Boost EVM

Security Audit Report

prepared by

OKX Web3 Audit Team

2025.08.07

Contents

1. Overview

1.1 About Boost - TokenDistributor (EVM)

TokenDistributor (EVM) is a decentralized token distribution system based on EVM, designed to efficiently distribute tokens to a large number of recipients using Merkle tree proofs. The system consists of a factory contract that allows anyone to create token distribution campaigns and individual distributor contracts that handle the actual token claiming process.

1.2 Audit Summary

Ecosystem	EVM	**Language**	Solidity
Repository	https://github.com/okxlabs/Boost-TokenDistributor-EVM		
Base Commit	24e166ffb309f913ba8cb93de371642ede015e2b		
Final Commit	24e166ffb309f913ba8cb93de371642ede015e2b		

1.3 Audit Scope

```
contracts/
 ├── TokenDistributor.sol      # Main distribution contract
 └── DistributorFactory.sol    # Factory contract for creating distributors
```

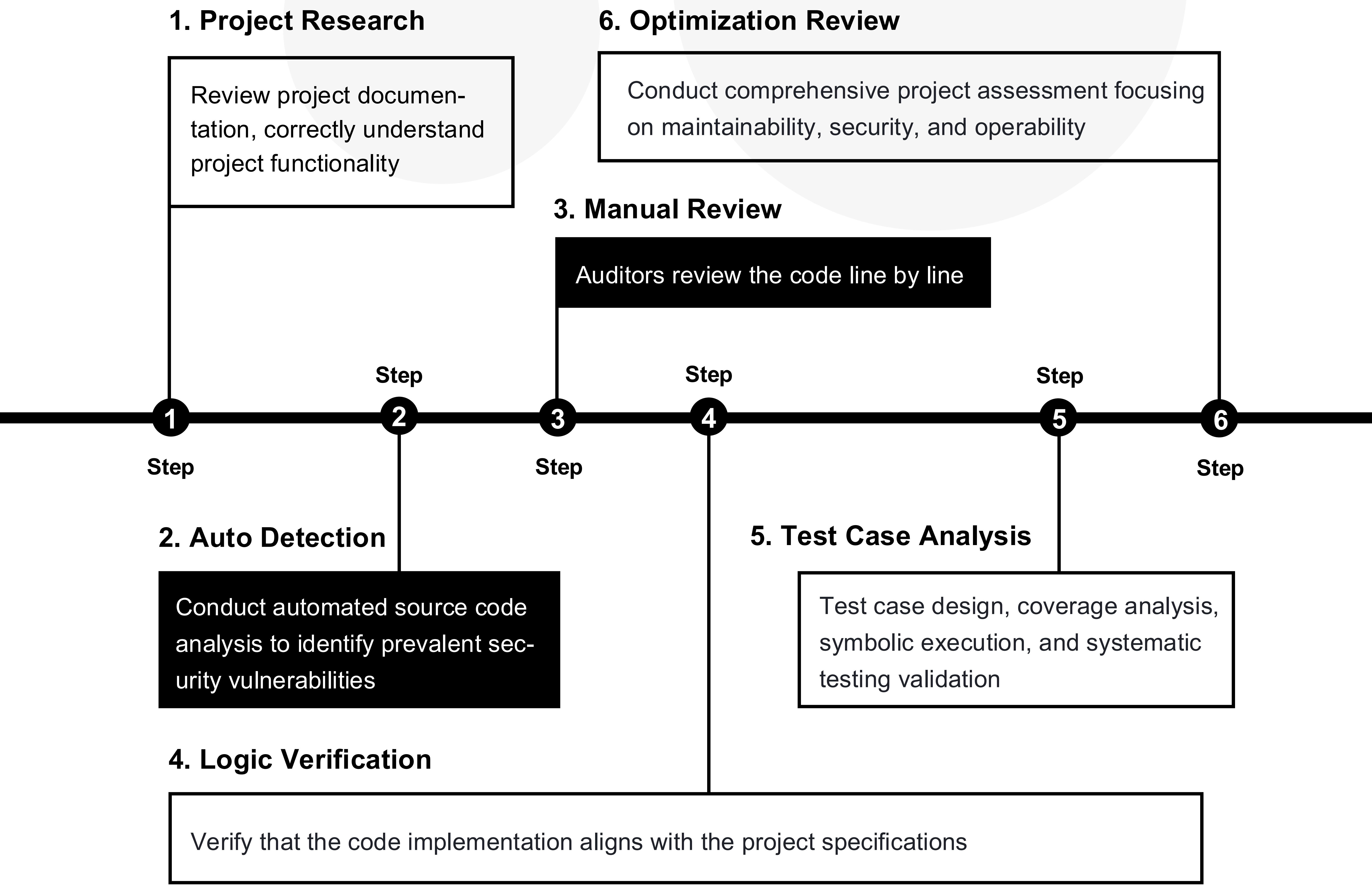
2. Audit Summary

2.1 Audit Methodology

The audit team conducted comprehensive analysis of the contract code through deep understanding of the project's design purpose, operating principles, and implementation methods. By mapping function call relationships, potential security vulnerabilities were systematically identified, with detailed problem descriptions and corresponding remediation recommendations provided.

2.2 Audit Process

The smart contract security audit follows a 6-phase process: Project Research, Automated Detection, Manual Review, Logic Verification, Test Case Analysis, and Optimization Review. During manual auditing, auditors perform comprehensive code review to identify vulnerabilities and provide detailed solutions. After completing all phases, the lead auditor communicates findings with the project team. Following the team's responses, we deliver final audit reports to the project team.



1. Project Research

Review project documentation, correctly understand project functionality

2. Auto Detection

Conduct automated source code analysis to identify prevalent security vulnerabilities

3. Manual Review

Auditors review the code line by line

4. Logic Verification

Verify that the code implementation aligns with the project specifications

5. Test Case Analysis

Test case design, coverage analysis, symbolic execution, and systematic testing validation

6. Optimization Review

Conduct comprehensive project assessment focusing on maintainability, security, and operability

2.3 Risk Classification and Description

Risk items are classified into 5 levels: Critical, High, Medium, Low, and Informational. Critical risks require immediate resolution and re-audit before final report delivery; unresolved critical risks result in audit failure. High risks must be addressed but are less urgent; failure to resolve also results in audit failure. Medium risks indicate potential exposure and require clear documentation of project team notification and response status without affecting report delivery. Low risks and informational items involve compliance or code detail issues that may be deferred without impacting report delivery.

Risk Level	**Icon**	**Risk Description**
Critical	🚨	Fatal risks requiring immediate resolution
High	‼	High-risk vulnerabilities that will cause similar issues, must be resolved
Medium	⚠	Medium-risk vulnerabilities with potential impact, should be resolved
Low	❗	Low-risk issues with improper handling or warning triggers, can be deferred
Informational	❕	Optimization opportunities, deferrable but recommended for resolution

2.4 Vulnerability Checklist

The vulnerability checklist is divided into two parts: one part is the vulnerability summary of the project audit, and the other part is the detailed vulnerability list.

- **Vulnerability Summary:**

Critical	**High**	**Medium**	**Low**	**Informational**	**Total**
0	**0**	**0**	**1**	**3**	**4**

- **Vulnerability list:**

No.	**Severity**	**Vulnerability**	**Category**	**Status**
1	Low	Centralization Risk	Centralization Risk	Confirmed
2	Info	Naming Error	Coding	Fixed
3	Info	Redundant Checks	Coding	Fixed
4	Info	Comment Errors	Coding	Fixed

- **Open:** The audit team has notified the project team of the vulnerability, but no reasonable remediation has been implemented.
- **Fixed:** The project team has addressed the vulnerability and the fix has been verified by the audit team.
- **Confirmed:** The project team has confirmed awareness of the vulnerability risk but considers it controllable.

3. Vulnerabilities

This section outlines the risk items identified through manual review and auditing tools. Each item includes the specific file path and code location, along with the assigned risk level. Detailed descriptions of the risks, recommended remediation measures, and relevant code snippets are provided to help clearly illustrate each issue.

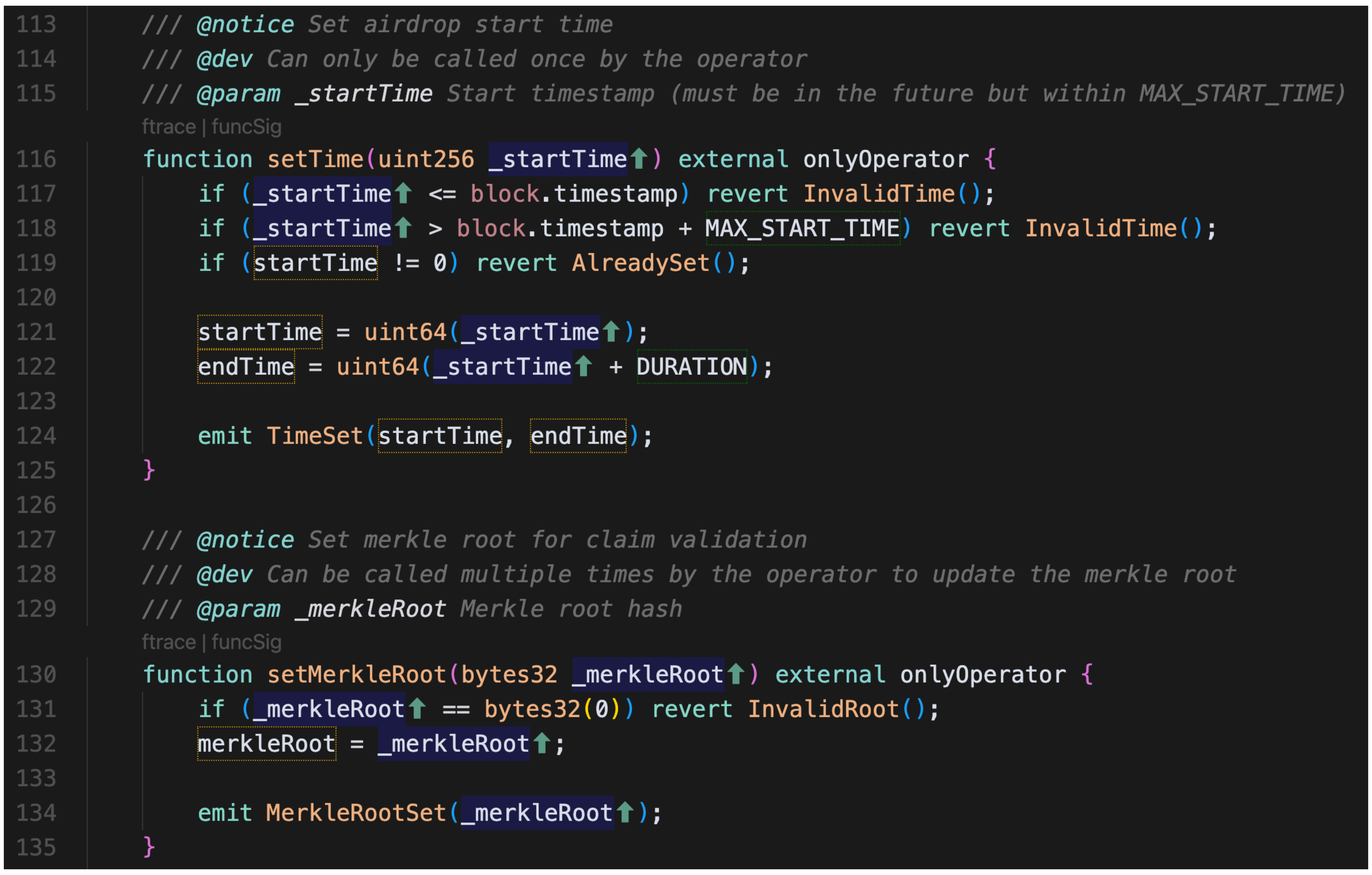
3.1 Centralization Risk

Location	File	Status	Severity
Multiple places	TokenDistributor.sol	Confirmed	!Low

- **Description**

The current contract implementation relies on two administrator permissions, Owner and Operator. The Owner can withdraw the remaining tokens from the contract after the distribution is completed/before setting the trigTime. The Operator can set the airdrop start time, set and modify the Merkel root for receiving airdrops. The normal operation of the contract depends on the safe operation of these administrators. If these administrators are not properly protected, it will affect the normal operation of the contract and cause asset losses.

- **Related Code**



```
/// @notice Set airdrop start time
/// @dev Can only be called once by the operator
/// @param _startTime Start timestamp (must be in the future but within MAX_START_TIME)
ftrace | funcSig
function setTime(uint256 _startTime) external onlyOperator {
    if (_startTime <= block.timestamp) revert InvalidTime();
    if (_startTime > block.timestamp + MAX_START_TIME) revert InvalidTime();
    if (startTime != 0) revert AlreadySet();

    startTime = uint64(_startTime);
    endTime = uint64(_startTime + DURATION);

    emit TimeSet(startTime, endTime);
}

/// @notice Set merkle root for claim validation
/// @dev Can be called multiple times by the operator to update the merkle root
/// @param _merkleRoot Merkle root hash
ftrace | funcSig
function setMerkleRoot(bytes32 _merkleRoot) external onlyOperator {
    if (_merkleRoot == bytes32(0)) revert InvalidRoot();
    merkleRoot = _merkleRoot;

    emit MerkleRootSet(_merkleRoot);
}
```

- **Recommendation**

Properly delegate relevant administrator permissions to the asset management group for management.

- **Project Team Feedback**

Team Response	Confirmed
Re-audit Result	Accept

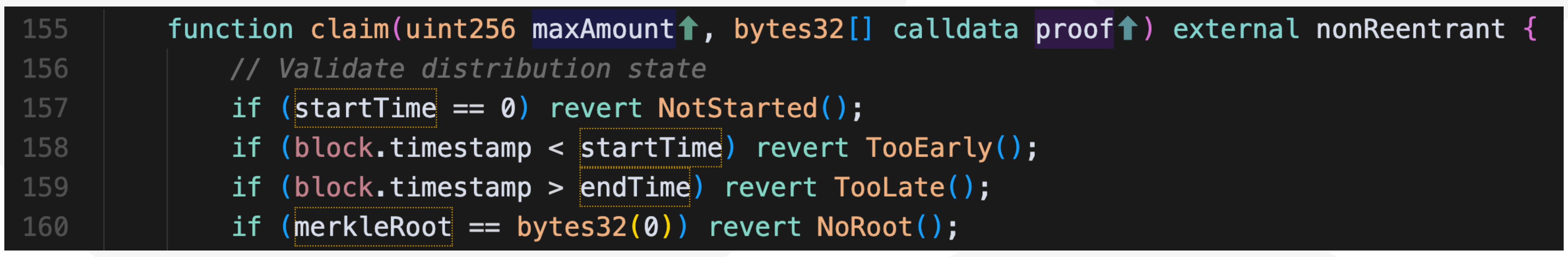
3.2 Naming Error

Location	File	Status	Severity
Line 155-160	TokenDistributor.sol	Fixed	! Info

- **Description**

The error message name does not match the code logic, and the error message returned when sending an error report will cause ambiguity.

- **Related Code**



```
155     function claim(uint256 maxAmount, bytes32[] calldata proof) external nonReentrant {
156         // Validate distribution state
157         if (startTime == 0) revert NotStarted();
158         if (block.timestamp < startTime) revert TooEarly();
159         if (block.timestamp > endTime) revert TooLate();
160         if (merkleRoot == bytes32(0)) revert NoRoot();
```

- **Recommendation**

We recommend modifying the error message to reflect the code logic, for example, to StartTimeNotSet() .

- **Project Team Feedback**

Team Response	Fixed
Re-audit Result	Confirmed

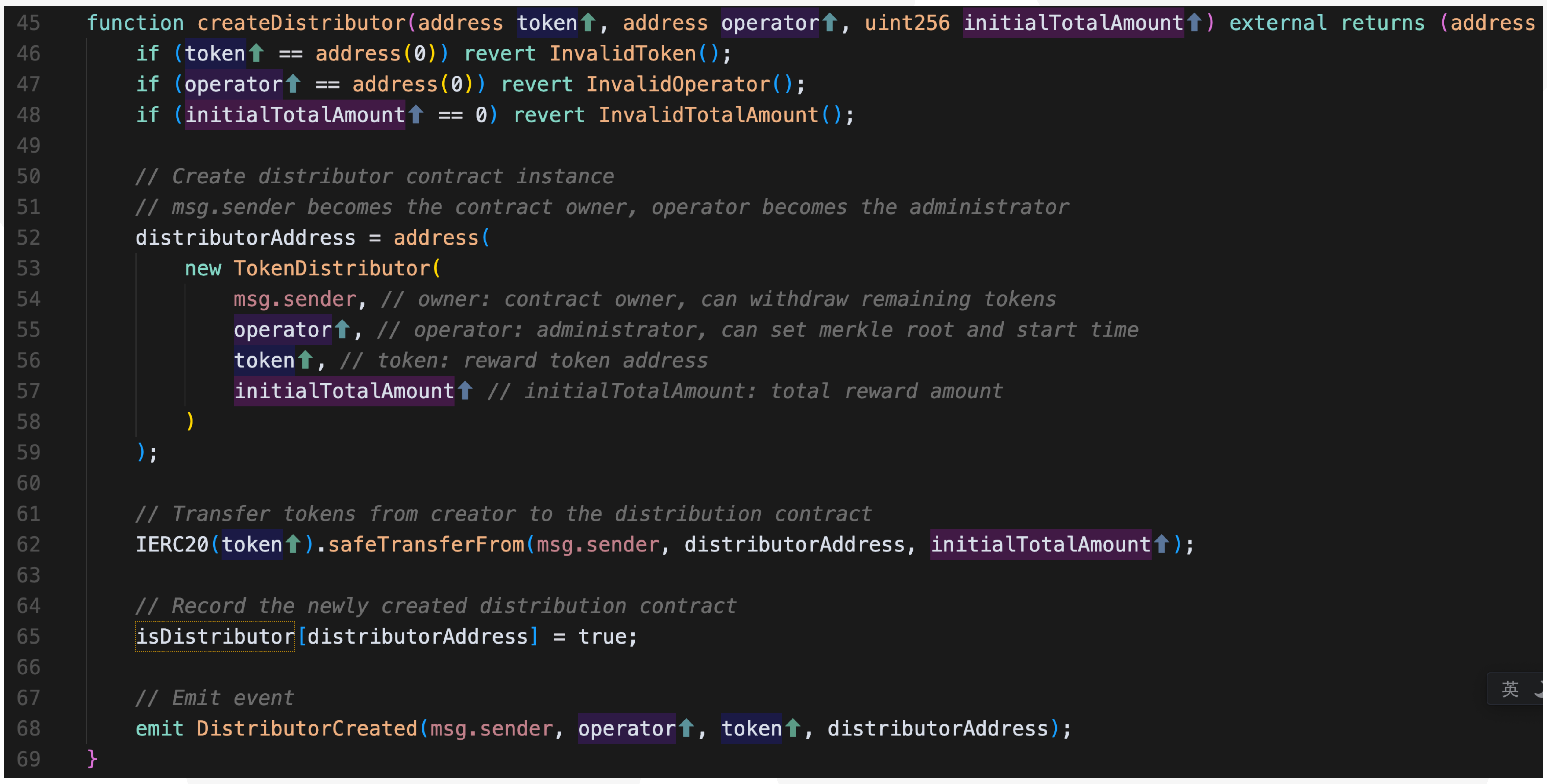
3.3 Redundant Checks

Location	File	Status	Severity
Line 45-69	DistributorFactory.sol	Fixed	! Info

- **Description**

There are redundant checks in the createDistributor function of DistributorFactory.

- **Related Code**



```
45  function createDistributor(address token, address operator, uint256 initialTotalAmount) external returns (address
46      if (token == address(0)) revert InvalidToken();
47      if (operator == address(0)) revert InvalidOperator();
48      if (initialTotalAmount == 0) revert InvalidTotalAmount();
49
50      // Create distributor contract instance
51      // msg.sender becomes the contract owner, operator becomes the administrator
52      distributorAddress = address(
53          new TokenDistributor(
54              msg.sender, // owner: contract owner, can withdraw remaining tokens
55              operator, // operator: administrator, can set merkle root and start time
56              token, // token: reward token address
57              initialTotalAmount // initialTotalAmount: total reward amount
58          )
59      );
60
61      // Transfer tokens from creator to the distribution contract
62      IERC20(token).safeTransferFrom(msg.sender, distributorAddress, initialTotalAmount);
63
64      // Record the newly created distribution contract
65      isDistributor[distributorAddress] = true;
66
67      // Emit event
68      emit DistributorCreated(msg.sender, operator, token, distributorAddress);
69  }
```

- **Recommendation**

Suggest removing redundant checks for createDistributor.

- **Project Team Feedback**

Team Response	Fixed
Re-audit Result	Confirmed

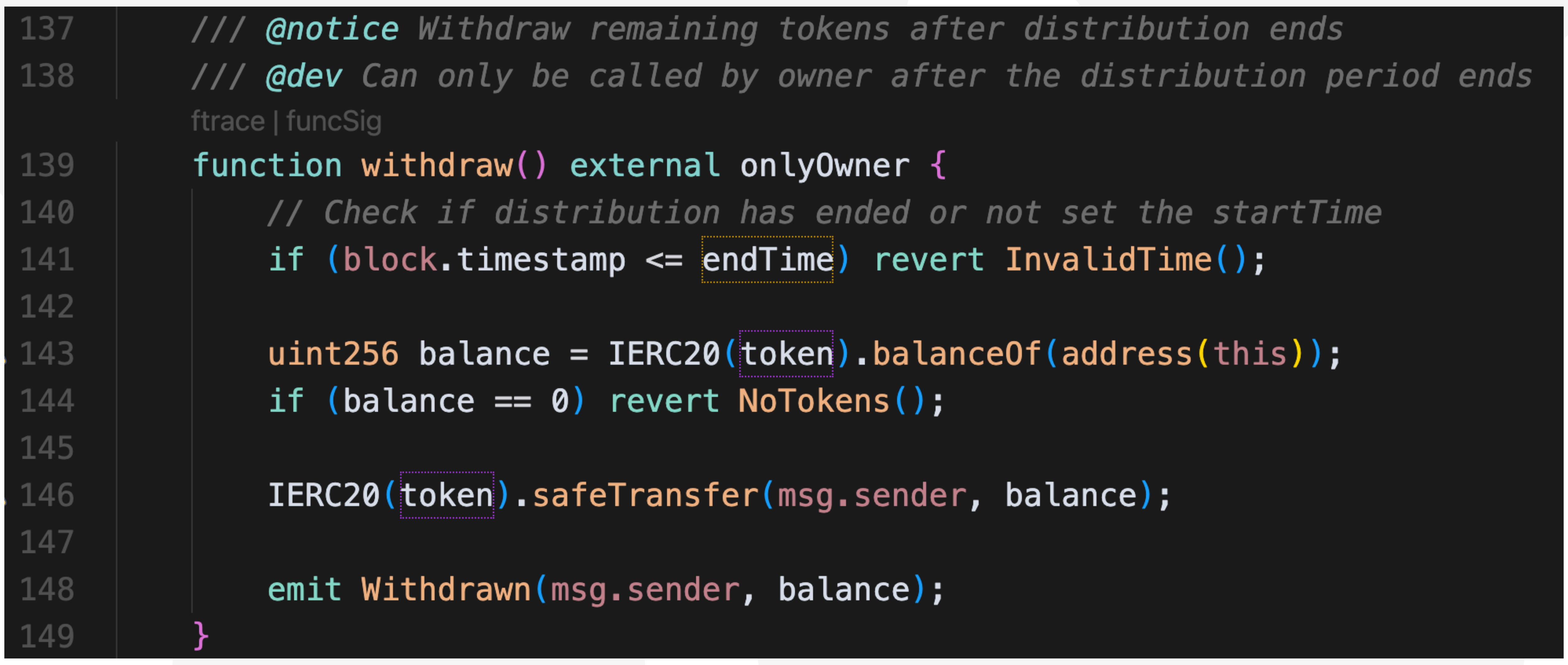
3.4 Comment Errors

Location	File	Status	Severity
Line 137-149	TokenDistributor.sol	Fixed	! Info

- **Description**

According to the documentation, when startTime is not set, the project party can retrieve the token. The comment here has not been updated.

- **Related Code**



```
137     /// @notice Withdraw remaining tokens after distribution ends
138     /// @dev Can only be called by owner after the distribution period ends
        ftrace | funcSig
139     function withdraw() external onlyOwner {
140         // Check if distribution has ended or not set the startTime
141         if (block.timestamp <= endTime) revert InvalidTime();
142
143         uint256 balance = IERC20(token).balanceOf(address(this));
144         if (balance == 0) revert NoTokens();
145
146         IERC20(token).safeTransfer(msg.sender, balance);
147
148         emit Withdrawn(msg.sender, balance);
149     }
```

- **Recommendation**

According to the documentation, it is supported for the project party to retrieve the Token when StartTime is not set. The comment here has not been updated.

- **Project Team Feedback**

Team Response	Fixed
Re-audit Result	Confirmed

4. Disclaimer

