Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.06.07, the SlowMist security team received the Cojam team's security audit application for Cojam Market, 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.

Level	Description
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

Serial Number	Audit Class	Audit Subclass
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
		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

Audit Version:

https://github.com/cojam-limited/cojam_smart_contract

commit: 30dba5879683fbed797a2162e4e86cffe536f0dc

Fixed Version:

https://github.com/cojam-limited/cojam_smart_contract

commit: c80e428a50168276bbb3683ce81b472b15d46a62

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Early unlock issue	Others	Low	Confirmed
N2	Duplicate creation of market issue	Design Logic Audit	Medium	Fixed
N3	Redundant logic issue	Others	Suggestion	Confirmed
N4	Potential Compatibility Issue	Design Logic Audit	Suggestion	Confirmed
N5	Safe Transfer issue	Others	Suggestion	Confirmed
N6	Checks-Effects-Interactions not followed	Design Logic Audit	Suggestion	Confirmed

4 Code Overview

4.1 Contracts Description

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:

KIP7			
Function Name	Visibility	Mutability	Modifiers
_transfer	Internal	Can Modify State	-
_approve	Internal	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
totalSupply	External	-	-
balanceOf	External	-	-
allowance	External	-	-
name	External	-	-
symbol	External	-	-
decimals	External	-	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
approve	External	Can Modify State	-

KIP7Burnable

KIP7Burnable			
Function Name	Visibility	Mutability	Modifiers
burn	External	Can Modify State	whenNotPaused
burnFrom	External	Can Modify State	whenNotPaused

KIP7Lockable			
Function Name	Visibility	Mutability	Modifiers
_lock	Internal	Can Modify State	-
_unlock	Internal	Can Modify State	-
unlock	External	Can Modify State	-
unlockAll	External	Can Modify State	-
releaseLock	External	Can Modify State	onlyOwner
transferWithLockUp	External	Can Modify State	onlyOwner
lockInfo	External	-	-
totalLocked	External	-	-

AnswerBettingConstraint			
Function Name	Visibility	Mutability	Modifiers
putBettingKey	Internal	Can Modify State	-
containsBettingKey	Internal	-	-
getAvailableBettingKeys	Internal	-	-

Freezable			
Function Name	Visibility	Mutability	Modifiers
freeze	External	Can Modify State	onlyOwner
unFreeze	External	Can Modify State	onlyOwner
isFrozen	External	-	-

MarketAnswerConstraint			
Function Name	Visibility	Mutability	Modifiers
putAnswerKey	Internal	Can Modify State	-
containsAnswerKey	Internal	-	-
getAvailableAnswerKeys	Internal	-	-

MarketManager			
Function Name	Visibility	Mutability	Modifiers
_getBetting	Internal	-	-
_bet	Internal	Can Modify State	-
_isRetrievable	Internal	-	-
_availableReceiveTokens	Internal	-	-
_draftMarket	Internal	Can Modify State	-
_approveMarket	Internal	Can Modify State	-
_addAnswerKeys	Internal	Can Modify State	-
_getMarket	Internal	-	-

MarketManager			
_finishMarket	Internal	Can Modify State	-
_setSuccessMarket	Internal	Can Modify State	-
_setAdjournMarket	Internal	Can Modify State	-
_getAnswerKeys	Internal	-	-
_getAnswer	Internal	-	-
_getBettingKeys	Internal	-	-
_changeMarketStatus	Internal	Can Modify State	-
_isMarketStatus	Internal	-	-

Ownable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
owner	External	-	-
transferOwnership	Public	Can Modify State	onlyOwner
renounceOwnership	External	Can Modify State	onlyOwner
_transferOwnership	Internal	Can Modify State	-

Pausable			
Function Name	Visibility	Mutability	Modifiers
pause	External	Can Modify State	onlyOwner whenNotPaused
unPause	External	Can Modify State	onlyOwner whenPaused

Pausable			
paused	External	-	-

UserManager			
Function Name	Visibility	Mutability	Modifiers
_insertLockUser	Internal	Can Modify State	-
_removeLockUser	Internal	Can Modify State	-
_containsLockUser	Internal	-	-

CojamMarket			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getMarket	External	-	-
getMarketFee	External	-	-
getAnswer	External	-	-
getBetting	External	-	-
getAccounts	External	-	-
_getAccounts	Internal	-	-
availableReceiveTokens	External	-	-
receiveToken	External	Can Modify State	-
isRetrievable	External	-	-
retrieveTokens	External	Can Modify State	onlyOwner

CojamMarket			
finishMarket	External	Can Modify State	onlyOwner
successMarket	External	Can Modify State	onlyOwner
adjournMarket	External	Can Modify State	onlyOwner
dividendToken	Internal	Can Modify State	-
setAccount	External	Can Modify State	onlyOwner
_setAccount	Internal	Can Modify State	-
bet	External	Can Modify State	bettable
availableBet	External	-	bettable
draftMarket	External	Can Modify State	onlyOwner
approveMarket	External	Can Modify State	onlyOwner
addAnswerKeys	External	Can Modify State	onlyOwner
isLock	External	-	-
lock	Public	Can Modify State	onlyOwner
unlock	Public	Can Modify State	onlyOwner

CojamToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
transfer	External	Can Modify State	whenNotFrozen whenNotPaused checkLock
transferFrom	External	Can Modify State	whenNotFrozen whenNotPaused checkLock

CojamToken			
approve	External	Can Modify State	-
name	External	-	-
symbol	External	-	-
decimals	External	-	-

4.3 Vulnerability Summary

[N1] [Low] Early unlock issue

Category: Others

Content

In the KIP7Lockable contract, the owner can release the locked tokens of any user through the releaseLock function.

If some tokens are unlocked unexpectedly, it will lead to unknown market risks.

Code location: cojam_smart_contract-main/contracts/kip7/KIP7Lockable.sol

```
function releaseLock(address from)
external
onlyOwner
returns (bool)
{
    for(uint256 i = 0; i < _locks[from].length; i++){
        if(_unlock(from, i)){
            i--;
        }
    }
    return true;
}
```

Solution

It is recommended to transfer the unlocking authority to community governance.

Status

Confirmed; After communicating with the project team, the project team stated that it will use community governance in the future.

[N2] [Medium] Duplicate creation of market issue

Category: Design Logic Audit

Content

In the CojamMarket contract, the owner can create a market through the draftMarket function, but it does not check whether the market already exists. If it is created repeatedly, it will overwrite the previously created market.

Code location: cojam_smart_contract-main/contracts/CojamMarket.sol

```
    function draftMarket(
        uint256 marketKey,
        address creator,
        string calldata title,
        uint256 creatorFee,
        uint256 creatorFeePercentage,
        uint256 cojamFeePercentage,
        uint256 charityFeePercentage
    ) external onlyOwner returns (bool) {
        _draftMarket(
            marketKey,
            creator,
            title,
            DRAFT_MARKET_KEY,
            creatorFee,
            creatorFeePercentage,
            cojamFeePercentage,
            charityFeePercentage
        );

        emit DraftMarket(
            marketKey,
```

```
            title,
            creator,
            creatorFee,
            creatorFeePercentage,
            cojamFeePercentage,
            charityFeePercentage
        );

        return true;
    }
```

Solution

It is recommended to check whether the market already exists when performing the draftMarket operation.

Status

Fixed

[N3] [Suggestion] Redundant logic issue

Category: Others

Content

In the CojamMarket contract, the owner can lock/unlock the user account through the lock/unlock function. But locked users can still operate on the market, so this business logic is not used.

Code location: cojam_smart_contract-main/contracts/CojamMarket.sol

```
    function isLock(address target) external view returns (bool) {
        return _containsLockUser(target);
    }

    function lock(address[] memory targets)
        public
        onlyOwner
        returns (bool[] memory)
    {
        ...
    }
```

```
    function unlock(address[] memory targets)
        public
        onlyOwner
        returns (bool[] memory)
    {
        ...
        }
```

Solution

If the design is not intended, it is recommended to remove redundant logic that is not used.

