Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.04.10, the SlowMist security team received the FLock team's security audit application for Flock v2, 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 an audit of the V2 version of the FLock protocol, mainly covering the gmFlock token, the exchange between gmFlock and Flock tokens, gmFlock token staking, task management, and pool management contracts.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	The contract storage structure has changed	Scoping and Declarations Audit	Information	Acknowledged

NO	Title	Category	Level	Status
N2	Redundant owner inheritance	Others	Suggestion	Acknowledged
N3	Unsafe integer conversion	Integer Overflow and Underflow Vulnerability	Suggestion	Fixed
N4	Not following the Checks-Effects-Interactions (CEI) specification	Reentrancy Vulnerability	Suggestion	Fixed
N5	Front-running Risk in Reward Distribution	Reordering Vulnerability	Medium	Fixed
N6	totalAssetValue increases from different sources	Others	Information	Fixed
N7	Optimizable mintDailyReward operation	Design Logic Audit	Low	Fixed
N8	Shadow variables	Scoping and Declarations Audit	Medium	Fixed
N9	Necessary variable checks	Design Logic Audit	Suggestion	Fixed
N10	Potential DoS risk for token swapping via admin	Denial of Service Vulnerability	Information	Acknowledged
N11	Potential risk of premature termination of the claimUnlockedFlockBylds operation	Design Logic Audit	Suggestion	Acknowledged
N12	Potential DoS risk of token unlocking	Denial of Service Vulnerability	Low	Fixed
N13	The risks of excessive privilege	Authority Control Vulnerability Audit	Medium	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/FLock-io/co-hosting-smart-contracts/tree/v2-staging/contracts/v2

commit: e61cb1782e141109626e59db2a59c887f81fcfc4

Fixed Version:

https://github.com/FLock-io/co-hosting-smart-contracts

commit: eac332c26642504ee7c6c2394cca953210d9a1e6

Audit Scope:

```
./contracts/v2
├── FlockMiniPoolV2.sol
├── FlockPoolManagerV2Upgradeable.sol
├── FlockStakeInfoV2Upgradeable.sol
├── FlockTaskManagerV2Upgradeable.sol
├── RbacUpgradeable.sol
├── config
│   ├── ConfigHelperV2.sol
│   ├── ConfigOptionsV2.sol
│   └── FlockConfigV2.sol
├── gmFlockExchangeUpgradeable.sol
└── gmFlockUpgradeable.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:

RbacUpgradeable			
Function Name	Visibility	Mutability	Modifiers
__RbacUpgradeable_init	Internal	Can Modify State	onlyInitializing
addDaoAdmin	External	Can Modify State	onlyDaoAdmin

RbacUpgradeable			
addProtocolManager	External	Can Modify State	onlyDaoAdmin
addRewardOperator	External	Can Modify State	onlyDaoAdmin
renounceDaoAdmin	External	Can Modify State	-
renounceProtocolManager	External	Can Modify State	-
renounceRewardOperator	External	Can Modify State	-

gmFlockUpgradeable			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
addBatchToBlacklist	External	Can Modify State	onlyDaoAdmin
removeBatchFromBlacklist	External	Can Modify State	onlyDaoAdmin
setTransferRestriction	External	Can Modify State	onlyDaoAdmin
setRegistered	External	Can Modify State	onlyDaoAdmin
mint	External	Can Modify State	onlyDaoAdmin
burn	External	Can Modify State	onlyDaoAdmin
transfer	Public	Can Modify State	notBlacklisted
transferFrom	Public	Can Modify State	notBlacklisted
burnNetworkFees	Public	Can Modify State	onlyDaoAdmin
transferERC20	Public	Can Modify State	onlyDaoAdmin
_beforeTokenTransfer	Internal	Can Modify State	-

gmFlockExchangeUpgradeable			
Function Name	Visibility	Mutability	Modifiers

gmFlockExchangeUpgradeable			
initialize	Public	Can Modify State	initializer
setFlockConfig	External	Can Modify State	onlyAdmin
setExchangeParams	External	Can Modify State	onlyAdmin
getExchangeMultiplier	Public	-	-
exchangeFlock	External	Can Modify State	-
claimUnlockedFlock	External	Can Modify State	-
claimUnlockedFlockByIds	External	Can Modify State	-
findLockIndex	Internal	-	-
getUnlockedAmounts	External	-	-
getUserLocks	External	-	-

FlockConfigV2			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
setAddress	Public	Can Modify State	onlyAdmin
copyFromOtherConfig	External	Can Modify State	onlyAdmin
getAddress	External	-	-

FlockMiniPoolV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
delegate	External	Can Modify State	-
undelegate	External	Can Modify State	-

FlockMiniPoolV2			
claimRewards	External	Can Modify State	-
fundPool	External	Can Modify State	-
delegationOf	Public	-	-
getTotalDelegationAmount	External	-	-
claimable	Public	-	-
getDelegators	External	-	-
interestPerYear	External	-	-
totalRewardForDelegator	External	-	-
assetValue	Internal	-	-

FlockPoolManagerV2Upgradeable			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
setMinSigma	External	Can Modify State	onlyProtocolManager
setMaxSigma	External	Can Modify State	onlyProtocolManager
setProtocolFeePercentage	External	Can Modify State	onlyProtocolManager
setSigmaModificationCooldown	External	Can Modify State	onlyProtocolManager
setConfig	External	Can Modify State	onlyDaoAdmin
setDelegatorCoolDown	External	Can Modify State	onlyProtocolManager
getMiniPool	External	-	-
getPoolSigma	External	-	-
isPool	Public	-	-
getPercentageBase	External	-	-

FlockPoolManagerV2Upgradeable			
getProtocolFeePercentage	External	-	-
getPools	External	-	-
getUsers	External	-	-
getUserAt	External	-	-
getUsersLength	External	-	-
getDelegatorPools	External	-	-
getDelegationCoolDown	External	-	-
createMiniPool	External	Can Modify State	-
setMiniPoolSigma	External	Can Modify State	-
addDelegatorToPool	External	Can Modify State	-

FlockStakeInfoV2Upgradeable			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
setConfig	External	Can Modify State	onlyDaoAdmin
setMaxDelegationWeight	External	Can Modify State	onlyDaoAdmin
addStakes	External	Can Modify State	-
removeStakes	External	Can Modify State	-
clearStakes	External	Can Modify State	-
retallyDelegation	External	Can Modify State	-
getDelegationTokenAmountForUser	Public	-	-
getTotalUserStakeOverAllActiveTasks	Public	-	-
getTotalActiveTaskStakes	External	-	-

FlockStakeInfoV2Upgradeable			
getTaskStakes	External	-	-
getTotalActiveTaskWeights	External	-	-
getTaskWeights	External	-	-
getTotalNodeStakesPerTask	External	-	-
getTotalValidatorStakesPerTask	External	-	-
retallyDelegationInternal	Internal	Can Modify State	-

FlockTaskManagerV2Upgradeable			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
setBaseStakeAmount	External	Can Modify State	onlyProtocolManager
setRewardTokenPerDay	External	Can Modify State	onlyProtocolManager
setLockedRewardPercentage	External	Can Modify State	onlyProtocolManager
setRewardPercentageBase	External	Can Modify State	onlyProtocolManager
setMinNodeStakes	External	Can Modify State	onlyProtocolManager
setMaxDuration	External	Can Modify State	onlyProtocolManager
setValidatorMinStake	External	Can Modify State	onlyProtocolManager
setValidatorFeeWithdrawalPercentage	External	Can Modify State	onlyProtocolManager
setTotalStakes	External	Can Modify State	-
setGamma	External	Can Modify State	onlyProtocolManager

FlockTaskManagerV2Upgradeable			
setConfig	External	Can Modify State	onlyDaoAdmin
setFeePercentage	External	Can Modify State	onlyProtocolManager
setDailyMintResetCount	External	Can Modify State	onlyProtocolManager
setDailyMintDecayPercentage	External	Can Modify State	onlyProtocolManager
setCurrentDailyMintCount	External	Can Modify State	onlyProtocolManager
getNodeStakes	External	-	-
getValidatorStakes	External	-	-
getAvailableRewardTasksForUser	External	-	-
isTaskCompleted	External	-	-
getTotalStakes	External	-	-
getNodeTasks	External	-	-
getValidatorTasks	External	-	-
createTask	External	Can Modify State	onlyProtocolManager
stakeTokensForTrainingNodes	External	Can Modify State	-
withdrawStakeTokensForTrainingNodes	External	Can Modify State	-
claimRewards	External	Can Modify State	-
stakeTokensForValidators	External	Can Modify State	-
withdrawStakeTokensForValidators	External	Can Modify State	-
transferRewardsToFLTasks	External	Can Modify State	onlyProtocolManager

FlockTaskManagerV2Upgradeable			
mintDailyReward	External	Can Modify State	onlyRewardOperator
markTaskAsFinished	Internal	Can Modify State	-
uploadRewardResultsForTrainingNodes	Public	Can Modify State	onlyRewardOperator
uploadRewardResultsForValidators	Public	Can Modify State	onlyRewardOperator
uploadRewardResultsOnlyForValidators	Public	Can Modify State	onlyRewardOperator
distributeRewardToDelegators	Internal	Can Modify State	-
uploadFinalRewardResultsForTrainingNodes	Public	Can Modify State	onlyRewardOperator
afterDelegationUpdate	External	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Information] The contract storage structure has changed

Category: Scoping and Declarations Audit

Content

This audit includes reviewing iterations of previously existing contracts. The v2 versions of these contracts have changed storage slots compared to the old implementations, which means that even if the old contracts are upgradeable, they cannot be directly upgraded to v2 contracts to avoid the risk of storage slot conflicts. Contracts with changed storage slots include: FlockMiniPoolV2, FlockPoolManagerV2Upgradeable, FlockConfigV2, FlockStakeInfoV2Upgradeable, and FlockTaskManagerV2Upgradeable.

Solution

The project team should be aware that v2 versions of the contracts cannot be directly upgraded from the old contracts.

Status

Acknowledged; After communicating with the project team, they indicated that they will not upgrade the contract directly from v1 to v2.

[N2] [Suggestion] Redundant owner inheritance

Category: Others

Content

In the FlockMiniPoolV2 contract, it inherits the Ownable contract from OpenZeppelin. In fact, this contract does not use any functionality related to the Ownable contract, so inheriting the Ownable contract is redundant. Similarly, importing the gmFlockExchangeUpgradeable contract in the FlockMiniPoolV2 contract is also redundant.

Code location: contracts/v2/FlockMiniPoolV2.sol#L11,L18

