Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.10.10, the SlowMist security team received the PancakeSwap team's security audit application for PancakeSwap - MOVE, 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	Replay Attack Audit	-
3	Flashloan Attack Audit	-
4	Denial of Service 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
		Show Coding Security Audit
		Block data Dependence Security Audit
		Explicit Visibility of Functions Audit
8	Gas Optimization Audit	-
9	Design Logic Audit	-
10	Arithmetic Accuracy Deviation Audit	-
11	Capability Security Usage Audit	-
12	Resource Security Usage Audit	-

3 Project Overview

3.1 Project Introduction

Audit Version:

https://github.com/pancakeswap/aptos-contracts

commit: fc315ef77b8645faf1e64de1a895bf591a46453a

Fixed Version:

https://github.com/pancakeswap/aptos-contracts

commit: 1793c456508932d9d50c17891f8f90ae9d0a6c3e

Audit Scpoe:

- pancake-swap/sources/
- pancake-swap-storage/sources/

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Repeatable initialization issue	Design Logic Audit	Low	Fixed
N2	Gas optimization	Gas Optimization Audit	Suggestion	Fixed
N3	Architecture optimization	Design Logic Audit	Suggestion	Fixed
N4	Assertion flaw issue	Design Logic Audit	Suggestion	Fixed
N5	Not checked if pair has been created	Design Logic Audit	Low	Fixed
N6	Redundant code	Others	Suggestion	Fixed
N7	k value check	Design Logic Audit	Suggestion	Confirming
N8	Risk of excessive authority	Excessive Authority Audit	Medium	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:

router			
Function Name	Visibility	Acquires	Modifier
create_pair	Public	-	-
add_liquidity	Public	-	entry
remove_liquidity	Public	-	entry
add_swap_event_internal	-	-	-
swap_exact_input	Public	-	entry
swap_exact_output	Public	-	entry
get_intermediate_output	-	-	-
get_intermediate_output_x_to_exact_y	-	-	-
swap_exact_input_double_internal	-	-	-
swap_exact_input_doublehop	Public	-	-
swap_exact_output_double_internal	-	-	-
swap_exact_output_doublehop	Public	-	entry
swap_exact_input_triple_internal	-	-	-
swap_exact_input_triplehop	Public	-	entry
swap_exact_output_triple_internal	-	-	-
swap_exact_output_triplehop	Public	-	entry

router			
register_lp	Public	-	entry
register_token	Public	-	entry

swap			
Function Name	Visibility	Acquires	Modifier
init_module	-	-	-
create_pair	Public(friend)	SwapInfo	-
register_lp	Public	-	-
is_pair_created	Public	-	-
lp_balance	Public	-	-
total_lp_supply	Public	-	-
token_reserves	Public	TokenPairReserve	-
token_balances	Public	TokenPairMetadata	-
check_coin_store	Public	-	-
add_liquidity	Public(friend)	TokenPairReserve, TokenPairMetadata, PairEventHolder	-
add_swap_event	Public(friend)	PairEventHolder	-
add_liquidity_direct	-	TokenPairReserve, TokenPairMetadata	-
remove_liquidity	Public(friend)	TokenPairMetadata, TokenPairReserve, PairEventHolder	-
remove_liquidity_direct	-	TokenPairMetadata, TokenPairReserve	-

swap			
quote_x_to_y_after_fees	-	TokenPairMetadata	-
swap_exact_x_to_y	Public(friend)	TokenPairReserve, TokenPairMetadata	-
swap_exact_x_to_y_direct	Public(friend)	TokenPairReserve, TokenPairMetadata	-
swap_x_to_exact_y	Public(friend)	TokenPairReserve, TokenPairMetadata	-
swap_x_to_exact_y_direct	Public(friend)	TokenPairReserve, TokenPairMetadata	-
quote_y_to_x_after_fees	-	TokenPairMetadata	-
swap_exact_y_to_x	Public(friend)	TokenPairReserve, TokenPairMetadata	-
swap_y_to_exact_x	Public(friend)	TokenPairReserve, TokenPairMetadata	-
swap_y_to_exact_x_direct	Public(friend)	TokenPairReserve, TokenPairMetadata	-
swap_exact_y_to_x_direct	Public(friend)	TokenPairReserve, TokenPairMetadata	-
swap	-	TokenPairReserve, TokenPairMetadata	-
mint	-	TokenPairReserve, TokenPairMetadata	-
burn	-	TokenPairMetadata, TokenPairReserve	-
update	-	-	-
mint_lp_to	-	-	-
mint_lp	-	-	-
burn_lp	-	-	-
transfer_lp_coin_in	-	TokenPairMetadata	-

