Tradoor BSC Vesting

Audit Report

Mon Aug 25 2025

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

Tradoor BSC Vesting Audit Report

1 Executive Summary

1.1 Project Information

Description	It is a token vesting project.
Type	DeFi
Auditors	ScaleBit
Timeline	Wed Aug 20 2025 - Wed Aug 20 2025
Languages	Solidity
Platform	BSC
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/TonTradoor/bsc-vesting
Commits	ef21e1519dc1730fd506b37d02f0e81abb7b3fbb 29c83e2443e33def3ef02487a7331228ba1c94a5

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
TVE	contracts/TokenVesting.sol	7cf194c781c2e1b8a7dc825cd926dc9868548189
TOK	contracts/Token.sol	76dd72fad9c7a892591ebe43a008acdc9ab5aabd

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	3	0
Informational	2	2	0
Minor	1	1	0
Medium	0	0	0
Major	0	0	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Tradoor to identify any potential issues and vulnerabilities in the source code of the Tradoor BSC Vesting smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 3 issues of varying severity, listed below.

ID	Title	Severity	Status
TVE-1	Unnecessary ETH Reception Functions	Minor	Fixed
TVE-2	Unused Import	Informational	Fixed
TVE-3	Lack of Event Emit	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Tradoor BSC Vesting Smart Contract :

Owner

- The owner can create a vesting schedule through `createVestingSchedule()` .
- The owner can revoke a vesting schedule through `revoke()` .
- The owner can update the `tge` time through `updateTge()` .

User

- The users can claim their released tokens through `claim()` .

4 Findings

TVE-1 Unnecessary ETH Reception Functions

Severity: Minor

Status: Fixed

Code Location:

contracts/TokenVesting.sol#64,70

Descriptions:

The contract TokenVesting includes a `receive()` and a `fallback()` function, both payable, allowing the contract to receive Ether (ETH). However, the core logic of the contract is designed exclusively for vesting and managing ERC20 tokens. There is no internal logic to handle, store, or distribute any received ETH. Consequently, any Ether sent to this contract will be permanently locked and inaccessible, as there are no functions to withdraw it.

Suggestion:

Remove the `receive()` and `fallback()` functions unless there is a specific requirement to accept ETH payments. If the contract must receive ETH for future upgrades or specific features, implement a secure withdrawal mechanism (e.g., a function guarded by the onlyOwner modifier) to recover accidentally sent Ether. Since this contract is for ERC20 vesting, the most secure practice is to remove these functions entirely to prevent any accidental loss of funds.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TVE-2 Unused Import

Severity: Informational

Status: Fixed

Code Location:

contracts/TokenVesting.sol#8

Descriptions:

In `TokenVesting.sol`, there is an unused import `import "hardhat/console.sol";`

Suggestion:

It is recommended to remove this unused import.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TVE-3 Lack of Event Emit

Severity: Informational

Status: Fixed

Code Location:

contracts/TokenVesting.sol#125

Descriptions:

The `revoke()` function in the contract lacks event logging, which is essential for blockchain transparency, off-chain data tracking, and frontend integration. Event logs allow external systems to monitor contract activities without querying the blockchain state directly.

Suggestion:

It is recommended to add an event.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer