Security Audit Report for PancakeSwap Cross Farming Contracts

Date: September 28, 2022

Version: 1.0

Contact: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.09.28 23:09:22 +08'00'

Contents

Report Manifest

Item	Description
Client	PancakeSwap Cross Farming
Target	PancakeSwap Cross Farming Contracts

Version History

Version	Date	Description
1.0	September 28, 2022	First Release

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 5 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the cross farming contracts of the PancakeSwap Protocol [1]. This project allows users depositing their LP tokens on EVM-compatible chains to the MasterChefV2 contract on the Binance Smart Chain (aka BSC) network by utilizing the cBridge SGN network as the cross-chain message forwarder. Note that the audit scope is limited to contracts under the `projects/cross-chain/contracts/` folder, while other contracts and files are out of the scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
pancake-contracts	`Version 1`	`e77daef1ae954d6fd96d3f4d41c5327ce5125d83`
	`Version 2`	`f56a59f56083d684b281a247670af9c600d923ee`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

[1] The `features/cross-chain` branch of repo `https://github.com/chefcooper/pancake-contracts`.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we find **four** potential issues. Besides, we also have **two** recommendations.

- Medium Risk: 1
- Low Risk: 3
- Recommendation: 2

ID	Severity	Description	Category	Status
1	Low	Potential improper gas price estimation	DeFi Security	Acknowledged
2	Low	Unchecked valid range for Chainlink prices	DeFi Security	Acknowledged
3	Low	Being unable to disable pools	DeFi Security	Acknowledged
4	Medium	Potential double deposit or withdrawal in the fallback situation	DeFi Security	Fixed
5	-	Remove unused inherited contract	Recommendation	Fixed
6	-	Follow the `check-effect-interactions` pattern	Recommendation	Fixed

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Potential improper gas price estimation

Severity Low

Status Acknowledged

Introduced by `Version 1`

Description The cross-farming project relies on a `CrossFarmingSender` contract to send messages to the cross-chain forwarder contract on the source chain. In the `sendFarmMessage` function of the `CrossFarmingSender` contract, the transaction fee in the destination chain is estimated by multiplying the gas price of the current transaction (i.e., in the source chain) with the estimated gas limit in the destination chain. This estimation would be improper because different chains may have different regular gas prices.

```
119    function sendFarmMessage(bytes calldata _message) external payable onlyVault {
120        // decode the message
121        DataTypes.CrossFarmRequest memory request = abi.decode((_message), (DataTypes.
               CrossFarmRequest));
122
123        // ETH/USD price
124        int256 ethPrice = _getPriceFromOracle(Feeds.ETHUSD);
125        // BNB/USD price
126        int256 bnbPrice = _getPriceFromOracle(Feeds.BNBUSD);
127
128        require(bnbPrice > 0 && ethPrice > 0, "Abnormal prices");
129
130        uint256 exchangeRate = (uint256(bnbPrice) * EXCHANGE_RATE_PRECISION) / uint256(ethPrice);
131        // msgbus fee price by native token
132        uint256 msgBusFee = IMessageBus(messageBus).calcFee(_message);
```

```
133
134         uint256 totalFee = msgBusFee +
135             // destTxFee
136             (tx.gasprice *
137                 estimateGaslimit(Chains.BSC, request.account, request.msgType) *
138                 exchangeRate *
139                 txFeeFloatRate) /
140             (EXCHANGE_RATE_PRECISION * FLOAT_RATE_PRECISION);
141
142         // BNB change fee for new BNB user
143         if (!is1st[request.account]) {
144             totalFee += (BNB_CHANGE * exchangeRate) / EXCHANGE_RATE_PRECISION;
145             is1st[request.account] = true;
146         }
147
148         if (request.msgType >= DataTypes.MessageTypes.Withdraw) {
149             totalFee += (// executor call fee(Ack call on this contract 'executeMessage' interface)
150             tx.gasprice *
151                 estimateGaslimit(Chains.EVM, request.account, request.msgType) +
152                 // withdraw ack msg messageBus fee on BSC chain
153                 ((msgBusFee * exchangeRate) / EXCHANGE_RATE_PRECISION));
154         }
```

Listing 2.1: CrossFarmingSender.sol

Impact The estimation of the gas price in the destination chain is incorrect.

Suggestion The contract may charge insufficient or extraneous transaction fees.

Feedback from the Developers Yes, we can't estimate accurate gas price, that's the reality, that's why we add the `compensationRate` state variable to hedge the gas price fluctuate risk.

2.1.2 Unchecked valid range for Chainlink prices

Severity Low

Status Acknowledged

Introduced by `Version 1`

Description The `CrossFarmingSender` contract uses Chainlink prices to convert from ETH (or the native tokens of other EVM-compatible chains if supported) to BNB (i.e., the native token of BSC). The prices are checked if they are updated within a reasonable delay, but the valid range (i.e., the `minAnswer` and `maxAnswer` range provided by the price oracle) is not checked. It is recommended to check the validity for the prices retrieved from Chainlink to prevent attacks like the LUNA incident of the Venus Protocol [1].

```
305     function _getPriceFromOracle(Feeds _feed) internal view returns (int256) {
306         (, int256 price, , uint256 timestamp, ) = oracle[_feed].latestRoundData();
307         require(timestamp + oracleUpdateBuffer[_feed] >= block.timestamp, "out of date oracle data"
                );
308         return price;
309     }
```

[1] https://medium.com/venusprotocol/venus-protocol-official-statement-regarding-luna-6eb45c3cb058.

Listing 2.2: CrossFarmingSender.sol

Impact Invalid prices may lead to unexpected behaviors.

Suggestion Check the valid range for the Chainlink price feeds.

Feedback from the Developers Got your concern, we can't predict what will happen in the market, so we don't want to set upper limit and lower limit, we accept the price of the market and do not want to block cross-chain transactions due to this factor

2.1.3 Being unable to disable pools

Severity Low

Status Acknowledged

Introduced by `Version 1`

Description The variable named `whitelistPool` in the `CrossFarmingVault` contract is used to allow depositing the trustworthy LP tokens. However, once an LP token is added to the vault as a pool, it is unable to remove or disable this pool anymore.

```
141  function add(IERC20 _lpToken, uint256 _mcv2PoolId) public onlyOwner {
142    require(!exists[_lpToken], "Existed token");
143    require(whitelistPool[_mcv2PoolId], "Not whitelist pool");
144    require(poolMapping[_mcv2PoolId] == 0, "MCV2 pool already matched");
145    require(_lpToken.balanceOf(address(this)) >= 0, "Not ERC20 token");
146
147    // add poolInfo
148    poolInfo.push(PoolInfo({lpToken: _lpToken, mcv2PoolId: _mcv2PoolId, totalAmount: 0}));
149
150    // update mappping
151    exists[_lpToken] = true;
152    poolMapping[_mcv2PoolId] = poolInfo.length - 1;
153
154    emit AddedPool(address(_lpToken), _mcv2PoolId);
155}
```

Listing 2.3: CrossFarmingVault.sol

Impact The added pools could not be disabled or removed.

Suggestion Add the functionality to remove pools.

Feedback from the Developers Yes, we can't disable the token, after we added the LP token to the pool in the vault contract, that means there is a mirror LP token on `MasterChefV2` pool, the LP token in `MasterChefV2` also can't be removed or disabled. Because that means there are many users who staked LP token in this pool, we can't stop it. We can set the `MasterChefV2` farm pool alloc point to zero if we want to stop this pool, that means there is no `CAKE` reward.

2.1.4 Potential double deposit or withdrawal in the fallback situation

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `CrossFarmingProxy` contract is used to delegate user operations for every user on BSC. Specifically, to solve the problem that cross-farming transactions may revert for some reason, the contract provides a mechanism that a special character can force a deposit or withdrawal. After that, the contract will set a value stored in the state variable named `fallbackNonce` as true, to mark the nonce is forced to execute.

```
178    function fallbackDeposit(uint256 _pid, uint64 _nonce) external onlyFactory onlyNotFallback(
           _pid, _nonce) {
179        // only mark fallback nonce used
180        fallbackNonce[_pid][_nonce] = true;
181
182        emit FallbackDeposit(user, _pid, _nonce);
183    }
```

Listing 2.4: CrossFarmingProxy.sol

However, the `withdraw` function of the `CrossFarmingProxy` contract does not update `fallbackNonce`, which leads to a potential attack surface. Specifically, if there is a completely reverted transaction to the `MessageBus` (i.e., the cross-chain forwarder contract of cBridge SGN network to relay the withdrawal message), making the operator (probably a bot) be triggered to call the `fallbackWithdraw` function.

```
107    function withdraw(
108        uint256 _pid,
109        uint256 _amount,
110        uint64 _nonce
111    ) external payable nonReentrant onlyFactory onlyNotFallback(_pid, _nonce) {
112        require(userInfo[_pid] >= _amount && _amount > 0, "Insufficient token");
113
114        // withdraw from MCV2 pool
115        MASTER_CHEF_V2.withdraw(_pid, _amount);
116        // burn LP token which equal to withdraw amount
117        IMintable(MASTER_CHEF_V2.lpToken(_pid)).burn(_amount);
118        // send CAKE reward
119        _safeTransfer(user);
120
121        // update state
122        userInfo[_pid] -= _amount;
123
124        if (_nonce > latestNonce[_pid]) {
125            latestNonce[_pid] = _nonce;
126        }
127
128        emit Withdraw(user, _pid, _amount, _nonce);
129    }
```

Listing 2.5: CrossFarmingProxy.sol

```
187    function fallbackWithdraw(
188        uint256 _pid,
189        uint256 _amount,
```

```
190         uint64 _nonce
191     ) external onlyFactory onlyNotFallback(_pid, _nonce) {
192         require(userInfo[_pid] >= _amount && _amount > 0, "Insufficient token");
193
194         // withdraw from MCV2 pool
195         MASTER_CHEF_V2.withdraw(_pid, _amount);
196         // burn LP token which equal to withdraw amount
197         IMintable(MASTER_CHEF_V2.lpToken(_pid)).burn(_amount);
198         // send CAKE reward
199         _safeTransfer(user);
200
201         userInfo[_pid] -= _amount;
202         // mark nonce used
203         fallbackNonce[_pid][_nonce] = true;
204
205         emit FallbackWithdraw(user, _pid, _amount, _nonce);
206     }
```

Listing 2.6: CrossFarmingProxy.sol

The attack sequence is as follows:

1. The attacker first needs to send a failed transaction to relay the withdrawal message to the `MessageBus`, and this transaction should be completely reverted.
2. The failure transaction will trigger the operator to initiate a call to the `fallbackWithdraw` function.
3. The attacker could front-running the `fallbackWithdraw` function using a call to the `withdraw` function.

Note that both invocations would succeed because the `withdraw` function does NOT update `fallbackNonce`.

Impact In the case of the failed cross-chain request triggering the fallback logic, the fallback invocation initiated by the operator can be front-run.

Suggestion Refactor the nonce-related logic to mitigate the potential risk.

Feedback from the Developers Yes, that's what we concern. We have replace the `fallbackNonce` to the `usedNonce`, which means all `deposit`/`withdraw`/`emergencywithdraw` and fallback functions all update the same used nonce.

2.2 Additional Recommendation

2.2.1 Remove unused contract

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `CrossFarmingSender` contract does not need to inherit from `MessageSenderApp`, because the corresponding functionalities are not used.

```
13  /// @title A cross chain contract for users from other EVM chain participate Pancakeswap MCV2
        farm pool in BSC chain.
14  /// @dev deployed on EVM chain(source chain, not BSC chain).
15  contract CrossFarmingSender is MessageSenderApp, MessageReceiverApp {
16      using SafeERC20 for IERC20;
17
```

```
18      // oracle data feeds
19      enum Feeds {
20          BNBUSD,
21          ETHUSD
22      }
```

Listing 2.7: CrossFarmingSender.sol

Impact May cause extra gas usage.

Suggestion Remove the unused contract.

2.2.2 Follow the `check-effect-interactions` pattern

Status Fixed in `Version 2`

Introduced by `Version 1`

Description It is recommended to follow the `check-effect-interactions` pattern in the contract code, which specifies that interactions like token transfers should be placed after the checks and effects on the contract state, including:

1. The `ackWithdraw` function in the `CrossFarmingVault` contract.
2. The `ackEmergencyWithdraw` function in the `CrossFarmingVault` contract.
3. The `sendFarmMessage` function in the `CrossFarmingSender` contract.

Impact N/A

Suggestion Revise code accordingly.