Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.09.19, the SlowMist security team received the PancakeSwap team's security audit application for PancakeSwap - CrossChain, 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/chefcooper/pancake-contracts/tree/feature/cross-chain/projects/cross-chain/contracts

commit: 35d1f31851bc98a6a13fcd467a67eae287b49563

Fixed Version:

https://github.com/chefcooper/pancake-contracts/tree/feature/cross-chain/projects/cross-chain/contracts

commit: ed53e47c747409330788e7fc51b75ae555cf0cf5

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Incorrect whitelist check	Design Logic Audit	High	Fixed
N2	Redundant balance checks	Others	Suggestion	Ignored
N3	Gas optimization	Others	Suggestion	Fixed
N4	Potential fake mining risk	Design Logic Audit	Low	Ignored
N5	Incorrect function state	Design Logic Audit	Low	Fixed
N6	Inappropriate exchange rate decimal	Design Logic Audit	Suggestion	Fixed
N7	Redundant payable label	Others	Suggestion	Fixed
N8	Potential duplicate deposit and withdrawal issue	Design Logic Audit	Low	Fixed

NO	Title	Category	Level	Status
N9	Allowance depletion issue	Design Logic Audit	Low	Fixed
N10	Incorrect interface call	Design Logic Audit	Medium	Fixed

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:

CrossFarmingVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
addWhiteListPool	Public	Can Modify State	onlyOwner
add	Public	Can Modify State	onlyOwner
deposit	External	Payable	nonReentrant whenNotPaused notContract
withdraw	External	Payable	nonReentrant whenNotPaused notContract

CrossFarmingVault			
emergencyWithdraw	External	Payable	nonReentrant notContract
ackWithdraw	External	Payable	nonReentrant onlySender
ackEmergencyWithdraw	External	Payable	nonReentrant onlySender
fallbackDeposit	External	Can Modify State	onlyOperator onlyNotFallback
fallbackWithdraw	External	Can Modify State	onlyOperator onlyNotFallback
_fallback	Internal	Can Modify State	-
setOperator	External	Can Modify State	onlyOwner
setCrossFarmingContract	External	Can Modify State	onlyOwner
poolLength	External	-	-
calcFee	External	Can Modify State	-
encodeMessage	Public	Can Modify State	-
pause	External	Can Modify State	onlyOwner whenNotPaused
unpause	External	Can Modify State	onlyOwner whenPaused
_isContract	Internal	-	-

CrossFarmingSender			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

CrossFarmingSender			
sendFarmMessage	External	Payable	onlyVault
executeMessage	External	Payable	onlyMessageBus
setVault	External	Can Modify State	onlyOwner
setReceiver	External	Can Modify State	onlyOwner
setOracle	External	Can Modify State	onlyOwner
setOracleUpdateBuffer	External	Can Modify State	onlyOwner
setGaslimits	External	Can Modify State	onlyOwner
setFloatRate	External	Can Modify State	onlyOwner
setBnbChange	External	Can Modify State	onlyOwner
setCreateProxyGasLimit	External	Can Modify State	onlyOwner
estimateDestGaslimit	Public	-	-
drainToken	External	Can Modify State	onlyOwner
claimFee	External	Payable	onlyOwner
_getPriceFromOracle	Internal	Can Modify State	-

CrossFarmingReceiver			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
executeMessage	External	Payable	onlyMessageBus

CrossFarmingReceiver

fallbackDeposit	External	Can Modify State	onlyOperator
fallbackWithdraw	External	Can Modify State	onlyOperator
setOperator	External	Can Modify State	onlyOwner
setBnbChange	External	Can Modify State	onlyOwner
drainToken	External	Can Modify State	onlyOwner
claimFee	External	Payable	onlyOwner
_createProxy	Internal	Can Modify State	-

CrossFarmingProxy

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
deposit	External	Can Modify State	nonReentrant onlyFactory onlyNotFallback
withdraw	External	Payable	nonReentrant onlyFactory onlyNotFallback
emergencyWithdraw	External	Payable	nonReentrant onlyFactory onlyNotFallback
harvest	External	Can Modify State	nonReentrant
fallbackDeposit	External	Can Modify State	onlyFactory onlyNotFallback
fallbackWithdraw	External	Can Modify State	onlyFactory onlyNotFallback
_safeTransfer	Internal	Can Modify State	-

CrossFarmingToken

CrossFarmingToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
mint	Public	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [High] Incorrect whitelist check

Category: Design Logic Audit

Content

In the CrossFarmingVault contract, the owner can add a new pool through the add function and check if it is in the whitelist. However, the non-whitelisted pool is allowed to enter by mistake, but the whitelisted pool cannot be successfully added.

Code location: contracts/CrossFarmingVault.sol

```
function add(IERC20 _lpToken, uint256 _mcv2PoolId) public onlyOwner {
    require(!exists[_lpToken], "Token added");
    require(!whitelistPool[_mcv2PoolId], "None whitelist pool");
    require(poolMapping[_mcv2PoolId] == 0, "MCV2 pool mappinged");
    require(_lpToken.balanceOf(address(this)) >= 0, "None ERC20 token");

    ...
}
```

Solution

It is recommended to only allow whitelisted pools to be added.

Status

Fixed

[N2] [Suggestion] Redundant balance checks

Category: Others

Content

In the CrossFarmingVault contract, the owner can add a new pool through the add function and check whether the balance of the LP tokens added in the current contract is greater than or equal to 0. In theory, the balance of each account will be greater than or equal to 0, so this check is redundant.

Code location: contracts/CrossFarmingVault.sol

```
    function add(IERC20 _lpToken, uint256 _mcv2PoolId) public onlyOwner {
        require(!exists[_lpToken], "Token added");
        require(!whitelistPool[_mcv2PoolId], "None whitelist pool");
        require(poolMapping[_mcv2PoolId] == 0, "MCV2 pool mappinged");
        require(_lpToken.balanceOf(address(this)) >= 0, "None ERC20 token");

        ...
    }
```

Solution

It is recommended to remove redundant code to save gas.

Status

Ignored; After communicating with the project team, the project team stated that this line is just for make sure the LP token is a ERC20 token(have balanceOf interface), in case the owner add wrong token address.

[N3] [Suggestion] Gas optimization

Category: Others

Content

In the CrossFarmingVault contract, the deposit, withdraw and emergencyWithdraw functions use low-level calls to make external function calls, which will consume more gas than using the interface method. The same is true for the executeMessage function in the CrossFarmingReceiver contract.

Code location:

contracts/CrossFarmingVault.sol

```
    function deposit(uint256 _pid, uint256 _amount) external payable nonReentrant
whenNotPaused notContract {
        ...
        (bool success, ) = CROSS_FARMING_SENDER.call{value: msg.value}(
            abi.encodeWithSignature("sendFarmMessage(bytes)", message)
        );

        ...
    }

    function withdraw(uint256 _pid, uint256 _amount) external payable nonReentrant
whenNotPaused notContract {
        ...
        (bool success, ) = CROSS_FARMING_SENDER.call{value: msg.value}(
            abi.encodeWithSignature("sendFarmMessage(bytes)", message)
        );

        ...
    }

    function emergencyWithdraw(uint256 _pid) external payable nonReentrant
notContract {
        ...
        (bool success, ) = CROSS_FARMING_SENDER.call{value: msg.value}(
            abi.encodeWithSignature("sendFarmMessage(bytes)", message)
        );
        ...
    }
```

contracts/CrossFarmingVault.sol

```
    function executeMessage(
        address _sender,
        uint64 _srcChainId,
        bytes calldata _message,
        address // executor who called the MessageBus execution function
    ) external payable override onlyMessageBus returns (ExecutionStatus) {
        ...
        } else if (request.msgType == DataTypes.MessageTypes.Withdraw) {
            proxy.withdraw(request.pid, request.amount, request.nonce);

            // send ack message back.
            (bool success, ) = _sender.call{value:
IMessageBus(messageBus).calcFee(_message)}(
                abi.encodeWithSignature("sendFarmMessage(bytes)", _message)
            );

            require(success, "send withdraw ack message failed");
        } else if (request.msgType == DataTypes.MessageTypes.EmergencyWithdraw) {
            proxy.emergencyWithdraw(request.pid, request.nonce);

            // send ack message back.
            (bool success, ) = _sender.call{value:
IMessageBus(messageBus).calcFee(_message)}(
                abi.encodeWithSignature("sendFarmMessage(bytes)", _message)
            );
            require(success, "send emergencyWithdraw ack message failed");
        }
        ...
    }
```

Solution

If the design is not expected, it is recommended to use the interface to make external calls to save gas.

Status

Fixed

[N4] [Low] Potential fake mining risk

Category: Design Logic Audit

Content

In the protocol, if the user's deposit in the src chain is successful, but the deposit in the dest chain fails, the operator role can return the LP tokens to the user through the fallbackDeposit function. However, if the user successfully deposits in the dest chain, but the operator role still triggers the fallbackDeposit function to refund, this will cause the user to still receive CAKE token rewards in the dest chain.

Code location: contracts/CrossFarmingVault.sol

```
    function fallbackDeposit(
        address _user,
        uint256 _pid,
        uint256 _amount,
        uint64 _nonce
    ) external onlyOperator onlyNotFallback(_user, _pid, _nonce) {
        // double check
        require(deposits[_user][_pid][_nonce] == _amount, "withdraw amount not match
staking record");

        _fallback(_user, _pid, _amount, _nonce);
        emit FallbackDeposit(_user, _pid, _amount, _nonce);
    }
```

Solution

It is recommended to mark the nonce on the src chain after the deposit on the dest chain is successful.

Status

Ignored; After communicating with the project team, the project team stated that the operator is the representative of the contract owner, we must trust the operator, just like all contracts which have owner.

[N5] [Low] Incorrect function state

Category: Design Logic Audit

Content

In the CrossFarmingVault, the calcFee function is used to calculate the fee required for the message cross-chain, and the encodeMessage function is used to encode the message. Both can use the view function.

Code location: contracts/CrossFarmingVault.sol

```
    function calcFee(bytes calldata _message) external returns (uint256) {
        address messageBus = IMessage(CROSS_FARMING_SENDER).messageBus();
        return IMessageBus(messageBus).calcFee(_message);
    }
    function encodeMessage(
        address _account,
        uint256 _pid,
        uint256 _amount,
        DataTypes.MessageTypes _msgType
    ) public returns (bytes memory) {
        return
            abi.encode(
                DataTypes.CrossFarmRequest({
                    receiver: CROSS_FARMING_RECEIVER,
                    dstChainId: BSC_CHAIN_ID,
                    nonce: IMessage(CROSS_FARMING_SENDER).nonces(_account, _pid),
                    account: _account,
                    amount: _amount,
                    pid: _pid,
                    msgType: _msgType
                })
            );
    }
```

Solution

It is recommended to use the correct function state.

Status

Fixed

[N6] [Suggestion] Inappropriate exchange rate decimal

Category: Design Logic Audit

Content

In the CrossFarmingSender contract, the exchange rate is obtained by multiplying the BNB price by EXCHANGE_RATE_PRECISION and dividing the ETH price. In the future, if the price of ETH is greater than 1e5 times the price of BNB, the result of this algorithm will be 0.

Code location: contracts/CrossFarmingSender.sol

```
    function sendFarmMessage(bytes calldata _message) external payable onlyVault {
        ...
        uint256 exchangeRate = (uint256(bnbPrice) * EXCHANGE_RATE_PRECISION) /
uint256(ethPrice);
        ...
    }
```

Solution

It is recommended to increase EXCHANGE_RATE_PRECISION.

Status

Fixed

[N7] [Suggestion] Redundant payable label

Category: Others

Content

In the CrossFarmingSender and CrossFarmingReceiver contract, the claimFee function user transfers the native tokens in the contract, so this function does not need the payable label.

Code location: contracts/CrossFarmingReceiver.sol & contracts/CrossFarmingSender.sol

```
    function claimFee(uint256 _gas) external payable onlyOwner {
        require(_gas >= 2300, "claimFee gaslimit should exceed 2300 ");

        uint256 amount = address(this).balance;
        (bool success, ) = msg.sender.call{value: amount, gas: _gas}("");

        emit FeeClaimed(amount, success);
    }
```

Solution

It is recommended to remove the payable label.

Status

Fixed

[N8] [Low] Potential duplicate deposit and withdrawal issue

Category: Design Logic Audit

Content

In the protocol, the deposit function of CrossFarmingProxy is indirectly triggered by SGN's MessageBus to make deposits for users. If SGN repeatedly executes the message due to failure, the message of the same nonce will be executed multiple times.

Solution

It is recommended that CrossFarmingProxy mark the message nonce as used after the deposit and withdrawal are successful to avoid the risk of repeated execution.

Status

Fixed

[N9] [Low] Allowance depletion issue

Category: Design Logic Audit

Content

In the CrossFarmingProxy contract, the deposit function is used to deposit LP tokens into the MASTER_CHEF_V2 contract. MASTER_CHEF_V2 will be approved before depositing, and `approved[lpToken]` will be set to true after approval, and will never be approved again in the future. Although the approved amount is uint256, the allowance may still be exhausted in the future, and after the allowance is exhausted, it will no longer be possible to approve. This will make the proxy contract unavailable.

Code location: contracts/CrossFarmingProxy.sol

```
    function deposit(
        uint256 _pid,
        uint256 _amount,
        uint64 _nonce
    ) external nonReentrant onlyFactory onlyNotFallback(_pid, _nonce) {
        address lpToken = MASTER_CHEF_V2.lpToken(_pid);
        if (!approved[lpToken]) {
            IERC20(lpToken).approve(address(MASTER_CHEF_V2), type(uint256).max);
            approved[lpToken] = true;
        }

        ...
    }
```

Solution

It is recommended to check whether the remaining allowance meets the required number of tokens deposited, and re-approve if not. As a best practice, each time a deposit is made, only the amount is approved for the amount deposited.

Status

Fixed

[N10] [Medium] Incorrect interface call

Category: Design Logic Audit

Content

In the executeMessage function of the CrossFarmingReceiver contract, when the msgType is Withdraw and EmergencyWithdraw, the sender contract that is not deployed in the BSC chain is called by mistake.

Code location: contracts/CrossFarmingReceiver.sol

```
    function executeMessage(
        address _sender,
        uint64 _srcChainId,
        bytes calldata _message,
        address // executor who called the MessageBus execution function
    ) external payable override onlyMessageBus returns (ExecutionStatus) {
        ...
        } else if (request.msgType == DataTypes.MessageTypes.Withdraw) {
            proxy.withdraw(request.pid, request.amount, request.nonce);

            // send ack message back.
            (bool success, ) = _sender.call{value:
IMessageBus(messageBus).calcFee(_message)}(
                abi.encodeWithSignature("sendFarmMessage(bytes)", _message)
            );

            require(success, "send withdraw ack message failed");
        } else if (request.msgType == DataTypes.MessageTypes.EmergencyWithdraw) {
            proxy.emergencyWithdraw(request.pid, request.nonce);

            // send ack message back.
            (bool success, ) = _sender.call{value:
IMessageBus(messageBus).calcFee(_message)}(
                abi.encodeWithSignature("sendFarmMessage(bytes)", _message)
            );
            require(success, "send emergencyWithdraw ack message failed");
        }

        ...
    }
```

Solution

It is recommended to reconfirm the intended design.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002209230001	SlowMist Security Team	2022.09.19 - 2022.09.23	Passed

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 high risk, 1 medium risk, 4 low risks, 4 suggestions. And 1 low risk, 1 suggestion were ignored; 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.

SLOWMIST

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist