# Smart Contract Security Audit Report

# Table Of Contents

# 1 Executive Summary

On 2025.10.29, the SlowMist security team received the Sigma Money team's security audit application for Sigma DAO round 3&4&5, 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

Sigma DAO protocol is forked from Shadow Protocol.

## 3.2 Vulnerability Information

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

| NO | Title | Category | Level | Status |
| --- | --- | --- | --- | --- |
| N1 | Cross-chain replay attacks | Replay Vulnerability | Suggestion | Acknowledged |
| N2 | Risk of excessive authority | Authority Control Vulnerability Audit | Medium | Acknowledged |

| NO | Title | Category | Level | Status |
| --- | --- | --- | --- | --- |
| N3 | Risk of excess preallocation amount | Design Logic Audit | Medium | Fixed |
| N4 | Risk of hash collision | Others | Suggestion | Fixed |

# 4 Code Overview

## 4.1 Contracts Description

https://github.com/SigmaMoney/dao/tree/feat/bsc

**Round 3&4:**

**Initial audit version:** ffb097a69b5338d9ffe19adc09aa798803d59b5d

**Final audit version**: 98590ecf15d6ace36bb88e8943097ef8a73c732e

**Audit Scope:**

```
- contracts/Voter.sol
- contracts/AccessHub.sol
- contracts/interfaces/IAccessHub.sol
- contracts/interfaces/IVoter.sol
- contracts/SigmaAirdrop.sol
```

**Round 5:**

**Initial audit version:** 11907f131ba65a50bb49ff5b24bf1d5aef54f640

**Final audit version**: 98590ecf15d6ace36bb88e8943097ef8a73c732e

**Audit Scope:**

```
- contracts/xShadow/x33.sol
```

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:

| SigmaAirdrop | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| claim | External | Can Modify State | nonReentrant |
| batchClaim | External | Can Modify State | nonReentrant |
| _processClaim | Internal | Can Modify State | - |
| emergencyWithdraw | External | Can Modify State | onlyGovernance |
| isSaltUsed | External | - | - |
| getMessageHash | External | - | - |
| getEthSignedMessageHash | External | - | - |

| Voter | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| constructor | Public | Can Modify State | - |
| initialize | External | Can Modify State | initializer |
| setGlobalRatio | External | Can Modify State | onlyGovernance |
| reset | External | Can Modify State | - |
| _reset | Internal | Can Modify State | - |
| poke | External | Can Modify State | - |
| vote | External | Can Modify State | - |
| _vote | Internal | Can Modify State | - |
| _distribute | Internal | Can Modify State | - |
| getVotes | External | - | - |

| Voter | | | |
|---|---|---|---|
| setGovernor | External | Can Modify State | onlyGovernance |
| whitelist | Public | Can Modify State | onlyGovernance |
| revokeWhitelist | Public | Can Modify State | onlyGovernance |
| killGauge | Public | Can Modify State | onlyGovernance |
| reviveGauge | Public | Can Modify State | onlyGovernance |
| stuckEmissionsRecovery | External | Can Modify State | onlyGovernance |
| whitelistGaugeRewards | External | Can Modify State | onlyGovernance |
| removeGaugeRewardWhitelist | External | Can Modify State | onlyGovernance |
| removeFeeDistributorReward | External | Can Modify State | onlyGovernance |
| getPeriod | Public | - | - |
| createSigmaGauge | External | Can Modify State | onlyGovernance |
| createVeFunderGauge | External | Can Modify State | onlyGovernance |
| setSigmaGaugePreallocation | External | Can Modify State | onlyGovernance |
| claimIncentives | External | Can Modify State | - |
| claimRewards | External | Can Modify State | - |
| claimLegacyRewardsAndExit | External | Can Modify State | - |
| notifyRewardAmount | External | Can Modify State | - |
| distribute | Public | Can Modify State | nonReentrant |
| distributeForPeriod | Public | Can Modify State | nonReentrant |
| distributeAll | External | Can Modify State | - |
| batchDistributeByIndex | External | Can Modify State | - |
| getAllGauges | External | - | - |

| Voter | | | |
|---|---|---|---|
| getAllFeeDistributors | External | - | - |
| isGauge | External | - | - |
| isFeeDistributor | External | - | - |
| _claimablePerPeriod | Internal | - | - |
| _getTotalPreallocationForPeriod | Internal | - | - |
| _getCurrentTotalGaugePreallocation | Internal | - | - |

| AccessHub | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| initialize | External | Can Modify State | initializer |
| reinit | External | Can Modify State | timelocked |
| initializeVoter | External | Can Modify State | timelocked |
| addVestingSchedule | External | Can Modify State | onlyRole |
| removeVestingSchedule | External | Can Modify State | onlyRole |
| startRebase | External | Can Modify State | onlyRole |
| setNewGovernorInVoter | External | Can Modify State | onlyRole |
| createSigmaGauge | External | Can Modify State | onlyRole |
| createVeFunderGauge | External | Can Modify State | onlyRole |
| governanceWhitelist | External | Can Modify State | onlyRole |
| killGauge | External | Can Modify State | onlyRole |
| reviveGauge | External | Can Modify State | onlyRole |
| setEmissionsRatioInVoter | External | Can Modify State | onlyRole |

| AccessHub | | | |
|---|---|---|---|
| retrieveStuckEmissionsToGovernance | External | Can Modify State | onlyRole |
| setGaugePreallocation | External | Can Modify State | onlyRole |
| createCLGaugeOveridden | External | Can Modify State | onlyRole |
| transferWhitelistInXShadow | External | Can Modify State | onlyRole |
| toggleXShadowGovernance | External | Can Modify State | onlyRole |
| operatorRedeemXShadow | External | Can Modify State | onlyRole |
| migrateOperator | External | Can Modify State | onlyRole |
| rescueTrappedTokens | External | Can Modify State | onlyRole |
| setExemptionToInXShadow | External | Can Modify State | onlyRole |
| setEmissionsMultiplierInMinter | External | Can Modify State | onlyRole |
| setGaugeActiveInMinter | External | Can Modify State | onlyRole |
| augmentGaugeRewardsForPair | External | Can Modify State | onlyRole |
| removeFeeDistributorRewards | External | Can Modify State | onlyRole |
| setCooldownExemption | External | Can Modify State | timelocked |
| setNewRebaseStreamingDuration | External | Can Modify State | timelocked |
| setNewVoteModuleCooldown | External | Can Modify State | timelocked |
| kickInactive | External | Can Modify State | onlyRole |
| execute | External | Can Modify State | timelocked |
| setNewTimelock | External | Can Modify State | timelocked |

| x33 | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | ERC20 ERC4626 |

| x33 | | | |
|---|---|---|---|
| submitVotes | External | Can Modify State | onlyOperator |
| compound | External | Can Modify State | onlyOperator |
| claimRebase | External | Can Modify State | onlyOperator |
| claimIncentives | External | Can Modify State | onlyOperator |
| swapIncentiveViaAggregator | External | Can Modify State | nonReentrant onlyOperator |
| rescue | External | Can Modify State | nonReentrant onlyAccessHub |
| unlock | External | Can Modify State | onlyOperator |
| transferOperator | External | Can Modify State | onlyAccessHub |
| whitelistAggregator | External | Can Modify State | onlyAccessHub |
| totalAssets | Public | - | - |
| ratio | Public | - | - |
| getPeriod | Public | - | - |
| isUnlocked | Public | - | - |
| isCooldownActive | Public | - | - |
| _deposit | Internal | Can Modify State | whileNotLocked |
| _withdraw | Internal | Can Modify State | - |

## 4.3 Vulnerability Summary

**[N1] [Suggestion] Cross-chain replay attacks**

**Category: Replay Vulnerability**

**Content**

In the SigmaAirdrop contract, the `claim` and `_processClaim` functions do not include `block.chainid` in the hash construction of the signed message. This means that if the contract is deployed on multiple blockchains, an

attacker can obtain a valid signature generated on one chain and then repeatedly submit the signature to the same contract instance on other chains to claim the airdrop.

- contracts/SigmaAirdrop.sol#L77-L106, L137-L166

```
    function claim(
        bytes calldata _signature,
        uint256 _amount,
        address _recipient,
        string calldata _salt
    ) external nonReentrant {
        if (_amount == 0) revert ZeroAmount();
        if (_recipient == address(0)) revert ZeroAddress();

        bytes32 saltHash = keccak256(bytes(_salt));
        if (usedSalts[saltHash]) revert SaltAlreadyUsed();

        // Construct the message hash (order: address, uint256, string)
        bytes32 messageHash = keccak256(abi.encodePacked(_recipient, _amount, _salt));
        bytes32 ethSignedMessageHash = messageHash.toEthSignedMessageHash();

        // Verify signature
        address recoveredSigner = ethSignedMessageHash.recover(_signature);
        if (recoveredSigner != signer) {
            revert InvalidSignature();
        }

        // Mark salt as used
        usedSalts[saltHash] = true;

        // Transfer tokens
        IERC20(sigma).safeTransfer(_recipient, _amount);

        emit AirdropClaimed(_recipient, _amount, saltHash);
    }

    function _processClaim(
        bytes calldata _signature,
        uint256 _amount,
        address _recipient,
        string calldata _salt
    ) internal {
        if (_amount == 0) revert ZeroAmount();
        if (_recipient == address(0)) revert ZeroAddress();

        bytes32 saltHash = keccak256(bytes(_salt));
        if (usedSalts[saltHash]) revert SaltAlreadyUsed();
```

```
        // Construct the message hash (order: address, uint256, string)
        bytes32 messageHash = keccak256(abi.encodePacked(_recipient, _amount, _salt));
        bytes32 ethSignedMessageHash = messageHash.toEthSignedMessageHash();

        // Verify signature
        address recoveredSigner = ethSignedMessageHash.recover(_signature);
        if (recoveredSigner != signer) {
            revert InvalidSignature();
        }

        // Mark salt as used
        usedSalts[saltHash] = true;

        // Transfer tokens
        IERC20(sigma).safeTransfer(_recipient, _amount);

        emit AirdropClaimed(_recipient, _amount, saltHash);
    }
```

**Solution**

It is recommended to modify the message hash construction to include the chainid.

**Status**

Acknowledged; The project team stated that each deployed SigmaAirdrop contract will use a unique signer and will not be deployed on multiple chains.

### [N2] [Medium] Risk of excessive authority

**Category: Authority Control Vulnerability Audit**

**Content**

1.In the AccessHub contract, the `DEFAULT_ADMIN_ROLE` role can grant and revoke all other roles and has the `kickInactive` function permission to kick out inactive users who have not voted and reset their voting status.

- dao/contracts/AccessHub.sol#L382-L404

```
    function kickInactive(
        address[] calldata _nonparticipants
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {}
```

2.In the AccessHub contract, the PROTOCOL_OPERATOR role has operational management permissions, including: managing the addition and removal of SigmaVesting, setting the governor of the Voter contract, creating and managing SigmaGauge and VeFunderGauge, whitelist management (token whitelist, reward whitelist), pause/unpause Gauge, setting emission ratios and pre-allocation, controlling xShadow's transfer whitelist and pause/unpause functions, operator redemption and migration, adjusting Minter's emission multiples, and switching Gauge token emission status, among other core functions.