This audit report only covers the specific audit types stated herein. We assume no responsibility for unknown security vulnerabilities outside this scope.

We rely on audit reports issued before existing attacks or vulnerabilities are published. For future or new vulnerabilities, we cannot guarantee project security impact and assume no responsibility.

Our security audit analysis should be based on documents provided by the project before report release (including contract code). These materials should not contain false information, tampering, deletion, or concealment. If provided materials are false, inaccurate, missing, tampered, deleted, concealed, or modified after report release, we assume no responsibility for resulting losses and adverse effects.

The project team should understand our audit report is based on provided materials and current technical capabilities. Due to institutional technical limitations, our report may not detect all risks. We encourage continued testing and auditing by the development team and stakeholders.

The project team must ensure compliance with applicable laws and regulations.

The audit report is for reference only. Its content, acquisition methods, usage, and related services cannot serve as basis for investment, taxation, legal, regulatory, or construction decisions. Without our prior written consent, the project team may not reference, cite, display, or distribute report content to third parties. Any resulting losses shall be borne by the project team.

This report does not cover contract compiler bugs or scope beyond programming languages. Smart contract risks from underlying vulnerabilities should be borne by the project team.

Force majeure includes unforeseeable, unavoidable events like wars, natural disasters, strikes, epidemics, and legal/regulatory changes preventing contract performance. When occurring, neither party breaches contract obligations. For unaffected economic responsibilities, the project team should pay for completed work.

5. About Us

OKX Web3 Audit Team specializes in blockchain security with expertise in smart contract auditing, token security assessment, and Web3 security tool development. We provide comprehensive security solutions for OKX's internal Web3 projects, conduct pre-listing token audits, and develop security tools to protect OKX Web3 wallet users. Our team combines automated analysis with manual review to deliver thorough security assessments and maintain the highest security standards in the Web3 ecosystem.

PASSED

This audit has been conducted to review the Boost EVM project's Solidity-based smart contracts running on EVM chains, examining design, architecture, and implementation to identify potential vulnerabilities

OKX Web3 Audit Team