# Smart Contract Security Audit Report

## Table Of Contents

# 1 Executive Summary

On 2024.05.22, the SlowMist security team received the 9GAG team's security audit application for Memecoin Farming, 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

StakelandFarmClaim contracts are memecoin farming reward claim contracts. The contract sets up a tokenConfig to deposit tokens through the initDepositAndSetupTokenConfig function called by the admin role and issues a MerkleProof through the role. Users withdraw tokens with merkle proofs presented to the claim function before a configurable expiration time. The withdrawal can be done by a wallet that is delegated through delegate.xyz.

## 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 | Missing the correct validation | Design Logic Audit | Low | Fixed |
| N3 | Preemptive Initialization | Race Conditions Vulnerability | Suggestion | Acknowledged |
| N4 | Potential duplicate withdrawals | Design Logic Audit | Information | Acknowledged |
| N5 | External call reminder | Unsafe External Call Audit | Suggestion | Acknowledged |

# 4 Code Overview

## 4.1 Contracts Description

**Audit Version:**

https://github.com/9gag/memecoin-staking-audit/contracts/memecoin/farming

commit: ade496568b1b07ef7772fba66a48d3d302fba189

**Fixed Version:**

https://github.com/9gag/memecoin-staking-audit/contracts/memecoin/farming

commit: ac3c66dc0991e43e731d97bfb0557767c75b0f66

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:

| StakelandFarmClaim | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |

| StakelandFarmClaim | | | |
|---|---|---|---|
| <Constructor> | Public | Can Modify State | - |
| initialize | Public | Can Modify State | initializer |
| claim | External | Can Modify State | nonReentrant onlyClaimNotPaused onlyClaimNotPausedById |
| _getRequester | Private | - | - |
| _calculateClaim | Internal | - | - |
| _checkValidClaim | Internal | - | - |
| _verifyProof | Private | - | - |
| _checkValidSetup | Internal | - | - |
| initDepositAndSetupTokenConfig | External | Can Modify State | onlyOwner |
| depositToken | External | Can Modify State | onlyOwner onlyClaimPausedById |
| withdrawExpiredToken | External | Can Modify State | onlyOwner |
| setTokenMerkleRootById | External | Can Modify State | onlyOwner onlyClaimPausedById |
| setTokenClaimStartTsById | External | Can Modify State | onlyOwner onlyClaimPausedById |
| pauseTokenClaim | External | Can Modify State | onlyOwner |
| unpauseClaim | External | Can Modify State | onlyOwner |
| pauseClaim | External | Can Modify State | onlyOwner |
| unpauseClaim | External | Can Modify State | onlyOwner |
| getTokenConfigById | External | - | - |
| getTokenBalanceById | External | - | - |

| StakelandFarmClaim | | | |
|---|---|---|---|
| getTokensClaimedByUserAndId | External | - | - |
| getTokensClaimableByClaimInfo | External | - | - |

## 4.3 Vulnerability Summary

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

**Category: Authority Control Vulnerability Audit**

**Content**

1.In the StakelandFarmClaim contract, the owner role can modify the `merkleRoot` and the `claimStartTs` for a specific farmId, leading to the risk of over-privilege of the owner role.

Code location:

memecoin-staking-audit/contracts/memecoin/farming/StakelandFarmClaim.sol#239-262

```
    function setTokenMerkleRootById(uint16 farmId, bytes32 newMerkleRoot)
        external
        onlyOwner
        onlyClaimPausedById(farmId)
    {
        _tokenConfigs[farmId].merkleRoot = newMerkleRoot;

        emit TokenMerkleRootUpdated(farmId, newMerkleRoot);
    }

    function setTokenClaimStartTsById(uint16 farmId, uint40 newTokenClaimStartTs)
        external
        onlyOwner
        onlyClaimPausedById(farmId)
    {
        _tokenConfigs[farmId].claimStartTs = newTokenClaimStartTs;

        emit TokenClaimStartTsUpdated(farmId, newTokenClaimStartTs);
    }
```

2.The contract inherits the UUPSUpgrader contract and in this UUPSUpgrader contract, the owner role can set the upgrader role and upgrade the contract through the upgrader role.

3.If a user approves the `tokenConfig.token` to the contract, the owner role can call the depositToken function to deposit the token to this contract, or the owner role can call the initDepositAndSetupTokenConfig function to deposit the token to the contract when there is a new token config.