swap			
deposit_x	-	TokenPairMetadata	-
deposit_y	-	TokenPairMetadata	-
extract_x	-	-	-
extract_y	-	-	-
transfer_x	-	-	-
transfer_y	-	-	-
mint_fee	-	-	-
set_admin	Public	SwapInfo	entry
set_fee_to	Public	SwapInfo	entry
withdraw_fee	Public	SwapInfo, TokenPairMetadata	entry
upgrade_swap	Public	SwapInfo	entry

swap_utils			
Function Name	Visibility	Acquires	Modifier
get_amount_out	Public	-	-
get_amount_in	Public	-	-
quote	Public	-	-
get_token_info	Public	-	-
compare_struct	-	-	-
get_smaller_enum	Public	-	-

swap_utils			
get_greater_enum	Public	-	-
get_equal_enum	Public	-	-
sort_token_type	Public	-	-

4.3 Vulnerability Summary

[N1] [Low] Repeatable initialization issue

Category: Design Logic Audit

Content

In the swap module RESOURCE_ACCOUNT can initialize SwapInfo through the init_storage function, but the function does not check for repeated initialization.

Code location: sources/swap/swap.move

```
    public entry fun init_storage(sender: &signer) {
        let resource_account = signer::address_of(sender);
        assert!(resource_account == RESOURCE_ACCOUNT, ERROR_NOT_RESOURCE_ACCOUNT);
        let signer_cap = resource_account::retrieve_resource_account_cap(sender,
RESOURCE_CONTAINER_ACCOUNT);
        let resource_signer = account::create_signer_with_capability(&signer_cap);
        move_to(&resource_signer, SwapInfo{
            signer_cap,
            fee_to: ZERO_ACCOUNT,
            admin: DEFAULT_ADMIN,
            pair_created: account::new_event_handle<PairCreatedEvent>
(&resource_signer),
        });
    }
```

Solution

It is recommended to disallow repeatable initialization of the init_storage function

Status

Fixed

[N2] [Suggestion] Gas optimization

Category: Gas Optimization Audit

Content

In the swap module, the add_liquidity function is used to add liquidity and return the remaining tokens to the user.

But without checking whether the token value to be returned is greater than 0, the coin::deposit function is called. If the returned token value is 0, this will cause unnecessary gas consumption.

Code location: sources/swap/swap.move

```
    public(friend) fun add_liquidity<X, Y>(
        sender: &signer,
        amount_x: u64,
        amount_y: u64
    ): (u64, u64, u64) acquires TokenPairReserve, TokenPairMetadata, PairEventHolder
{
        let (a_x, a_y, coin_lp, fee_amount, coin_left_x, coin_left_y) =
add_liquidity_direct(coin::withdraw<X>(sender, amount_x), coin::withdraw<Y>(sender,
amount_y));
        let sender_addr = signer::address_of(sender);
        let lp_amount = coin::value(&coin_lp);
        check_coin_store<LPToken<X, Y>>(sender);
        coin::deposit(sender_addr, coin_lp);
        coin::deposit(sender_addr, coin_left_x);
        coin::deposit(sender_addr, coin_left_y);

        ...
    }
```

Solution

It is recommended to check whether the value of the token to be returned is greater than 0.

Status

Fixed

[N3] [Suggestion] Architecture optimization

Category: Design Logic Audit

Content

The storage of LP tokens is realized through storage.move in the protocol. But the creation of Pair is realized through the create_pair of the swap module and the resource storage is carried out by TokenPairMetadata and TokenPairReserve. So LPToken can be implemented directly in swap module without having to implement it in storage.move separately.

Code location: sources/storage.move

```
module pancake_swap_storage::storage {
    /// The LP Token type
    struct LPToken<phantom X, phantom Y> has key {}
}
```

Solution

It is recommended to implement LPToken in the swap module.

Status

Fixed

[N4] [Suggestion] Assertion flaw issue

Category: Design Logic Audit

Content

In the swap module, the mint function is used to mint LP tokens when liquidity is added. When adding liquidity for the first time, the liquidity amount needs to meet MINIMUM_LIQUIDITY. If the MINIMUM_LIQUIDITY is not met, the

transaction will be revet. In this case the protocol will throw an overflow error instead of

ERROR_INSUFFICIENT_LIQUIDITY_MINTED.

Code location: sources/swap/swap.move

```
    fun mint<X, Y>(): (coin::Coin<LPToken<X, Y>>, u128) acquires TokenPairReserve,
TokenPairMetadata {
        ...
        let liquidity = if (total_supply == 0u128) {
            let l = math::sqrt(amount_x * amount_y) - MINIMUM_LIQUIDITY;
            mint_lp_to<X, Y>(RESOURCE_ACCOUNT, (MINIMUM_LIQUIDITY as u64),
&metadata.mint_cap);
            l
        } else {
            math::min(amount_x * total_supply / (reserves.reserve_x as u128),
amount_y * total_supply / (reserves.reserve_y as u128))
        };

        assert!(liquidity > 0u128, ERROR_INSUFFICIENT_LIQUIDITY_MINTED);

        ...
    }
```

Solution

It is recommended to add liquidity for the first time to check whether the added liquidity amount meets

MINIMUM_LIQUIDITY.

Status

Fixed

[N5] [Low] Not checked if pair has been created

Category: Design Logic Audit

Content

In the router module contract, users can remove liquidity and exchange tokens through the functions

remove_liquidity, swap_exact_input, swap_exact_output, swap_exact_input_doublehop,

swap_exact_output_doublehop, swap_exact_input_triplehop and swap_exact_output_triplehop respectively, but do not check whether a pair is created first.

Code location: sources/router.move

Solution

It is recommended to check whether a pair has been created before performing liquidity withdrawal and token swap.

Status

Fixed

[N6] [Suggestion] Redundant code

Category: Others

Content

In the Swap module, the quote_x_to_y_after_fees, quote_y_to_x_after_fees, transfer_x and transfer_y functions are all internal functions, but no other public functions call them.

Code location: sources/swap.move

```
    fun quote_x_to_y_after_fees<X, Y>(
        amount_x_in: u64
    ): u64 acquires TokenPairMetadata {
        let (x_balance, y_balance) = token_balances<X, Y>();
        swap_utils::get_amount_out(amount_x_in, x_balance, y_balance)
    }

    fun quote_y_to_x_after_fees<X, Y>(
        amount_y_in: u64
    ): u64 acquires TokenPairMetadata {
        let (x_balance, y_balance) = token_balances<X, Y>();
        swap_utils::get_amount_out(amount_y_in, y_balance, x_balance)
    }

    fun transfer_x<X, Y>(amount: u64, recipient: address, metadata: &mut
TokenPairMetadata<X, Y>) {
        let coins = extract_x(amount, metadata);
        coin::deposit(recipient, coins);
```

```
    }

    fun transfer_y<X, Y>(amount: u64, recipient: address, metadata: &mut
TokenPairMetadata<X, Y>) {
        let coins = extract_y(amount, metadata);
        coin::deposit(recipient, coins);
    }
```

Solution

It is recommended to remove redundant functions.

Status

Fixed

[N7] [Suggestion] k value check

Category: Design Logic Audit

Content

During the swap process, it is necessary to check whether the multiplication of the token balance of the pair after the swap is strictly greater than or equal to the k value. However, due to the fee charged during the swap process, in theory, the multiplication of the token balance of the pair after swap must be strictly greater than the k value. While using u256 avoids close rounding errors it is still not necessary to check if it is equal to the k value.

Code location: sources/swap.move

```
    fun swap<X, Y>(
        amount_x_out: u64,
        amount_y_out: u64
    ): (coin::Coin<X>, coin::Coin<Y>) acquires TokenPairReserve, TokenPairMetadata {
        ...
        // balance_x_adjusted * balance_y_adjusted >= k
        let compare = u256::compare(&u256::mul(balance_x_adjusted,
balance_y_adjusted), &k);
        assert!(compare == u256::get_greater_than() || compare == u256::get_equal(),
ERROR_K);
```

```
        ...
    }
```

Solution

It is recommended to check whether the multiplication of the swap pair token balance is strictly greater than the k value.

Status

Confirming

[N8] [Medium] Risk of excessive authority

Category: Excessive Authority Audit

Content

In the swap module contract, the admin role can call the upgrade_swap function to upgrade the entire contract. If administrator privileges are stolen, it may have an impact on the normal operation of the contract.

Code location: sources/swap.move

```
    public entry fun upgrade_swap(sender: &signer, metadata_serialized: vector<u8>,
code: vector<vector<u8>>) acquires SwapInfo {
        let sender_addr = signer::address_of(sender);
        let swap_info = borrow_global<SwapInfo>(RESOURCE_ACCOUNT);
        assert!(sender_addr == swap_info.admin, ERROR_NOT_ADMIN);
        let resource_signer =
account::create_signer_with_capability(&swap_info.signer_cap);
        code::publish_package_txn(&resource_signer, metadata_serialized, code);
    }
```

Solution

The short-term recommendation is to manage the admin account with multiple signatures, and in the long term we recommend giving the admin role to the timelock contract or the community.

Status

Confirmed; After communicating with the project team, the project team stated that we already wrote multi sign wallet SC to control the admin role and also support time lock.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002210190001	SlowMist Security Team	2022.10.10 - 2022.10.19	Medium Risk

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 medium risk, 2 low risks, and 5 suggestions. And 1 medium-risk vulnerability was confirmed, 1 suggestion is being confirmed; 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