

Smart Contract Security Audit Report

The SlowMist Security Team received the Primus team's application for smart contract security audit of the PRIM on 2026.04.07. The following are the details and results of this smart contract security audit:

Token Name :

PRIM

File name and hash (SHA256) :

primus-contracts.zip: 4d27486553b46d797a80d73885bc4aa7821a71ff3dbb49e03651b99f192e2a48

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability Audit	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed

NO.	Audit Items	Result
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002604080001

Audit Date : 2026.04.07 - 2026.04.08

Audit Team : SlowMist Security Team

Summary conclusion : This is an OFT token contract that includes a token distribution module. The total amount of contract tokens remains unchanged. The contract does not have the Overflow and the Race Conditions issue.

The source code:

PRIM.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity 0.8.30;

import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
import {ERC20Permit} from
"@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";
import {ERC20Burnable} from
"@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import {OFT} from "@layerzerolabs/oft-evm/contracts/OFT.sol";

/// @notice OFT is an ERC-20 token that extends the OFTCore contract.
contract PRIM is OFT, ERC20Permit, ERC20Burnable {
    constructor(string memory _name, string memory _symbol, address _lzEndpoint,
address _owner, address _tempOwner)
        OFT(_name, _symbol, _lzEndpoint, _tempOwner)
        Ownable(_tempOwner)
        ERC20Permit(_name)
    {
        if (block.chainid == 84532) {
```