```
import {gmFlockExchangeUpgradeable} from "./gmFlockExchangeUpgradeable.sol";

contract FlockMiniPoolV2 is Ownable {
    ...
}
```

Solution

It is recommended to remove redundant code to simplify the contract implementation.

Status

Acknowledged

[N3] [Suggestion] Unsafe integer conversion

Category: Integer Overflow and Underflow Vulnerability

Content

In the FlockMiniPoolV2 contract, the lingeringRewards variable is used to record delegators' rewards. When recording rewards, lingeringReward needs to be converted from uint to int type, but during this conversion process, there is no check whether lingeringReward is less than the maximum value of int256, which may lead to overflow risks during conversion.

Code location: contracts/v2/FlockMiniPoolV2.sol#L79,L104,L134,L180

```
function delegate(uint256 _amount) external {
    ...
    lingeringRewards[delegator] -= int256(lingeringReward);
    ...
}

function undelegate(uint256 _amount) external {
    ...
    lingeringRewards[delegator] += int256(lingeringReward);
    ...
}

function claimRewards() external {
    ...
    lingeringRewards[delegator] = -int256(nominalReward);
    ...
}

function claimable(address _delegator) public view returns (uint256) {
    ...
    return uint256(lingeringReward + int256(nominalReward));
}
```

Solution

Although the above risk is unlikely to occur in the current business scenario, it is still recommended that the project team use OpenZeppelin's SafeCast library for safe conversion.

Status

Fixed

[N4] [Suggestion] Not following the Checks-Effects-Interactions (CEI) specification

Category: Reentrancy Vulnerability

Content

In the FlockMiniPoolV2 contract, users can undelegate through the undelegate function. When undelegating, the contract transfers gmFlock tokens to the delegator before updating the contract variables, which does not comply with the CEI (Checks-Effects-Interactions) pattern. Although there is no re-entrancy risk in this business scenario, following the standard can better improve the robustness of the code.

The same is true in the claimRewards function of the FlockMiniPoolV2 contract.

Code location:

contracts/v2/FlockMiniPoolV2.sol#L100,L128

```
    function undelegate(uint256 _amount) external {
        ...
        require(config.getGmFlockToken().transfer(delegator, _amount), "transfer
failed");

        uint256 assetValueOfAmount=assetValue(_amount);
        uint256 lingeringReward =  assetValueOfAmount-_amount;
        lingeringRewards[delegator] += int256(lingeringReward);

        // perform global update after the asset value is calculated
        delegatedAmount[delegator] -= _amount;
        totalAssetValue = assetValue(totalDelegationAmount-_amount);
        totalDelegationAmount -= _amount;

        ...
    }

    function claimRewards() external {
        ...
        require(config.getFlockToken().transfer(delegator, rewards - fee), "transfer
failed");
        require(config.getFlockToken().transfer(config.flockTokenAddress(), fee),
"transfer failed");
        claimedRewards[delegator] += rewards - fee;

        uint256 assetValueOfAmount=assetValue(delegationOf(delegator));
        uint256 nominalReward=assetValueOfAmount-delegationOf(delegator);
        lingeringRewards[delegator] = -int256(nominalReward);
        ...
    }
```

Solution

It is recommended to follow the CEI pattern when dealing with transfer operations: first modify the contract state, then execute the transfer operation.

Status

Fixed

[N5] [Medium] Front-running Risk in Reward Distribution

Category: Reordering Vulnerability

Content

In the FlockMiniPoolV2 contract, any user can add rewards for delegators through the fundPool function. Once rewards are added, users who delegated before this point can receive additional rewards. However, it's important to note that the reward amount is related to the amount of user delegation but has no relationship with the duration of the delegation. This means that if poolDelegatorCoolDown is small, users can monitor when the fundPool function is being called and use front-running to make large delegations to increase capital efficiency, which is not the intended behavior of the protocol and breaks fairness for other long-term delegators.

Code location: contracts/v2/FlockMiniPoolV2.sol#L148

```
    function fundPool(uint256 _reward) external {
        require(config.getFlockToken().transferFrom(msg.sender, address(this),
_reward), "Transfer failed");
        totalAssetValue += _reward;
        emit Fund(_reward);
    }
```

Solution

To completely solve this risk, the delegation duration should be considered when calculating rewards. Without changing the reward rules, setting an appropriate poolDelegatorCoolDown can mitigate this risk.

Status

Fixed

[N6] [Information] totalAssetValue increases from different sources

Category: Others

Content

In the FlockMiniPoolV2 contract, the totalAssetValue variable is used to record user delegation amounts and reward increments. When users perform delegate/undelegate operations, the token involved is the gmFlock token, and in this case, the increase/decrease of the totalAssetValue variable depends on the delegation amount of gmFlock tokens. However, when users add rewards to the FlockMiniPoolV2 contract through the fundPool function, the token

involved is the Flock token. This means that the sources causing changes to the totalAssetValue variable are not the same.

Code location: contracts/v2/FlockMiniPoolV2.sol#L71,L146

```
    function delegate(uint256 _amount) external {
        ...
        require(config.getGmFlockToken().transferFrom(delegator, address(this),
_amount), "Transfer failed");
        ...
        totalAssetValue = assetValue(totalDelegationAmount+_amount);
        ...
    }

    function fundPool(uint256 _reward) external {
        require(config.getFlockToken().transferFrom(msg.sender, address(this),
_reward), "Transfer failed");
        totalAssetValue += _reward;
        emit Fund(_reward);
    }
```

Solution

N/A

Status

Fixed

[N7] [Low] Optimizable mintDailyReward operation

Category: Design Logic Audit

Content

In the FlockTaskManagerV2Upgradeable contract, mintDailyReward is mainly used for reward minting. When the daily minting count reaches dailyMintResetCount, the reward amount will be multiplied by dailyMintDecayPercentage and currentDailyMintCount will be reset. It's important to note that the function checks currentDailyMintCount first and then increments it, meaning the check for currentDailyMintCount starts from 0 rather than 1. This could result in the first reset cycle having one more mint than expected.

Code location: contracts/v2/FlockTaskManagerV2Upgradeable.sol#L526

```
    function mintDailyReward() external onlyRewardOperator {
        if(currentDailyMintCount>=dailyMintResetCount){
            rewardTokenPerDay = (rewardTokenPerDay * dailyMintDecayPercentage) /
rewardPercentageBase;
            currentDailyMintCount = 0;
        }
        currentDailyMintCount++;
        config.getFlockToken().mint(address(this), rewardTokenPerDay);
        emit MintDailyReward(rewardTokenPerDay);
    }
```

Solution

It is recommended to place the increment of currentDailyMintCount before the reset check.

Status

Fixed

[N8] [Medium] Shadow variables

Category: Scoping and Declarations Audit

Content

In the uploadRewardResultsForTrainingNodes function of the FlockTaskManagerV2Upgradeable contract, the unlockedRewardPercentage variable is used to calculate delegators' rewards. However, it should be noted that unlockedRewardPercentage is already a global variable, yet the uploadRewardResultsForTrainingNodes function defines a temporary variable with the same name, which creates a variable shadowing issue.

Code location: contracts/v2/FlockTaskManagerV2Upgradeable.sol#L567C17-L570

```
    uint256 public unlockedRewardPercentage; // Unlocked Reward Percentage for daily
emission,100 is 10%
    ...
    function uploadRewardResultsForTrainingNodes(uint256 taskId, address[] calldata
trainingNodesAddrs, uint256[] calldata rewardPercentages) public onlyRewardOperator {
        ...
        uint256 unlockedRewardPercentage = rewardPercentageBase -
lockedRewardPercentage;
        lockedRewardsForNodes[taskId] += (lockedRewardPercentage *
taskRewardForNodeAndDelegator) / (rewardPercentageBase);
        for (uint i = 0; i < trainingNodesAddrs.length; i++) {
            uint256 newReward = (unlockedRewardPercentage *
taskRewardForNodeAndDelegator * rewardPercentages[i]) / (rewardPercentageBase *
```

```
rewardPercentageBase);
            ...
        }
        ...
    }
```

Solution

It is recommended to remove the temporary unlockedRewardPercentage variable and use the global variable instead.

Status

Fixed

[N9] [Suggestion] Necessary variable checks

Category: Design Logic Audit

Content

In the gmFlockExchangeUpgradeable contract, the t0 and t1 variables are primarily used to check user locking periods. According to business requirements, t1 should be greater than t0. The admin role can set the t1 and t0 variables through the setExchangeParams function, but there is no check for the relative size of these two variables.

Code location: contracts/v2/gmFlockExchangeUpgradeable.sol#L57-L62

```
    function setExchangeParams(uint256 _t0,uint256 _t1,uint256 _epsilon,uint256 _base)
external onlyAdmin {
        t0 = _t0;
        t1 = _t1;
        epsilon = _epsilon;
        base = _base;
    }
```

Solution

It is recommended to add a check in the setExchangeParams function to ensure that `_t1` must be greater than `_t0`.

Status

Fixed

[N10] [Information] Potential DoS risk for token swapping via admin

Category: Denial of Service Vulnerability

Content

In the gmFlockExchangeUpgradeable contract, the admin user can exchange tokens for a beneficiary through the exchangeFlock function. During the exchange process, the Flock token locking information is recorded in the admin's userLocks state, but the gmFlock tokens are minted to the beneficiary address. This means that the minting record and the token recipient are separate, which prevents both the admin and the beneficiary from unlocking tokens through the claimUnlockedFlock function, because the beneficiary has no userLocks record, while the admin does not have the corresponding gmFlock tokens.

Similarly, neither the admin nor the beneficiary can unlock tokens through the claimUnlockedFlockByIds function.

Code location: contracts/v2/gmFlockExchangeUpgradeable.sol#L99,L115,L159

```
   function exchangeFlock(uint256 flockAmount, uint256 lockPeriod,address
beneficiary) external {
        ...
        lockInfoMap[currentLockInfoId] = LockInfo(currentLockInfoId,flockAmount,
gmFlockAmount, unlockTime, beneficiary);
        userLocks[msg.sender].push(currentLockInfoId);
        currentLockInfoId++;

        require(config.getFlockToken().transferFrom(msg.sender, address(this),
flockAmount), "FLOCK transfer failed");
        IMintBurnableERC20(address(config.gmFlockTokenAddress())).mint(beneficiary,
gmFlockAmount);

        emit FlockExchanged(msg.sender, flockAmount, gmFlockAmount, unlockTime,
currentLockInfoId-1);
    }

    function claimUnlockedFlock(uint256 timestamp) external {
        require(timestamp <= block.timestamp, "Cannot claim for future time");

        uint256[] storage locks = userLocks[msg.sender];
        ...
        IMintBurnableERC20(config.gmFlockTokenAddress()).burn(msg.sender,
totalgmFlock);
        require(config.getFlockToken().transfer(msg.sender, totalFlock), "FLOCK
transfer failed");
    }

    function claimUnlockedFlockByIds(uint256[] calldata ids) external {
        uint256[] storage locks = userLocks[msg.sender];
```

```
        ...
        while (i < ids.length) {
            ...
            require(lock.beneficiary == msg.sender, "Not the beneficiary");
            ...
        }
        ...
        IMintBurnableERC20(config.gmFlockTokenAddress()).burn(msg.sender,
totalgmFlock);
        require(config.getFlockToken().transfer(msg.sender, totalFlock), "FLOCK
transfer failed");
    }
```

Solution

If this is not the intended design, it is recommended to record the token locking information in the beneficiary's userLocks state when performing the exchangeFlock operation.

Status

Acknowledged; After communicating with the project team, the project team stated that this is consistent with our original design: the DAO admin has the authority to lock FLOCKs in order to mint gmFLOCKs as rewards for others. Once distributed, these gmFLOCKs are non-reversible and cannot be converted back into FLOCKs.

[N11] [Suggestion] Potential risk of premature termination of the claimUnlockedFlockByIds operation

Category: Design Logic Audit

Content

In the gmFlockExchangeUpgradeable contract, users can unlock specific indexes of userLocks through the claimUnlockedFlockByIds function. It uses the findLockIndex function to match the unlock id with the id recorded in userLocks to verify if the user's unlock id is valid. When ids do not match, the findLockIndex function directly reverts instead of skipping the current invalid id and continuing to check the next one. This causes the claimUnlockedFlockByIds operation to terminate prematurely.

Code location: contracts/v2/gmFlockExchangeUpgradeable.sol#L188

```
    function claimUnlockedFlockByIds(uint256[] calldata ids) external {
        ...
        while (i < ids.length) {
            uint256 lockIndex = findLockIndex(locks, ids[i]); // Find correct index in
`locks`
```

```
            require(lockIndex < locks.length, "Invalid lock ID");

