beneficiary addresses
  * @param _cliff array of cliff durations in seconds
  * @param _duration array of vesting durations in seconds
  * @param _slicePeriodSeconds array of slice period durations
  * @param _amounts array of total amounts to be vested
  */
 function createVestingSchedulesBatch(
     address[] calldata _beneficiaries,
     uint256[] calldata _amounts,
     uint256 _start,
     uint256 _cliff,
     uint256 _duration,
     uint256 _slicePeriodSeconds,
     bool isRevocable
 ) external onlyOwner {...}
```

Solution

It's recommended to update the NatSpec documentation to accurately reflect the function parameters.

Status

Fixed

[N7] [Information] Token Distribution Mechanism Inconsistent with AUDIT_README Documentation

Category: Others

Content

The AUDIT_README.md document describes the contract as distributing "native HAUST tokens" to beneficiaries, with an automatic conversion from WETH9 to HAUST during the release process. However, the actual implementation only converts WETH to native ETH (or sends WETH as fallback), with no HAUST token minting, conversion, or distribution logic present in the contract. This represents a fundamental mismatch between the documented design intent and the actual implementation, which may indicate either incomplete implementation or outdated documentation.

Solution

If ETH/WETH distribution is the actual intent, it is recommended to update the AUDIT_README.md documentation to accurately reflect the implementation.

Status

Fixed; After communicating with the project team, they stated that the HAUST is ETH and WHAUST is WETH because its native and wrapped tokens of their Haust network. And the AUDIT_README.md is modifed in the fixed version.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002510240001	SlowMist Security Team	2025.10.23 - 2025.10.24	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 identified 3 low risks, 2 suggestions, and 2 information. All the findings were fixed or acknowledged. The code was not deployed to the mainnet.

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