- dao/contracts/AccessHub.sol##L127-L134, L137-L139, L141-L143, L148-L153, L155-L160, L162-L169, L172-L189, L192-L202, L205-L213, L216-L220, L223-L228, L231-L236, L238-L244, L249-L256, L259-L264, L267-L271, L274-L278, L281-L286, L289-L296, L301-L305, L307-L311, L316-L340, L342-L353

```
    function addVestingSchedule(
        address _beneficiary,
        address _tokenAddress,
        uint8 _category,
        ISigmaVesting.UnlockEntry[] calldata _entries
    ) external onlyRole(PROTOCOL_OPERATOR) {}

    function removeVestingSchedule(address _beneficiary, address _tokenAddress)
external onlyRole(PROTOCOL_OPERATOR) {}

    function startRebase(address _voteModule, address _voter) external
onlyRole(PROTOCOL_OPERATOR) {}

    function setNewGovernorInVoter(address _newGovernor) external
onlyRole(PROTOCOL_OPERATOR) {}

    function createSigmaGauge(address _pool, uint256 _preallocationBps) external
onlyRole(PROTOCOL_OPERATOR) {}

    function createVeFunderGauge(
        address _receiver,
        uint256 _maxEmission,
        address _pool,
        uint256 _preallocationBps
    ) external onlyRole(PROTOCOL_OPERATOR) { }

    function governanceWhitelist(address[] calldata _token, bool[] calldata
_whitelisted) external onlyRole(PROTOCOL_OPERATOR) {}

    function killGauge(address[] calldata _pairs) external
onlyRole(PROTOCOL_OPERATOR) {}
```

```
    function reviveGauge(address[] calldata _pairs) external
onlyRole(PROTOCOL_OPERATOR) {}

    function setEmissionsRatioInVoter(uint256 _pct) external
onlyRole(PROTOCOL_OPERATOR) {}

    function retrieveStuckEmissionsToGovernance(address _gauge, uint256 _period)
external onlyRole(PROTOCOL_OPERATOR) {}

    function setSigmaGaugePreallocation(address _gauge, uint256 _preallocationBps)
external onlyRole(PROTOCOL_OPERATOR) {}

    function createCLGaugeOveridden(address tokenA, address tokenB, int24
tickSpacing) external onlyRole(PROTOCOL_OPERATOR) {}

    function transferWhitelistInXShadow(address[] calldata _who, bool[] calldata
_whitelisted) external onlyRole(PROTOCOL_OPERATOR) {}

    function toggleXShadowGovernance(bool enable) external
onlyRole(PROTOCOL_OPERATOR) {}

    function operatorRedeemXShadow(uint256 _amount) external
onlyRole(PROTOCOL_OPERATOR) {}

    function migrateOperator(address _operator) external onlyRole(PROTOCOL_OPERATOR)
{}

    function rescueTrappedTokens(address[] calldata _tokens, uint256[] calldata
_amounts) external onlyRole(PROTOCOL_OPERATOR) {}

    function setExemptionToInXShadow(address[] calldata _who, bool[] calldata
_whitelisted) external onlyRole(PROTOCOL_OPERATOR) {}

    function setEmissionsMultiplierInMinter(uint256 _multiplier) external
onlyRole(PROTOCOL_OPERATOR) {}

    function setGaugeActiveInMinter(bool _isGaugeActive) external
onlyRole(PROTOCOL_OPERATOR) {}

    function augmentGaugeRewardsForPair(address[] calldata _pools, address[] calldata
_rewards, bool[] calldata _addReward) external onlyRole(PROTOCOL_OPERATOR) {}

    function removeFeeDistributorRewards(address[] calldata _pools, address[]
calldata _rewards) external onlyRole(PROTOCOL_OPERATOR) {}
```

**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. And the authority involving user funds should be managed by the community, and the EOA address can manage the authority involving emergency contract suspension. This ensures both a quick response to threats and the safety of user funds.

**Status**

Acknowledged

### [N3] [Medium] Risk of excess preallocation amount

**Category: Design Logic Audit**

**Content**

In the Voter contract, when the `killGauge` function kills the Gauge, the `preallocationBps` in its `gaugePreallocation` mapping is not cleared. When the `reviveGauge` function revives the Gauge, it does not check whether the recovered `preallocationBps` exceeds the total limit. This may cause the actual effective preallocation amount to exceed the limit of `maxTotalSigmaGaugePreallocation`.

- dao/contracts/Voter.sol#L471-L507, L509-L520

```
    function killGauge(address _gauge) public onlyGovernance {
        /// @dev ensure the gauge is alive already, and exists
        require(
            isAlive[_gauge] && gauges.contains(_gauge),
            GAUGE_INACTIVE(_gauge)
        );
        /// @dev set the gauge to dead
        isAlive[_gauge] = false;
        address pool = poolForGauge[_gauge];
        /// @dev fetch the last distribution
        uint256 _lastDistro = lastDistro[_gauge];
        /// @dev fetch the current period
        uint256 currentPeriod = getPeriod();
        /// @dev placeholder
        uint256 _claimable;
        /// @dev loop through the last distribution period up to and including the
current period
        for (uint256 period = _lastDistro; period <= currentPeriod; ++period) {
```

```
            /// @dev if the gauge isn't distributed for the period
            if (!gaugePeriodDistributed[_gauge][period]) {
                uint256 additionalClaimable = _claimablePerPeriod(pool, period);
                _claimable += additionalClaimable;

                /// @dev prevent gaugePeriodDistributed being marked true when the
minter hasn't updated yet
                if (additionalClaimable > 0) {
                    gaugePeriodDistributed[_gauge][period] = true;
                }
            }
        }
        /// @dev if there is anything claimable left
        if (_claimable > 0) {
            /// @dev send to the governor contract
            IERC20(shadow).transfer(governor, _claimable);
        }
        /// @dev update last distribution to the current period
        lastDistro[_gauge] = currentPeriod;
        emit GaugeKilled(_gauge);
    }

    function reviveGauge(address _gauge) public onlyGovernance {
        /// @dev ensure the gauge is dead and exists
        require(
            !isAlive[_gauge] && gauges.contains(_gauge),
            ACTIVE_GAUGE(_gauge)
        );
        /// @dev set the gauge to alive
        isAlive[_gauge] = true;
        /// @dev update last distribution to the current period
        lastDistro[_gauge] = getPeriod();
        emit GaugeRevived(_gauge);
    }
```

**Solution**

It is recommended to clear the preallocation amount of SigmaGauge in the killGauge function and reallocate the preallocation amount of SigmaGauge in the reviveGauge function.

**Status**

Fixed

### [N4] [Suggestion] Risk of hash collision

**Category: Others**

**Content**

In the SigmaAirdrop contract, the functions `claim`, `_processClaim`, `getMessageHash`, and `getEthSignedMessageHash` use `abi.encodePacked(_recipient, _amount, _salt)` when constructing the signed message. Since the parameter contains a dynamically long string type (`_salt`), it may cause hash collisions in some extreme cases.

- contracts/SigmaAirdrop.sol#L77-L106, L137-L166, L207-L213, L222-L229

```
function claim(
    bytes calldata _signature,
    uint256 _amount,
    address _recipient,
    string calldata _salt
) external nonReentrant {
    //...
    bytes32 messageHash = keccak256(abi.encodePacked(_recipient, _amount, _salt));
    //...
}

function _processClaim(
    bytes calldata _signature,
    uint256 _amount,
    address _recipient,
    string calldata _salt
) internal {
    //...
    bytes32 messageHash = keccak256(abi.encodePacked(_recipient, _amount, _salt));
    //...
}

function getMessageHash(
    address _recipient,
    uint256 _amount,
    string calldata _salt
) external pure returns (bytes32) {
    return keccak256(abi.encodePacked(_recipient, _amount, _salt));
}

function getEthSignedMessageHash(
    address _recipient,
    uint256 _amount,
    string calldata _salt
) external pure returns (bytes32) {
```

```
        bytes32 messageHash = keccak256(abi.encodePacked(_recipient, _amount, _salt));
        return messageHash.toEthSignedMessageHash();
    }
```

**Solution**

It is recommended to use abi.encode instead of abi.encodePacked.

**Status**

Fixed

# 5 Audit Result

| Audit Number | Audit Team | Audit Date | Audit Result |
| --- | --- | --- | --- |
| 0X002510290001 | SlowMist Security Team | 2025.10.29 - 2025.10.29 | 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 medium risk, 1 suggestion.

# 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