PancakeSwap

Pancake Swap

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen r@osec.io

Harrison Green hgarrereyn@osec.io

Contents

01 | Executive Summary

Overview

Pancake Swap engaged OtterSec to perform an assessment of the `pancake-swap` program. This assessment was conducted between October 10th and October 24th, 2022.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation. We delivered final confirmation of the patches November 4th, 2022.

Key Findings

Over the course of this audit engagement, we produced 6 findings total.

In particular, we identified an issue with the way token structs were being compared to generate pairs (OS-PAN-ADV-00).

We also made recommendations around clean coding practices and general security recommendations. These recommendations serve to clarify the purpose and logic of functions in the program and can help prevent future security vulnerabilities stemming from misunderstanding or needlessly entangled code.

Overall, the Pancake Swap team was responsive to feedback and great to work with.

02 | Scope

The source code was delivered to us in a git repository at github.com/pancakeswap/aptos-contracts.

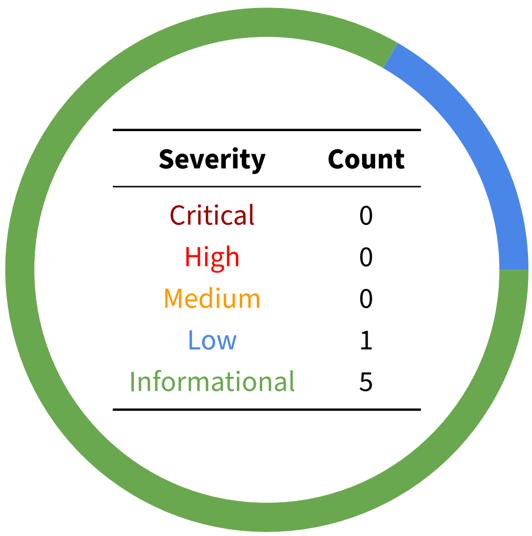
A brief description of the programs is as follows.

Name	Description
pancake-swap	Token swap program

03 | Findings

Overall, we report 6 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	0
Low	1
Informational	5

04 | Vulnerabilities

Here we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-PAN-ADV-00	Low	Resolved	Faulty token struct comparison

OS-PAN-ADV-00 [low] [resolved] | Faulty Token Struct Comparison

Description

In order to construct a deterministic ordering of two tokens in a swap pair, it is necessary to be able to compare them. The current implementation concatenates the address, module, and struct names into a vector and invokes `compare_u8_vector`.

This implementation generates collisions for certain token structs that should not collide. For example, the following two structs would generate the same comparison string:

```move
module address::FO {
    struct OBAR {}
}

module address::FOO {
    struct BAR {}
}
```

Both structs generate the string: `addressFOOBAR`. The protocol will incorrectly reject this swap pair from being constructed.

Remediation

Use `type_info::type_name` to generate a fully qualified name for each struct. In the example above, the two token structs would produce the following names respectively:

1. `address::FO::OBAR`
2. `address::FOO::BAR`

Patch

Fixed in 0a0ead7.

05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-PAN-SUG-00	Consolidate TokenPairMetadata and TokenPairReserve
OS-PAN-SUG-01	Rename `check_coin_store`
OS-PAN-SUG-02	Use arguments instead of global changes.
OS-PAN-SUG-03	Remove the lp field in TokenPairMetadata
OS-PAN-SUG-04	Clarify arguments in `create_pair`

