# Smart Contract
# Security Audit Report

# Table Of Contents

# 1 Executive Summary

On 2023.09.07, the SlowMist security team received the Good Entry Token team's security audit application for Good Entry Token, 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 GETOKEN contract that contains the Crowdsale section,GoodAirdrop section,GoodGovernor section,esGOOD section.

Crowdsale section is a crowdsale contract that allows users to purchase tokens using USDC and receive the purchased tokens after the crowdsale ends.

esGOOD section implement a token contract with specific functions, including crowdfunding, reward tracking, and token locking.

GoodAirdrop section is mainly used for issuing airdrops.

The GoodGovernor section implements a set of on-chain governance protocols for managing and executing proposals, including upgrades, integration of other protocols, fund management, authorization, etc.

## 3.2 Vulnerability Information

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

| NO | Title | Category | Level | Status |
| --- | --- | --- | --- | --- |
| N1 | Risk of excessive authority | Authority Control Vulnerability Audit | Medium | Acknowledged |
| N2 | Duplicate claim and unchecked totalAirdropAmount vulnerability | Design Logic Audit | Critical | Fixed |
| N3 | Transfer and transferFrom function misuse vulnerability | Design Logic Audit | Critical | Fixed |
| N4 | TransferFrom function unrestricted vulnerability | Design Logic Audit | Critical | Fixed |
| N5 | Hardcap not checked correctly vulnerability | Design Logic Audit | Low | Fixed |

# 4 Code Overview

## 4.1 Contracts Description

https://github.com/GoodEntry-io/geToken

commit: 2b42e202d524becbc3e543251bb623352bf09919

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:

| ERC20Vesting | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| vestingDuration | Public | - | - |
| totalVestingBalance | Public | - | - |
| vestingBalanceOf | Public | - | - |
| getVestingLength | Public | - | - |
| getVestingSchedule | Public | - | - |
| getVestingScheduleId | Public | - | - |
| getUserVestingSchedule | Public | - | - |
| _updateVestingDuration | Internal | Can Modify State | - |
| vest | Public | Can Modify State | - |
| _vest | Internal | Can Modify State | - |
| _withdraw | Internal | Can Modify State | - |
| vestingStatus | Public | - | - |
| _vestingStatus | Internal | - | - |
| transfer | Public | Can Modify State | - |

| Crowdsale | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| initSale | Public | Can Modify State | onlyOwner |
| pauseSale | Public | Can Modify State | onlyOwner |

| Crowdsale | | | |
|---|---|---|---|
| withdrawFunds | Public | Can Modify State | onlyOwner |
| deposit | Public | Can Modify State | - |
| claim | Public | Can Modify State | - |
| claimable | Public | - | - |

| esGOOD | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | ERC20 ERC20Permit ERC20Vesting |
| _afterTokenTransfer | Internal | Can Modify State | - |
| _mint | Internal | Can Modify State | - |
| _burn | Internal | Can Modify State | - |
| transfer | Public | Can Modify State | - |
| updateVestingDuration | Public | Can Modify State | onlyRole |
| updateRewardTracker | Public | Can Modify State | onlyRole |
| burn | Public | Can Modify State | onlyRole |
| mint | Public | Can Modify State | onlyRole |
| _beforeTokenTransfer | Internal | Can Modify State | - |
| withdraw | Public | Can Modify State | - |
| depositFor | Public | Can Modify State | - |

| ERC20PresetVestingExponential | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| _vestingStatus | Internal | - | - |

| GOOD | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | ERC20PresetMinterPauser ERC20Capped |
| _beforeTokenTransfer | Internal | Can Modify State | - |
| _mint | Internal | Can Modify State | - |
| burn | Public | Can Modify State | onlyRole |

| GoodAirdrop | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| claim | Public | Can Modify State | - |

| GoodGovernor | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | Governor GovernorSettings GovernorVotes GovernorVotesQuorumFraction GovernorTimelockControl |
| state | Public | - | - |
| propose | Public | Can Modify State | - |
| cancel | Public | Can Modify State | - |
| _execute | Internal | Can Modify State | - |
| _cancel | Internal | Can Modify State | - |
| _executor | Internal | - | - |
| supportsInterfac | Public | - | - |

| GoodGovernor | | | |
|---|---|---|---|
| e | | | |
| proposalThreshold | Public | - | - |

| RewardTracker | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| streamer | Public | - | - |
| balanceOf | External | - | - |
| rewardToken | Public | - | - |
| claimableRewardsOf | Public | - | - |
| setStreamer | External | Can Modify State | onlyRole |
| updateRewards | External | Can Modify State | - |
| claim | External | Can Modify State | - |
| _claim | Internal | Can Modify State | - |
| _afterClaimedRewards | Internal | Can Modify State | - |
| stake | External | Can Modify State | - |
| stake | Public | Can Modify State | - |
| _actionStake | Internal | Can Modify State | - |
| _updateStake | Internal | Can Modify State | - |
| unstake | External | Can Modify State | - |
| unstake | Public | Can Modify State | - |
| _actionUnstake | Internal | Can Modify State | - |
| _updateUnstake | Internal | Can Modify State | - |

| RewardTracker | | | |
|---|---|---|---|
| pendingRewards | Public | - | - |
| _updateRewards | Private | Can Modify State | - |

| RewardTrackerPresetMinter | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | RewardTracker |
| _afterClaimedRewards | Internal | Can Modify State | - |

| RewardTrackerPresetVirtualStake | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | RewardTracker |
| _actionStake | Internal | Can Modify State | - |
| _actionUnstake | Internal | Can Modify State | - |

| RewardStreamer | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| lifetimeRewards | Public | - | - |
| rewardToken | Public | - | - |
| rewardRate | Public | - | - |
| lastDistributionTime | Public | - | - |
| rewardTracker | Public | - | - |
| setRewardRate | External | Can Modify State | onlyRole |
| _updateRewards | Internal | Can Modify State | - |

| RewardStreamer | | | |
|---|---|---|---|
| pendingRewards | Public | - | - |
| _maxAvailableRewards | Internal | - | - |
| distribute | External | Can Modify State | - |
| _transferRewards | Internal | Can Modify State | - |

## 4.3 Vulnerability Summary

**[N1] [Medium] Risk of excessive authority**

**Category: Authority Control Vulnerability Audit**

**Content**

In the GOOD contract, the minter role can burn arbitrarily through the mint function.

Code location:

- GOOD.sol#32-34

```
  function burn(address account, uint256 amount) onlyRole(MINTER_ROLE) public {
    _burn(account, amount);
  }
```

In the GOOD contract and the ERC20PresetMinterPauser contract, the minter role can burn arbitrarily through the mint function.

Code location:

- GOOD.sol#28-30 ERC20PresetMinterPauser.sol#54-57

```
  function _mint(address account, uint256 amount) internal override (ERC20, ERC20Capped) {
    super._mint(account, amount);
  }

  function mint(address to, uint256 amount) public virtual {
    require(hasRole(MINTER_ROLE, _msgSender()),    "ERC20PresetMinterPauser: must have minter role to mint");
```

```
    _mint(to, amount);
    }
```

In the esGOOD contract, the minter role can mint arbitrarily through the mint function.

Code location:

- esGOOD.sol#58-63,100-103

```
 function _mint(address to, uint256 amount)
    internal
    override(ERC20, ERC20Votes)
  {
    super._mint(to, amount);
  }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
    require(goodToken.totalSupply() + totalSupply() <= goodToken.cap(), "esGOOD: Mint
excess GOOD");
    _mint(to, amount);
  }
```

In the esGOOD contract, the minter role can burn arbitrarily through the mint function.

Code location:

- esGOOD.sol#65-70,100-103

```
 function _burn(address account, uint256 amount)
    internal
    override(ERC20, ERC20Votes)
  {
    super._burn(account, amount);
  }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
    require(goodToken.totalSupply() + totalSupply() <= goodToken.cap(), "esGOOD: Mint
excess GOOD");
    _mint(to, amount);
  }
```

In the esGOOD contract and the ERC20Vesting contract, the admin role can change vesting duration through the updateVestingDuration function.

Code location:

- esGOOD.sol#79-82 ERC20Vesting#108-112

```
  function updateVestingDuration(uint64 _vestingDuration)
onlyRole(DEFAULT_ADMIN_ROLE) public
  {
    _updateVestingDuration(_vestingDuration);
  }

    function _updateVestingDuration(uint64 vestingDuration_) internal virtual {
    require(vestingDuration_ > 0, "ERC20Vesting: Invalid vesting duration");
    _vestingDuration = vestingDuration_;
    emit UpdatedVestingDuration(_vestingDuration);
  }
```

In the esGOOD contract, the admin role can withdraw on behalf of the user through the withdraw function.When the user's vesting duration has not expired, the admin role can force the withdrawal, causing user losses.

Code location:

- esGOOD.sol#122-127

```
  function withdraw(address account, uint256 userVestingId) public returns (uint256
received, uint256 penalty){
    require(msg.sender == account || hasRole(DEFAULT_ADMIN_ROLE, msg.sender),
"esGOOD: Unauthorized Withdrawal");
    (received, penalty) = _withdraw(account, userVestingId);
    _burn(account, received + penalty);
    goodToken.mint(account, received);
  }
```

In the Crowdsale contract,the owner can change tokensPerUSDC througth the initSale function during crowdsale.

code location:

- Crowdsale.sol#60-66

```
  function initSale(uint tokenAmount) public onlyOwner {
    require(block.timestamp < endDate, "Sale already ended");
```

```
    token.transferFrom(msg.sender, address(this), tokenAmount);
    crowdsaleTokenCap = token.balanceOf(address(this));
    tokensPerUSDC = crowdsaleTokenCap / hardcap;
    isPaused = false;
  }
```

**Solution**

1. It is recommended to transfer roles with excessive authority to community governance.
2. It is recommended to modify the burn function to only allow users to burn their own tokens, instead of the issuer burning any user tokens.
3. It is recommended to modify the withdraw function to only allow users to withdraw their own tokens, instead of the admin withdraw any user tokens.
4. It is recommended to modify the initSale function to limit the block.timestamp before startDate.