Code location:

memecoin-staking-audit/contracts/memecoin/farming/StakelandFarmClaim.sol#179-212

```
    function initDepositAndSetupTokenConfig(
        address depositor,
        uint16 farmId,
        uint256 amount,
        TokenConfig calldata tokenConfig
    ) external onlyOwner {
        ...
        IERC20(tokenConfig.token).safeTransferFrom(depositor, address(this), amount);
        _tokenConfigs[farmId] = tokenConfig;
        ...
    }

    function depositToken(address depositor, uint16 farmId, uint256 amount)
        external
        onlyOwner
        onlyClaimPausedById(farmId)
    {
        ...
        IERC20(tokenConfig.token).safeTransferFrom(depositor, address(this), amount);
    }
```

**Solution**

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

**Status**

Acknowledged; After communicating with the project team, they expressed that they plan on using a multi-sig as the contract owner, and roles with separate permissions and responsibilities will also be set up.

### [N2] [Low] Missing the correct validation

**Category: Design Logic Audit**

**Content**

1.When the owner role sets the tokenConfig in the initDepositAndSetupTokenConfig function or modifies the claimStartTs in the setTokenClaimStartTsById function, the internal function _checkValidSetup checks the claimStartTs can not be 0, and the setTokenClaimStartTsById does not check the claimStartTs.

Code location:

memecoin-staking-audit/contracts/memecoin/farming/StakelandFarmClaim.sol#155-167 ,254-262

```
    function _checkValidSetup(address depositor, uint16 farmId, uint256 amount,
TokenConfig calldata tokenConfig)
        internal
        pure
    {
        // this contract only records farmId starting from season 2, so no 0 and 1
farmId
        if (
            depositor == address(0) || farmId == 0 || farmId == 1 || amount == 0 ||
tokenConfig.token == address(0)
                || tokenConfig.initialUnlockBP > _BASIS_POINTS ||
tokenConfig.expiryInDays == 0
                || tokenConfig.claimStartTs == 0 || tokenConfig.merkleRoot.length ==
0
        ) {
            revert InvalidSetup();
        }
    }

    function setTokenClaimStartTsById(uint16 farmId, uint40 newTokenClaimStartTs)
        external
        onlyOwner
        onlyClaimPausedById(farmId)
    {
        _tokenConfigs[farmId].claimStartTs = newTokenClaimStartTs;

        emit TokenClaimStartTsUpdated(farmId, newTokenClaimStartTs);
    }
```

2.In the withdrawExpiredToken function, the owner role can withdraw the unclaimed token from the contract after its claim period has expired. But the expiry time only checks the claimStartTs add the expiry days. This check does not contain the `vestDurationInDays` and it is not the same as the calculation of the `expiryTs`.

And the calculation in the withdrawExpiredToken function, the `tokenConfig.claimStartTs + tokenConfig.expiryInDays * 1 days` time is in the claim time and does not reach the `expiryTs`.

Code location:

memecoin-staking-audit/contracts/memecoin/farming/StakelandFarmClaim.sol#223-225

```
    function withdrawExpiredToken(uint16 farmId, address receiver) external onlyOwner
 {
        ...
        if (block.timestamp <= tokenConfig.claimStartTs + tokenConfig.expiryInDays *
 1 days) {
            revert TokenClaimNotExpired();
        }
        ...
    }
```

**Solution**

1. It's recommended to check the claimStartTs whether is larger than the block.timestamp(now). And when the vesting begins, it is recommended not to modify the claimStartTs after the vesting ends.
2. It's recommended to check the claim expiry time as `expiryTs`.

**Status**

Fixed

### [N3] [Suggestion] Preemptive Initialization

**Category: Race Conditions Vulnerability**

**Content**

By calling the initialize and deploy functions to initialize the contracts, there is a potential issue that malicious attackers preemptively call the initialize function to initialize.

Code location:

memecoin-staking-audit/contracts/memecoin/farming/StakelandFarmClaim.sol#52-57

```
    function initialize() public initializer {
        UUPSUpgrader.__UUPSUpgrader_init();
        ReentrancyGuardUpgradeable.__ReentrancyGuard_init();
        OwnableUpgradeable.__Ownable_init(_msgSender());
```

```
            _dcV2 = IDelegateRegistry(0x00000000000000447e69651d841bD8D104Bed493);
        }
```

**Solution**

It is suggested that the initialization operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

**Status**

Acknowledged

### [N4] [Information] Potential duplicate withdrawals

**Category: Design Logic Audit**

**Content**

In the contract, the user receives the reward through the claim function. The claim can be msg.sender or the vault. Each time the user claims the reward, it will be recorded in _usersTokensClaimedById, and its mapping includes farmId and user address. The verification is to verify the user proof and merkleRoot through the _verifyProof function. The verification has a replay write-up code. Users issued by the centralized merkleRoot can call it repeatedly to pass the _verifyProof verification. However, there is a _usersTokensClaimedById reward record for a single user. But there is no way to obtain rewards through this verification after receiving all `totalAllocated`. However, if the requester address permitted and delegated by this user is also issued during the merkleRoot issuance process. And in the case of the current tokenConfig.token still exists in the contract, the reward tokens can be received here for their requester address.

Code location:

memecoin-staking-audit/contracts/memecoin/farming/StakelandFarmClaim.sol#65-83, 147-153

```
        function claim(address vault, TokenClaim calldata tokenClaim)
            external
            nonReentrant
            onlyClaimNotPaused
            onlyClaimNotPausedById(tokenClaim.farmId)
        {
            address requester = _getRequester(vault);

            TokenConfig memory tokenConfig = _tokenConfigs[tokenClaim.farmId];
            _checkValidClaim(requester, tokenConfig, tokenClaim);
            uint256 amount = _calculateClaim(requester, tokenConfig, tokenClaim.farmId,
    tokenClaim.totalAllocated);
            if (amount == 0) revert ZeroClaimAmount();

            _usersTokensClaimedById[requester][tokenClaim.farmId] += amount;
            IERC20(tokenConfig.token).safeTransfer(requester, amount);
            ...
        }

        function _checkValidClaim(address user, TokenConfig memory tokenConfig,
    TokenClaim calldata tokenClaim)
            internal
            view
        {
            ...
            if (!_verifyProof(user, tokenClaim)) revert InvalidProof();
        }

        function _verifyProof(address user, TokenClaim calldata tokenClaim) private view
    returns (bool) {
            return MerkleProof.verifyCalldata(
                tokenClaim.proof,
                _tokenConfigs[tokenClaim.farmId].merkleRoot,
                keccak256(bytes.concat(keccak256(abi.encode(user, tokenClaim.farmId,
    tokenClaim.totalAllocated))))
            );
        }
```

**Solution**

It is recommended that the project team confirm whether the code conforms to the actual business design, and pay attention to the accuracy of the centralized issuance of merkleRoot.

**Status**

Acknowledged; After communicating with the project team, they expressed that the code conforms to the actual business design as all Merkle proofs are initially only generated once with all eligible wallet addresses for every new farming campaign. Wallet who acts as a delegator will not be able to claim using the same proof from delegatee, thus no duplicate withdrawal even when they are “undelegated”. As long as the claimer is not the eligible address they used to generate from in the first place, it will not pass _verifyProof check.

### [N5] [Suggestion] External call reminder

**Category: Unsafe External Call Audit**

**Content**

In the claim function, the user can authorize a third-party contract to delegate an acceptance address as a token requester, but this validation contract is not in the scope of the audit. The legitimacy of this contract and the security of the calling logic need to be confirmed.

Code location:

memecoin-staking-audit/contracts/memecoin/farming/StakelandFarmClaim.sol#95

```
    bool isDelegateValid = _dcV2.checkDelegateForAll(_msgSender(), vault, "");
```

**Solution**

It is recommended to clarify whether this external call contract is credible and check the validity of the incoming resolver address and data.

**Status**

Acknowledged; After communicating with the project team, they expressed that they have made sure that the external call to the third party DelegateRegistry contract is working as intended.

## 5 Audit Result

| Audit Number | Audit Team | Audit Date | Audit Result |
|---|---|---|---|
| 0X002405240001 | SlowMist Security Team | 2024.05.22 - 2024.05.24 | Medium Risk |

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 medium risk, 1 low risk, 2 suggestions, and 1 information. And 1 low risk was fixed. 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.

SLOWMIST

**Official Website**
www.slowmist.com

**E-mail**
team@slowmist.com

**Twitter**
@SlowMist_Team

**Github**
https://github.com/slowmist