OS-PAN-SUG-00 | Consolidate Token Metadata

Description

Information for a token pair is stored in two separate metadata structs:

```move
/// Stores the metadata required for the token pairs
struct TokenPairMetadata<phantom X, phantom Y> has key {
    /// The admin of the token pair
    creator: address,
    /// fee amount , record fee amount which is not withdrawed
    fee_amount: coin::Coin<LPToken<X, Y>>,
    /// It's reserve_x * reserve_y, as of immediately after the most
    ↪  recent liquidity event
    k_last: u128,
    /// T0 token balance
    balance_x: coin::Coin<X>,
    /// T1 token balance
    balance_y: coin::Coin<Y>,
    /// Mint capacity of LP Token
    mint_cap: coin::MintCapability<LPToken<X, Y>>,
    /// Burn capacity of LP Token
    burn_cap: coin::BurnCapability<LPToken<X, Y>>,
    /// Freeze capacity of LP Token
    freeze_cap: coin::FreezeCapability<LPToken<X, Y>>,
}

/// Stores the reservation info required for the token pairs
struct TokenPairReserve<phantom X, phantom Y> has key {
    reserve_x: u64,
    reserve_y: u64,
    block_timestamp_last: u64
}
```

Some information between the two structs is redundant. For example `reserve_x` is intended to represent the current value of `balance_x` and same for `reserve_y`. Tracking these fields in multiple places presents the chance of accidentally de-syncing leading to a potential security issue.

Remediation

Use one struct to hold swap pair metadata and do not duplicate information across multiple fields. For the case presented here, remove `TokenPairReserve` and use `balance_x` and `balance_y` directly when it is necessary to check their values.

OS-PAN-SUG-01 [resolved] | Rename check_coin_store

Description

The function `check_coin_store` creates a coin store for an account if it does not already exist. However, the current name implies it is a type of assertion. To prevent accidental misuse, consider renaming this function.

Patch

Renamed to `check_or_register_coin_store`.

OS-PAN-SUG-02 | Use Arguments Instead of Global Changes

Description

In certain functions such as `mint`, information such as the amount to mint is obtained by observing immediate changes in global state (caused by predecessor functions):

MOVE

```
fun mint<X, Y>(): (coin::Coin<LPToken<X, Y>>, u64) acquires
    ↪  TokenPairReserve, TokenPairMetadata {
    let metadata = borrow_global_mut<TokenPairMetadata<X,
    ↪  Y>>(RESOURCE_ACCOUNT);
    let (balance_x, balance_y) = (coin::value(&metadata.balance_x),
    ↪  coin::value(&metadata.balance_y));
    let reserves = borrow_global_mut<TokenPairReserve<X,
    ↪  Y>>(RESOURCE_ACCOUNT);
    let amount_x = (balance_x as u128) - (reserves.reserve_x as u128);
    let amount_y = (balance_y as u128) - (reserves.reserve_y as u128);
    ...
}
```

This type of logic heavily depends on the sequence of function calls may lead to issues during refactoring. If `mint` is called from other contexts, the global state may not be changed in the same way.

Remediation

Do not use changes in global state to pass information to subroutines. Instead pass these values directly as arguments to the function.

OS-PAN-SUG-03 [resolved] | Remove Unnecessary Parameter

Description

The `TokenPairMetadata` struct contains an unnecessary `lp` field. This field is used during `remove_liquidity` as an intermediary but serves no purpose and can be removed:

```move
fun remove_liquidity_direct<X, Y>(
    liquidity: coin::Coin<LPToken<X, Y>>,
): (coin::Coin<X>, coin::Coin<Y>, u64) acquires TokenPairMetadata,
    ↪  TokenPairReserve {
    transfer_lp_coin_in<X, Y>(liquidity);

    let (coins_x, coins_y, fee_amount) = burn<X, Y>();

    (coins_x, coins_y, fee_amount)
}
```

Coins are transferred to `TokenPairMetadata.lp` in `transfer_lp_coin_in` and then immediately burned in the subsequent `burn` call.

Patch

Fixed in 3597098.

OS-PAN-SUG-04 [resolved] | Clarify Arguments in create_pair

Description

The first argument in `create_pair` is called `admin` but it is not authenticated:

```move
public(friend) fun create_pair<X, Y>(
    admin: &signer,
) acquires SwapInfo {
    ...
}
```

To prevent misuse, consider renaming this to something that indicates the correct level of privilege.

Patch

Fixed in 537997c.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities that immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority or access control validation • Improperly designed economic incentives leading to loss of funds
High	Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities that could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation