SLOWMIST

Smart Contract Security Audit Report

The SlowMist Security Team received the Trusta AI team's application for smart contract security audit of the Trusta AI on 2025.03.24. The following are the details and results of this smart contract security audit:

Token Name :

Trusta AI

The contract address :

https://github.com/TrustaLabs/TA_contract

commit:c9fec6e02725d37c9b07d3687c9b1540a0f4dc38

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
13	Safety Design Audit	Passed

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

Audit Result : Passed

Audit Number : 0X002503260001

Audit Date : 2025.03.24 - 2025.03.26

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the vault section and the dark coin functions. The total amount of contract tokens remains unchangeable. The contract does not have the Overflow and the Race Conditions issue.

During the audit,we found the following information:

1. In the Claim contract, the owner can configure the airdrop details, allowing users to claim the airdrop within the specified time frame.

The source code:

Claim.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.0;

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

contract Claim is Ownable{
    using SafeERC20 for IERC20;

    IERC20 public immutable token;

    bytes32 public merkleRoot;

    uint256 public startTime;
    uint256 public endTime;

    address public vault;
    // leaf => hasClaimed
    mapping(bytes32 => bool) public hasClaimed;

    event Claimed(address indexed account, uint256 amount);

    constructor(address _token) Ownable(_msgSender()){
        require(_token != address(0), "Zero check");
        token = IERC20(_token);
    }

    // set airdrop info
    //SlowMist// The owner can set the airdrop information, including the vault
address, the merkle root, the start time, and the end time.
    function config(address _vault, bytes32 _root, uint256 _startTime, uint256
_endTime) external onlyOwner {
        vault = _vault;
        merkleRoot = _root;
        startTime = _startTime;
        endTime = _endTime;
    }


    // check the proof
    function check_prrof(address claimer, uint256 amount, bytes32[] calldata
merkleProof) public view returns (bool) {
        bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode(claimer,
amount))));
        bool result = MerkleProof.verify(merkleProof, merkleRoot, leaf);
        return result;
    }

    // claim airdrop using merkle proof
    //SlowMist// The user can claim the airdrop within the specified time frame.
    function claim(uint256 amount, bytes32[] calldata merkleProof) public {
        require(block.timestamp <= endTime, "Claim is end");
        require(block.timestamp >= startTime, "Not started");
        require(merkleRoot != bytes32(0), "Not configured");
        bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode(msg.sender,
amount))));
        require(!hasClaimed[leaf], "Already claimed");
        require(MerkleProof.verify(merkleProof, merkleRoot, leaf), "Invalid proof");
        hasClaimed[leaf] = true;
        SafeERC20.safeTransferFrom(token, vault, msg.sender, amount);
        emit Claimed(msg.sender, amount);
    }

}
```

TA.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";

contract TA is ERC20, ERC20Permit{

    constructor() ERC20("Trusta AI", "TA") ERC20Permit("Trusta AI") {
        _mint(_msgSender(), 1_000_000_000 * 10 ** decimals());
    }
}
```

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.