**Status**

Acknowledged

### [N2] [Critical] Duplicate claim and unchecked totalAirdropAmount vulnerability

**Category: Design Logic Audit**

**Content**

In the GoodAirdrop contract, the claim function does not check the value of the claimed mapping, allowing users to receive airdrops repeatedly. Additionally, without verifying the totalAirdropAmount against the claimed amount, there is a possibility that the airdrop amount could exceed the intended limit.

code location:

- GoodAirdrop.sol#40-49

```
  function claim(bytes32[] memory proof, uint amount) public returns (bool) {
    bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode(msg.sender,
amount))));
    require(MerkleProof.verify(proof, root, leaf), "Invalid proof");

    claimed[msg.sender] = true;
    esgood.mint(msg.sender, amount);
    claimedAmount += amount;
    emit Claimed(msg.sender, amount);
```

```
    return true;
  }
```

**Solution**

It is recommended to modify the claim function by adding a check for the claimed mapping value and checking the totalAirdropAmount value.

**Status**

Fixed

### [N3] [Critical] Transfer and transferFrom function misuse vulnerability

**Category: Design Logic Audit**

**Content**

In the Crowdsale contract,the transfer and transferFrom functions are called in the contract without checking whether their transfers are successful or not, and continue to execute the subsequent logic.

code location:

- Crowdsale.sol#60-66,74-80,89-99,104-111

```
  function initSale(uint tokenAmount) public onlyOwner {
    require(block.timestamp < endDate, "Sale already ended");
    token.transferFrom(msg.sender, address(this), tokenAmount);
    crowdsaleTokenCap = token.balanceOf(address(this));
    tokensPerUSDC = crowdsaleTokenCap / hardcap;
    isPaused = false;
  }

    function withdrawFunds() public onlyOwner  {
    require(block.timestamp > endDate, "TGE ongoing");
    USDCe.transfer(owner, USDCe.balanceOf(address(this)));
    uint amountBought = tokensPerUSDC * totalContributions;
    // return unsold tokens to admin for burn
    if (amountBought < crowdsaleTokenCap) token.transfer(msg.sender,
crowdsaleTokenCap - amountBought);
  }

    function deposit(uint amount) public returns (uint tokenAmount){
    require(!isPaused, "TGE paused");
    require(startDate < block.timestamp, "TGE hasnt started");
    require(block.timestamp <= endDate, "TGE has ended");
```

```
    require(totalContributions + amount < hardcap, "Hardcap reached");
    USDCe.transferFrom(msg.sender, address(this), amount);
    contributions[msg.sender] += amount;
    totalContributions += amount;
    tokenAmount = amount * tokensPerUSDC;
    emit TokensPurchased(msg.sender, amount);
  }

    function claim() public returns (uint tokenAmount) {
    require(block.timestamp > endDate, "TGE hasnt ended");
    require(!hasClaimed[msg.sender], "Already Claimed");
    hasClaimed[msg.sender] = true;
    tokenAmount = contributions[msg.sender] * tokensPerUSDC;
    token.transfer(msg.sender, tokenAmount);
    emit TokensClaimed(msg.sender, tokenAmount);
  }
```

**Solution**

It is recommended to use the safetransfer and safetransferfrom functions.

**Status**

Fixed

### [N4] [Critical] TransferFrom function unrestricted vulnerability

**Category: Design Logic Audit**

**Content**

In the ERC20Vesting contract, the transferFrom function inherits from ERC20.sol. However, the transferFrom function is not restricted, allowing users to transfer vested tokens through it.

code location:

- ERC20.sol#158-163

```
    function transferFrom(address from, address to, uint256 amount) public virtual
override returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender, amount);
        _transfer(from, to, amount);
        return true;
    }
```

**Solution**

It is recommended to modify the transferFrom function to limit the transfer of vested tokens.

**Status**

Fixed

### [N5] [Low] Hardcap not checked correctly vulnerability

**Category: Design Logic Audit**

**Content**

In the Crowdsale contract, the deposit function should ensure that the sum of totalContributions and amount is less than or equal to the value of hardcap.

code location:

- Crowdsale.sol#89-99

```
function deposit(uint amount) public returns (uint tokenAmount){
  require(!isPaused, "TGE paused");
  require(startDate < block.timestamp, "TGE hasnt started");
  require(block.timestamp <= endDate, "TGE has ended");
  require(totalContributions + amount < hardcap, "Hardcap reached");
  USDCe.transferFrom(msg.sender, address(this), amount);
  contributions[msg.sender] += amount;
  totalContributions += amount;
  tokenAmount = amount * tokensPerUSDC;
  emit TokensPurchased(msg.sender, amount);
}
```

**Solution**

It is recommended to modify the deposit function and check that the value of totalContributions + amount is less than or equal to the value of hardcap.

**Status**

Fixed

# 5 Audit Result

| Audit Number | Audit Team | Audit Date | Audit Result |
| --- | --- | --- | --- |
| 0X002309120001 | SlowMist Security Team | 2023.09.07 - 2023.09.12 | Medium Risk |

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 3 critical risk, 1 medium risk, 1 low risk vulnerabilities. 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