```
            _mint(_owner, 1_000_000_000 * 10 ** decimals());
        }
    }
}
```

MerkleDistributor.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;

import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
import {Pausable} from "@openzeppelin/contracts/utils/Pausable.sol";
import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import {SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import {MerkleProof} from
"@openzeppelin/contracts/utils/cryptography/MerkleProof.sol";

// src/interfaces/IMerkleDistributor.sol

/// @title MerkleDistributor
/// @notice Allows anyone to claim a token if they exist in a merkle root.
interface IMerkleDistributor {
    /// @notice Emitted whenever a call to claim succeeds.
    event Claimed(uint256 index, address indexed account, uint256 amount);

    /// @notice Thrown when an account already claimed.
    error AlreadyClaimed();

    /// @notice Thrown when the provided merkle proof is invalid.
    error InvalidProof();

    /// @notice Thrown when the end time is in the past.
    error EndTimeInPast();

    /// @notice Thrown when the claim window is finished.
    error ClaimWindowFinished();

    /// @notice Thrown when a withdrawal is attempted by the owner during the claim
window.
    error NoWithdrawDuringClaim();

    /// @notice Returns the address of the token distributed by this contract.
```

```
    function token() external view returns (address);

    /// @notice Returns the merkle root of the merkle tree.
    /// @dev The merkle tree contains the account balances available to claim.
    function merkleRoot() external view returns (bytes32);

    /// @notice Returns the end time of the distribution.
    function endTime() external view returns (uint256);

    /// @notice Returns true if the index has been marked as claimed.
    /// @param index The index of the account in the merkle tree.
    function isClaimed(uint256 index) external view returns (bool);

    /// @notice Claims the specified amount to the specified account address.
    /// @dev Reverts if the inputs are invalid for the merkle proof.
    /// @param index The index of the account in the merkle tree.
    /// @param account The address of the account to claim for.
    /// @param amount The amount of the token to claim.
    /// @param merkleProof The merkle proof for the account.
    function claim(uint256 index, address account, uint256 amount, bytes32[] calldata
merkleProof) external;

    /// @notice Withdraws the entire token balance to the specified recipient.
    /// @dev Only callable by the owner after the claim window is finished.
    /// @param recipient The address of the account to withdraw to.
    function withdraw(address recipient) external;

    /// @notice Pauses the contract.
    /// @dev Only callable by the owner.
    function pause() external;

    /// @notice Unpauses the contract.
    /// @dev Only callable by the owner.
    function unpause() external;
}

// src/MerkleDistributor.sol

/// @title MerkleDistributor
/// @notice Allows anyone to claim a token if they exist in a merkle root.
contract MerkleDistributor is Ownable, Pausable, IMerkleDistributor {
    using SafeERC20 for IERC20;

    /// @dev Immutable storage for token address (SCREAMING_SNAKE_CASE per lint)
```

```
    address internal immutable TOKEN;

    /// @dev Immutable storage for merkle root (SCREAMING_SNAKE_CASE per lint)
    bytes32 internal immutable MERKLE_ROOT;

    /// @dev Immutable storage for end time (SCREAMING_SNAKE_CASE per lint)
    uint256 internal immutable END_TIME;

    /// @dev This is a packed array of booleans.
    mapping(uint256 => uint256) private claimedBitMap;

    /// @dev The token and merkle root can only be set once in the constructor.
    //SlowMist// It is recommended to add a zero address check to prevent the TOKEN
from being set to a zero address.
    constructor(address _token, bytes32 _merkleRoot, uint256 _endTime, address
_owner) Ownable(_owner) {
        if (_endTime <= block.timestamp) revert EndTimeInPast();
        TOKEN = _token;
        MERKLE_ROOT = _merkleRoot;
        END_TIME = _endTime;
        _pause();
    }

    /// @inheritdoc IMerkleDistributor
    function token() public view override returns (address) {
        return TOKEN;
    }

    /// @inheritdoc IMerkleDistributor
    function merkleRoot() public view override returns (bytes32) {
        return MERKLE_ROOT;
    }

    /// @inheritdoc IMerkleDistributor
    function endTime() public view override returns (uint256) {
        return END_TIME;
    }

    /// @inheritdoc IMerkleDistributor
    function isClaimed(uint256 index) public view override returns (bool) {
        uint256 claimedWordIndex = index / 256;
        uint256 claimedBitIndex = index % 256;
        uint256 claimedWord = claimedBitMap[claimedWordIndex];
        uint256 mask = (uint256(1) << claimedBitIndex);
```

```
        return claimedWord & mask == mask;
    }

    /// @dev Marks an index as claimed.
    function _setClaimed(uint256 index) private {
        uint256 claimedWordIndex = index / 256;
        uint256 claimedBitIndex = index % 256;
        claimedBitMap[claimedWordIndex] = claimedBitMap[claimedWordIndex] |
(uint256(1) << claimedBitIndex);
    }

    /// @inheritdoc IMerkleDistributor
    function claim(uint256 index, address account, uint256 amount, bytes32[] calldata
merkleProof)
        public
        virtual
        override
        whenNotPaused
    {
        // Ensure the claim window is not finished.
        if (block.timestamp > END_TIME) revert ClaimWindowFinished();

        // Ensure the index has not been marked as claimed.
        if (isClaimed(index)) revert AlreadyClaimed();

        // Verify the merkle proof.
        bytes32 node;
        assembly {
            // Get free memory pointer
            let ptr := mload(0x40)
            // Store index (32 bytes)
            mstore(ptr, index)
            // Store account (20 bytes, left-shifted to pack)
            mstore(add(ptr, 0x20), shl(96, account))
            // Store amount (32 bytes)
            mstore(add(ptr, 0x34), amount)
            // Hash the packed data (32 + 20 + 32 = 84 bytes = 0x54)
            node := keccak256(ptr, 0x54)
        }
        if (!MerkleProof.verify(merkleProof, MERKLE_ROOT, node)) revert
InvalidProof();

        // Mark the index as claimed and transfer the token.
        _setClaimed(index);
```

```
        IERC20(TOKEN).safeTransfer(account, amount);

        // Emit the event.
        emit Claimed(index, account, amount);
    }

    /// @inheritdoc IMerkleDistributor
    function withdraw(address recipient) external onlyOwner {
        if (block.timestamp < END_TIME) revert NoWithdrawDuringClaim();
        IERC20(TOKEN).safeTransfer(recipient, IERC20(TOKEN).balanceOf(address(this)));
    }

    /// @inheritdoc IMerkleDistributor
    function pause() external whenNotPaused onlyOwner {
        _pause();
    }

    /// @inheritdoc IMerkleDistributor
    function unpause() external whenPaused onlyOwner {
        _unpause();
    }
}
```

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