            ...
        }
        ...
    }

    function findLockIndex(uint256[] storage locks, uint256 id) internal view returns
(uint256) {
        for (uint256 j = 0; j < locks.length; j++) {
            if (locks[j] == id) {
                return j;
            }
        }
        revert("Lock ID not found");
    }
```

Solution

If this is not the intended design, it is recommended that when matching is unsuccessful in the findLockIndex function, it should return the maximum uint256 value and the claimUnlockedFlockByIds function should continue with unlocking the next id.

Status

Acknowledged; After communicating with the project team, the project team stated that this was the expected design. In order to ensure the atomicity of the operation, all operations must either succeed or fail.

[N12] [Low] Potential DoS risk of token unlocking

Category: Denial of Service Vulnerability

Content

In the gmFlockExchangeUpgradeable contract, users can unlock tokens through the claimUnlockedFlock and claimUnlockedFlockByIds functions. When unlocking tokens, both functions need to loop through the user's userLocks list. If the userLocks list becomes too large, it may lead to a DoS (Denial of Service) risk.

Code location: contracts/v2/gmFlockExchangeUpgradeable.sol#L121,L183

```
    function claimUnlockedFlock(uint256 timestamp) external {
        require(timestamp <= block.timestamp, "Cannot claim for future time");

        uint256[] storage locks = userLocks[msg.sender];
        ...
```

```
        while (i < locks.length) {
            ...
        }
        ...
    }

    function claimUnlockedFlockByIds(uint256[] calldata ids) external {
        uint256[] storage locks = userLocks[msg.sender];
        ...
        while (i < ids.length) {
            uint256 lockIndex = findLockIndex(locks, ids[i]); // Find correct index in
`locks`
            ...
        }
        ...
    }

    function findLockIndex(uint256[] storage locks, uint256 id) internal view returns
(uint256) {
        for (uint256 j = 0; j < locks.length; j++) {
            ...
        }
        revert("Lock ID not found");
    }
```

Solution

By limiting the minimum exchange amount in the exchangeFlock function, the issue of the list becoming too large due to dust exchanges can be effectively mitigated.

Status

Fixed

[N13] [Medium] The risks of excessive privilege

Category: Authority Control Vulnerability Audit

Content

In the v2 protocol, FlockPoolManagerV2Upgradeable, FlockStakeInfoV2Upgradeable, FlockTaskManagerV2Upgradeable, and gmFlockUpgradeable contracts all inherit from the RbacUpgradeable contract. During the initialization of the RbacUpgradeable contract, it sets the deployer as the DAO_ADMIN role, which means the DAO_ADMIN role is managed by an EOA (Externally Owned Account). In the absence of EIP7702 implementation, this creates a single point of failure risk.

In the gmFlockUpgradeable contract, the DAO_ADMIN role can arbitrarily mint tokens or burn any user's tokens through the mint/burn functions, which leads to the risk of excessive privilege.

Code location:

contracts/v2/RbacUpgradeable.sol#L23

```
    function __RbacUpgradeable_init(address deployer) internal onlyInitializing {
        __AdminUpgradeable_init(deployer);
        _setupRole(DAO_ADMIN, deployer);
    }
```

contracts/v2/gmFlockUpgradeable.sol#L92-L105

```
    function mint(address _account, uint256 _amount) external onlyDaoAdmin {
        currentMinted += _amount;
        _mint(_account, _amount);
    }

    function burn(address _account, uint256 _amount) external onlyDaoAdmin {
        currentMinted -= _amount;
        _burn(_account, _amount);
    }
```

Solution

In the short term, transferring privileged roles to a multi-signature wallet can effectively mitigate the single point of failure risk. In the long term, transferring privileged roles to DAO governance can effectively address the risk of excessive privilege. During the transition period, managing through multi-signature with delayed transaction execution via timelock can effectively mitigate the risk of excessive privilege.

Status

Fixed;

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002504140002	SlowMist Security Team	2025.04.10 - 2025.04.14	Passed

Summary conclusion: The SlowMist security team uses a manual and the SlowMist team's analysis tool to audit the project, during the audit work we found 3 medium risk, 2 low risk, 5 suggestions, and 3 information. 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