Status

Confirmed

[N4] [Suggestion] Potential Compatibility Issue

Category: Design Logic Audit

Content

In the CojamMarket contract, users can bet through the betting function and transfer vote tokens into this contract. However, if the vote token is a deflationary token, the amount of tokens actually received by the contract is less than the amount transferred by the user, but the contract still records the amount passed in by the user.

Code location: cojam_smart_contract-main/contracts/CojamMarket.sol

```
    function bet(
        uint256 marketKey,
        uint256 answerKey,
        uint256 bettingKey,
        uint256 tokens
    )
        external
        bettable(marketKey, answerKey, bettingKey, tokens)
        returns (bool)
    {
        _bet(marketKey, answerKey, bettingKey, msg.sender, tokens); // 데이터 변경이 실패하면 거래 전으로 돌리기
        baseToken.transferFrom(msg.sender, address(this), tokens);
        emit Bet(marketKey, answerKey, bettingKey, tokens);
```

```
        return true;
    }
```

Solution

It is recommended to use the difference between the vote token balance of the contract before and after the user's transfer as the actual amount transferred by the user.

Status

Confirmed

[N5] [Suggestion] Safe Transfer issue

Category: Others

Content

In the CojamMarket contract, the bet function transfers vote tokens to this contract through the transferFrom interface. The dividendToken function transfers vote tokens to users through the transfer interface. If the vote token does not meet the EIP20 standard, there will be unknown risks.

Code location: cojam_smart_contract-main/contracts/CojamMarket.sol

```
    function dividendToken(
        Market storage market,
        address to,
        uint256 token
    ) internal {
        market.marketRemainTokens = market.marketRemainTokens.sub(token);
        baseToken.transfer(to, token);
    }

    function bet(
        uint256 marketKey,
        uint256 answerKey,
        uint256 bettingKey,
        uint256 tokens
    )
        external
```

```
        bettable(marketKey, answerKey, bettingKey, tokens)
        returns (bool)
    {
        _bet(marketKey, answerKey, bettingKey, msg.sender, tokens); // 데이터 변경이 실패하면 거래 전으로 돌리기
        baseToken.transferFrom(msg.sender, address(this), tokens);
        emit Bet(marketKey, answerKey, bettingKey, tokens);

        return true;
    }
```

Solution

It is recommended to use OpenZeppelin's SafeERC20 library for token transfers.

Status

Confirmed

[N6] [Suggestion] Checks-Effects-Interactions not followed

Category: Design Logic Audit

Content

In the CojamMarket contract, the user can perform the betting operation through the bet function, but the contract state is modified first and then the transfer operation is performed. Users can receive vote tokens through the receiveToken function, but the user is transferred first and then the contract status is modified.

Code location: cojam_smart_contract-main/contracts/CojamMarket.sol

```
    function receiveToken(uint256 marketKey, uint256 bettingKey)
        external
        returns (bool)
    {
        uint256 receiveTokens = _availableReceiveTokens(marketKey, bettingKey);

        Market storage market = _markets[marketKey];
        dividendToken(market, msg.sender, receiveTokens);
        _bettings[bettingKey].tokens = 0;

        emit TokenReceived(msg.sender, marketKey, bettingKey, receiveTokens);
```

```
        return true;
    }

    function bet(
        uint256 marketKey,
        uint256 answerKey,
        uint256 bettingKey,
        uint256 tokens
    )
        external
        bettable(marketKey, answerKey, bettingKey, tokens)
        returns (bool)
    {
        _bet(marketKey, answerKey, bettingKey, msg.sender, tokens);
        baseToken.transferFrom(msg.sender, address(this), tokens);
        emit Bet(marketKey, answerKey, bettingKey, tokens);

        return true;
    }
```

Solution

It is recommended to follow the Checks-Effects-Interactions principle. When users transfer tokens to the contract, the contract should perform the transfer first and then modify the state. When the contract transfers tokens to the user, the contract state should be modified before transferring.

Status

Confirmed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002206140002	SlowMist Security Team	2022.06.07 - 2022.06.14	Low Risk

Summary conclusion: The SlowMist security team used a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 medium risk, 1 low risk, and 4 suggestions. And 1 low risk, 4 suggestions were confirmed and fixed; All other findings were 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist