BAPTSWAP

Audit Report

MOVEBIT

contact@movebit.xyz

https://twitter.com/movebit_

Mon Dec 18 2023

BAPTSWAP Audit Report

1 Executive Summary

1.1 Project Information

Description	BAPTSWAP is the decentralized exchange, powered by BAPT LABS
Type	Dex
Auditors	MoveBit
Timeline	Wed Nov 15 2023 – Mon Dec 18 2023
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/BAPTSWAP/V2-core
Commits	37005ff50d6a53a84ba23e2fc50ecf4d6f9d7691 7e3fe20cea068c731bc0d76fc7d654455a454d1a 605766ef22b5234159631675a9e9f246a28676e7 1b6d6d9fd314b92378eac6fccc8840f706d67c51 2b5aae52f933f74a0650b20265830949f9b49927 c9aa8f1b0544a1ad2adac41853d87dc5f3677c8d 01f0e05dc25cb9585a66938426103a7434336586 ba48a64b1f15cacdd86346a3599d031990269a1d 39458ecb7f5e1f24bc1c3e9d0a1640da1080adf3 be5b720e4c5e6fa9cbd14fb93e3788c94c5779a4

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
MOV	Move.toml	dd7ff81230c5ff8bbfdf19c2db15c048451e883e
RV2	sources/router_v2.move	a03fceaa23c8b5b9f5aa87487024bf216ee412e3
SV2	sources/swap_v2.move	3c96c00c858c4aebad0eb0fa961224d76344f681
CON	sources/utils/constants.move	0c0875a3d98cedd11c0b640a348bc3986552f678
UTI	sources/utils/utils.move	31a83aba79158b5e29595d57df712b0380bdf879
ERR	sources/utils/errors.move	2c83e25832d068dd40ae299e799046dc87c89f46
SUV2	sources/utils/swap_utils_v2.move	b722cc5ad6f971b709a62c7e2b3797cdf4b1879e
ADM	sources/admin.move	065fe17e946df2c9913c5fa8a5682b8e94cc4778
STA	sources/stake.move	4952e1a37c819f4d9fd3c10581f2cc22b5927cf2
FOT	sources/fee_on_transfer.move	ee0ed69995eb7722623a922d8393ff6f0e5f36c4

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	37	35	2
Informational	5	5	0
Minor	14	13	1
Medium	3	3	0
Major	14	13	1
Critical	1	1	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by BAPTSWAP to identify any potential issues and vulnerabilities in the source code of the BAPTSWAP smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 37 issues of varying severity, listed below.

ID	Title	Severity	Status
ADM-1	The Distinction is Lacking When Setting Admin And `treasury_address`	Major	Fixed
ADM-2	Lack of Access Control	Major	Fixed
ADM-3	Permission Conflict	Major	Fixed
ADM-4	Sensitive Operation Lacks Event	Minor	Fixed
ADM-5	The Comments Do Not Match the Actual Functionality	Informational	Fixed
ADM-6	The Comment in the `claim_treasury_previliges()` Function is Incorrect	Informational	Fixed
FOT-1	Functions with Similar Functionality	Minor	Fixed
RV2-1	Nonexistent Token Pair	Major	Fixed
RV2-2	The Admin is Unable to Update the Liquidity Fee and Treasury Fee	Major	Fixed
RV2-3	Unused Private Function	Minor	Fixed

RV2-4	Code Refactoring Suggestions in `router_v2` Module	Minor	Fixed
RV2-5	The Specification for Assert Statements	Minor	Fixed
STA-1	Updating Magnified Dividends Per Share during Unstaking is Incorrect	Major	Fixed
STA-2	Direct Invocation Risk in `unstake_tokens()` and `claim_rewards()` Functions in `stake` Module	Medium	Fixed
STA-3	Optimization through Consolidating `claim_rewards()` and `unstake_tokens()` Functions	Minor	Fixed
SV2-1	The Constant Product Rule is Compromised, Enabling Pool Draining	Critical	Fixed
SV2-2	There is No Slippage Protection During The Distribution of DEX Fees	Major	Fixed
SV2-3	Infinite Recursion in `distribute_dex_fees()` Leading to Transaction Failure	Major	Fixed
SV2-4	Single-step Ownership Transfer Can be Dangerous	Major	Fixed
SV2-5	Initializing `fee_to` As `ZERO_ACCOUNT` May Result In Rransferring Fees to The Zero Address	Major	Fixed
SV2-6	When Calculating Fees for Token Info Y Only, There is An Incorrect Passing of `rewards_coins`	Major	Fixed

SV2-7	Centralization Risk	Major	Acknowledged
SV2-8	Token Extraction Mismatch in Fee Distribution Logic	Major	Fixed
SV2-9	Incorrect Fee Handling in `swap_with_no_fee()`	Major	Fixed
SV2-10	Update the Reserves within the `swap()` Function	Medium	Fixed
SV2-11	Update `magnified_dividends_per_share` Values When `staked_tokens` Reaches Zero	Medium	Fixed
SV2-12	The FeeChangeEvent Structure is Not Being Utilized	Minor	Fixed
SV2-13	Redundant Operations in the Code	Minor	Fixed
SV2-14	Accessibility Contradiction in the Utilization of `swap_exact_x_to_y_direct()` Function	Minor	Fixed
SV2-15	The Necessity of Controlling Return Value Order in the `token_reserves()` Function	Minor	Fixed
SV2-16	Unused Constant	Minor	Fixed
SV2-17	Code Redundancy in The `toggle_individual_token_liquidity_fee()` Function	Minor	Fixed
SV2-18	Residual Coin Unable to be Extracted	Minor	Acknowledged
SV2-19	Redundant Pair Creation Check in `init_rewards_pool()` Function	Minor	Fixed

SV2-20	The Conventions for Using Boolean Values in Conditional Statements	Informational	Fixed
SV2-21	Function Name Typo	Informational	Fixed
SV2-22	The `toggle_individual_token_rewards_fee()` Function's Functionality is Inconsistent With Its Comment	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the BAPTSWAP Smart Contract:

Admin

- Admin can offer admin previliges through the `offer_admin_previliges()` function.
- Admin can offer treasury previliges through the `offer_treasury_previliges()` function.
- Admin can cancel previliges through the `cancel_admin_previliges()` function.
- Admin can claim previliges through the `claim_admin_previliges()` function.
- Admin can set dex liquidity fee through the `set_dex_liquidity_fee()` function.
- Admin can set dex treasury fee through the `set_dex_treasury_fee()` function.
- Admin can updates dex fee given a tier through the `update_fee_tier()` function.

Treasury

- Treasury can cancel previliges through the `cancel_treasury_previliges()` function.
- Treasury can claim previliges through the `claim_treasury_previliges()` function.

User

- Users can create a pair from 2 Coins through the `create_pair()` function.
- Users can stake tokens in pool through the `stake_tokens_in_pool()` function.
- Users can unstake tokens from pool through the `unstake_tokens_from_pool()` function.
- Users can claim rewards from pool through the `claim_rewards_from_pool()` function.
- Users can add Liquidity, create pair if it's needed through the `add_liquidity()` function.
- Users can remove liquidity through the `remove_liquidity()` function.
- Users can swap exact input amount of X to maxiumin possible amount of Y through the `swap_exact_input()` function.

- Users can swap miniumn possible amount of X to exact output amount of Y through the `swap_exact_output()` function.
- Users can swap exact input with z as intermidiate through the `swap_exact_input_with_z_as_intermidiate()` function.
- Users can swap exact input with apt as intermidiate through the `swap_exact_input_with_apt_as_intermidiate()` function.
- Users can swap exact output with z as intermidiate through the `swap_exact_output_with_z_as_intermidiate()` function.
- Users can swap exact output with apt as intermidiate through the `swap_exact_output_with_apt_as_intermidiate()` function.
- Users can register lp through the `register_lp()` function.
- Users can register token through the `register_token()` function.

Token Owner

- The owner of token can initialize individual token fees through the `initialize_fee_on_transfer()` function.
- The owner of token can set liquidity fee through the `set_liquidity_fee()` function.
- The owner of token can set reward fee through the `set_rewards_fee()` function.
- The owner of token can set team fee through the `set_team_fee()` function.
- The owner of token can add fee on transfer to a pair through the `register_fee_on_transfer_in_a_pair()` function.
- The owner of token can claim team fees in a given pair through the `claim_accumulated_team_fee()` function.
- The owner of token can toggle rewards fee through the `toggle_rewards_fee()` function.
- The owner of token can toggle all individual token fees through the `toggle_all_fees()` function.
- The owner of token can toggle liquidity fee through the `toggle_liquidity_fee()` function.
- The owner of token can toggle team fee through the `toggle_team_fee()` function.

4 Findings

ADM-1 The Distinction is Lacking When Setting Admin And `treasury_address`

Severity: Major

Status: Fixed

Code Location:

sources/admin.move#73,82

Descriptions:

Use `offer_admin_previliges()` and `offer_treasury_previliges()` to set `admin` and `treasury_address`, but lacking differentiation between role types can result in a situation where the recipient of `treasury_address` calling the function `claim_admin_previliges()` can make themselves the admin, and vice versa. This role confusion might lead to significant losses in the contract.

Suggestion:

It is recommended to make distinctions for different roles when setting recipients and perform role verification upon reception.

Resolution:

This issue has been fixed. The client has differentiated between different roles.

ADM-2 Lack of Access Control

Severity: Major

Status: Fixed

Code Location:

sources/admin.move#91-99

Descriptions:

The `cancel_admin_previliges` and `cancel_treasury_previliges` functions lack any form of access control. This implies that anyone can directly cancel any pending privileges, making it susceptible to exploitation by malicious actors and leading to potential failures in the proper execution of privilege transfers.

```
    public entry fun cancel_admin_previliges(signer_ref: &signer, id: u64) acquires Pending
{
        // destruct the pending resource
        smart_table::remove<u64, address>(&mut borrow_global_mut<Pending>
(constants::get_resource_account_address()).table, id);
    }

    public entry fun cancel_treasury_previliges(signer_ref: &signer, id: u64) acquires
Pending {
        // destruct the pending resource
        smart_table::remove<u64, address>(&mut borrow_global_mut<Pending>
(constants::get_resource_account_address()).table, id);
    }
```

Suggestion:

It is recommended to incorporate robust access control mechanisms for the `cancel_admin_previliges` and `cancel_treasury_previliges` functions.

Resolution:

This issue has been fixed. The client has already implemented additional access controls for this.

ADM-3 Permission Conflict

Severity: Major

Status: Fixed

Code Location:

sources/admin.move#73-89

Descriptions:

The presence of multiple simultaneous pending admin and treasury privileges can result in permission conflicts. For instance, if two pending admin privileges coexist, both have the ability to invoke the `claim_admin_previliges` function to acquire permissions. This scenario can lead to the loss of permissions for another admin, causing a conflict in permissions.

```
    // from the perspective of the sender
    public entry fun offer_admin_previliges(signer_ref: &signer, receiver_addr: address, id:
u64) acquires AdminInfo, Pending {
        // assert signer is the admin
        assert!(signer::address_of(signer_ref) == get_admin(), errors::not_admin());
        // assert receiver_addr is not the admin
        assert!(receiver_addr != get_admin(), errors::same_address());
        // create a new table entry
        smart_table::add<u64, address>(&mut borrow_global_mut<Pending>
(constants::get_resource_account_address()).table, id, receiver_addr)
    }

    public entry fun offer_treasury_previliges(signer_ref: &signer, receiver_addr: address,
id: u64) acquires AdminInfo, Pending {
        // assert signer is the admin
        assert!(signer::address_of(signer_ref) == get_admin(), errors::not_admin());
        // assert receiver_addr is not the admin
        assert!(receiver_addr != get_treasury_address(), errors::same_address());
        // create a new table entry
        smart_table::add<u64, address>(&mut borrow_global_mut<Pending>
(constants::get_resource_account_address()).table, id, receiver_addr)
    }
```

Suggestion:

It is recommended to restrict the simultaneous existence of multiple pending admin and treasury privileges.

Resolution:

This issue has been fixed. The client has been modified to have only one admin and treasury privilege simultaneously.

ADM-4 Sensitive Operation Lacks Event

Severity: Minor

Status: Fixed

Code Location:

sources/admin.move#73-180

Descriptions:

In the contract, some sensitive operations lack event listeners, making it difficult for external tracking of changes in related data within the contract.

The functions affected by this issue include `offer_admin_previliges()`, `cancel_admin_previliges()`, `claim_admin_previliges()`, and `set_dex_liquidity_fee()`,among others.

Suggestion:

It is recommended to add events similar to other operations to facilitate monitoring changes within the contract.

Resolution:

This issue has been acknowledged. The client has added events for critical operations.

ADM-5 The Comments Do Not Match the Actual Functionality

Severity: Informational

Status: Fixed

Code Location:

sources/admin.move#203-214

Descriptions:

The actual functionality here indicates that the `receiver_addr` cannot be the `treasury_address`. The comment is incorrect.

```
 public entry fun offer_treasury_previliges(signer_ref: &signer, receiver_addr: address)
acquires AdminInfo, Pending {
        // assert no request is pending
        assert!
(smart_table::length(&borrow_global_mut<Pending<Treasury>>
(constants::get_resource_account_address()).table) == 0,
errors::pending_request());
        // assert signer is the admin
        assert!(signer::address_of(signer_ref) == get_admin(),
errors::not_admin());
        // assert receiver_addr is not the admin
        assert!(receiver_addr != get_treasury_address(),
errors::same_address());
        // create a new table entry
        smart_table::add<Treasury, address>(&mut
borrow_global_mut<Pending<Treasury>>
(constants::get_resource_account_address()).table, Treasury {},
receiver_addr);
        // emit event
        emit_ownership_transfer_request_event(receiver_addr);
    }
```

Suggestion:

It is recommended to update the comments to accurately reflect their corresponding functionalities.

Resolution:

This issue has been fixed. The client has made modifications to the comments.

ADM-6 The Comment in the `claim_treasury_previliges()` Function is Incorrect

Severity: Informational

Status: Fixed

Code Location:

sources/admin.move#116

Descriptions:

In the function `admin.claim_treasury_previliges()`, there is a comment indicating "update admin info" which is incorrect. It should be corrected to "update treasury info"

```
    public entry fun claim_treasury_previliges(signer_ref: &signer, id: u64) acquires AdminInfo, Pending {
        // assert id exists and the signer is the receiver
        assert!(smart_table::contains<u64, address>(&borrow_global_mut<Pending>(constants::get_resource_account_address()).table, id), 1);
        assert!(signer::address_of(signer_ref) == *smart_table::borrow(&borrow_global_mut<Pending>(constants::get_resource_account_address()).table, id), 1);
        // update admin info
        set_treasury_address(*smart_table::borrow(&borrow_global_mut<Pending>(constants::get_resource_account_address()).table, id));
        // remove the entry
        smart_table::remove<u64, address>(&mut borrow_global_mut<Pending>(constants::get_resource_account_address()).table, id);
    }
```

Suggestion:

It is recommended to modify the corresponding comments accordingly.

Resolution:

This issue has been fixed. The client has updated the remarks accordingly.

FOT-1 Functions with Similar Functionality

Severity: Minor

Status: Fixed

Code Location:

sources/fee_on_transfer.move#217,235

Descriptions:

Within `fee_on_transfer.move`, the functions `get_info()` and `get_fee_on_transfer_info()` serve the same purpose. The only difference lies in their visibility. `get_fee_on_transfer_info()` can entirely replace `get_info()`. Redundant code may lead to increased gas consumption and impact code readability.

Suggestion:

It is recommended to delete the function `get_info()`.

Resolution:

This issue has been fixed. The client deleted the function `get_fee_on_transfer_info()`.

RV2-1 Nonexistent Token Pair

Severity: Major

Status: Fixed

Code Location:

sources/router_v2.move#227,229,244,246;

sources/swap_v2.move#1583,1603

Descriptions:

In the function `swap_v2::swap_exact_fee_to_apt()`, it attempts to retrieve information about `<TokenPairMetadata<X, APT>>`. However, under normal circumstances, such information doesn't exist unless created using the `create_pair()` function. Doing so would entail creating pairs for all tokens with `APT`, which clearly doesn't align with logic. `<TokenPairReserve<X, APT>>` faces a similar issue.

Suggestion:

It is recommended to use token X as the fee and transfer it to the `fee_to` address if the token pair does not exist.

Resolution:

This issue has been fixed. The client has deleted the code related to `APT`.

RV2-2 The Admin is Unable to Update the Liquidity Fee and Treasury Fee

Severity: Major

Status: Fixed

Code Location:

sources/router_v2.move#1261,1277

Descriptions:

The `swap_v2.set_dex_liquidity_fee()` function is marked as public(friend), indicating that it is accessible to modules declared as "friends" of the current module.

```
public(friend) fun set_dex_liquidity_fee(
        sender: &signer,
        new_fee: u128
    ) acquires SwapInfo {
        let swap_info = borrow_global_mut<SwapInfo>(RESOURCE_ACCOUNT);
        // assert sender is admin
        assert!(signer::address_of(sender) == swap_info.admin, ERROR_NOT_ADMIN);
        // assert new fee is not equal to the existing fee
        assert!(new_fee != swap_info.liquidity_fee_modifier, 1);
        // assert the newer total fee is less than the threshold
        assert!(does_not_exceed_dex_fee_threshold(new_fee +
swap_info.treasury_fee_modifier) == true, 1);
        // update the fee
        swap_info.liquidity_fee_modifier = new_fee;
    }
```

However, in the protocol, only `baptswap_v2::router_v2` is declared as a friend.

```
    use bapt_framework::deployer;

    friend baptswap_v2::router_v2;
```

The issue arises because the `router_v2` contract does not invoke the `set_dex_liquidity_fee()` method, preventing the protocol from updating the liquidity fee. The function `ser_dex_treasury_fee()` , `set_individual_token_team_fee()` and `set_individual_token_liquidity_fee()` also face a similar issue.

Suggestion:

It is recommended to remove the friend modifier and change the function to entry. This is because the function already includes a check for whether the sender is an admin or token owner internally.

Resolution:

This issue has been fixed. The client has updated the implementation logic.

RV2-3 Unused Private Function

Severity: Minor

Status: Fixed

Code Location:

sources/router_v2.move#118

Descriptions:

The function `assert_pair_is_not_created()` defined in module `router_v2` is not used, which leads to increased gas consumption and reduces the readability and understandability of the code.

Suggestion:

It is recommended to delete this function.

Resolution:

This issue has been fixed. The client has deleted this function.

RV2-4 Code Refactoring Suggestions in `router_v2` Module

Severity: Minor

Status: Fixed

Code Location:

sources/router_v2.move#40,54,69,82

Descriptions:

In the `router_v2` module, lines 40, 54, 69, and 82 can be replaced with a function named `assert_pair_is_created()`, as they serve the same purpose. This change would enhance readability and understanding while reducing code duplication. Additionally, the code on line 55 is repeated across multiple functions and could be encapsulated into its function for reusability.

Suggestion:

It is recommended to replace Repetitive Code with Function Calls

Resolution:

This issue has been fixed. The client has used an alternative function.

RV2-5 The Specification for Assert Statements

Severity: Minor

Status: Fixed

Code Location:

sources/router_v2.move#226,243;

sources/swap_v2.move#263,264,690,760,792,824,1269,1271,1285,1287,1304,1306,1321,1323,1338,134

Descriptions:

The error codes in assert statements show a number of '1's. Best practice suggests using constants, ensuring different error code constants have distinct values.

Suggestion:

It is recommended to modify the error codes in assert statements to align with best practices.

Resolution:

This issue has been fixed. The client has modified the error code of the assert.

STA-1 Updating Magnified Dividends Per Share during Unstaking is Incorrect

Severity: Major

Status: Fixed

Code Location:

sources/stake.move#220-221

Descriptions:

When users generate fees during transactions in the `swap_v2` contract, the protocol calls `stake.distribute_rewards()` to distribute these fees to the stakers and update the magnitude values.

```
        if (metadata.rewards_fee > 0) {
                let rewards_coins = coin::extract<X>(&mut metadata.balance_x,
(amount_to_rewards as u64));
                stake::distribute_rewards<X, Y>(rewards_coins, coin::zero<Y>());
            };
    public(friend) fun distribute_rewards<X, Y>(
        rewards_x: coin::Coin<X>,
        rewards_y: coin::Coin<Y>
    ) acquires TokenPairRewardsPool {
        // Update pool
        update_pool<X, Y>(coin::value<X>(&rewards_x), coin::value<Y>(&rewards_y));

        let rewards_pool = borrow_global_mut<TokenPairRewardsPool<X, Y>>
(constants::get_resource_account_address());
        coin::merge(&mut rewards_pool.balance_x, rewards_x);
        coin::merge(&mut rewards_pool.balance_y, rewards_y);
    }
```

However, when users unstake, the protocol also updates the magnified dividends per share. The calculation involves adding `((amount as u128) * pool_info.precision_factor / (pool_info.staked_tokens as u128))` to the original per share x or per share y values.

```
    if (amount > 0) {
            utils::transfer_out<X>(&mut user_info.staked_tokens, sender, amount);
```

```
            pool_info.staked_tokens = pool_info.staked_tokens - amount;
            // update magnified dividends per share
            pool_info.magnified_dividends_per_share_x =
pool_info.magnified_dividends_per_share_x + ((amount as u128) *
pool_info.precision_factor / (pool_info.staked_tokens as u128));
            pool_info.magnified_dividends_per_share_y =
pool_info.magnified_dividends_per_share_y + ((amount as u128) *
pool_info.precision_factor / (pool_info.staked_tokens as u128));
        };
```

This is incorrect, as it causes the `pool_info.magnified_dividends_per_share_x` or `pool_info.magnified_dividends_per_share_y` values to increase without actual rewards being distributed to the rewards pool.

Subsequent users who attempt to claim rewards or unstake may receive more rewards, as the values of `pool_info.magnified_dividends_per_share_x` or `pool_info.magnified_dividends_per_share_y` increase. This could potentially lead to users being unable to withdraw rewards or funds in future unstaking or reward claiming transactions, resulting in their funds being locked in the contract.

Suggestion:

It is recommended to refrain from updating magnified dividends per share during unstaking.

Resolution:

This issue has been fixed. The client removed the update code during unstaking.

STA-2 Direct Invocation Risk in `unstake_tokens()` and `claim_rewards()` Functions in `stake` Module

Severity: Medium

Status: Fixed

Code Location:

sources/stake.move#183,265

Descriptions:

The function `unstake_tokens()` in the `stake` module can be directly called. According to the contract code, only one of `TokenPairRewardsPool<X, Y>` and `TokenPairRewardsPool<Y, X>` exists based on the sorting of tokens. However, this function doesn't verify the existence of `TokenPairRewardsPool<Y, X>`. If a user incorrectly inputs the token order, the function won't execute successfully. Unlike other similar functions that are friend functions accessible only through the `router_v2` module, it's advisable for this function to also use a friend function to control its invocation. Additionally, the function `claim_rewards()` suffers from the same issue.

Suggestion:

It is recommended to modify functions to friend functions

Resolution:

This issue has been fixed. The client has changed to a friend function.

STA-3 Optimization through Consolidating `claim_rewards()` and `unstake_tokens()` Functions

Severity: Minor

Status: Fixed

Code Location:

sources/stake.move#265

Descriptions:

The functions `claim_rewards()` and `unstake_tokens()` within the `stake` module have almost identical code. Invoking `unstake_tokens(sender, 0)` within the `claim_rewards()` function achieves the same effect. Encapsulating the code within the `unstake_tokens()` function into a common function for caller use reduces redundant code and enhances readability and understanding.

Suggestion:

It is recommended to encapsulate identical code into functions to reduce redundancy.

Resolution:

This issue has been fixed. The client has optimized the repetitive code.

SV2-1 The Constant Product Rule is Compromised, Enabling Pool Draining

Severity: Critical

Status: Fixed

Code Location:

sources/swap_v2.move#886-939

Descriptions:

The function `swap_exact_x_to_y_direct()` is used to swap token X for token Y directly. In this function, the protocol swaps the user-provided amount, `amount_in`, for token Y (`coins_y_out`) in the swap function.

```
// Get amount after deducting fees and swap it to y
let amount_out = swap_utils::get_amount_out(amount_in, rin, rout, total_fees);
let (coins_x_out, coins_y_out) = swap<X, Y>(0, amount_out);
```

Within the swap function, the `update()` function is called, which is responsible for maintaining the constant product rule x * y = k

```
    // No need to use u256 when balance_x_adjusted * balance_y_adjusted and
reserve_x_adjusted * reserve_y_adjusted are less than constants::get_max_u128().
        let compare_result = if(
            balance_x_adjusted > 0
            && reserve_x_adjusted > 0
            && constants::get_max_u128() / balance_x_adjusted > balance_y_adjusted
            && constants::get_max_u128() / reserve_x_adjusted > reserve_y_adjusted
        ) { balance_x_adjusted * balance_y_adjusted >= reserve_x_adjusted *
reserve_y_adjusted } else {
            let p = u256::mul_u128(balance_x_adjusted, balance_y_adjusted);
            let k = u256::mul_u128(reserve_x_adjusted, reserve_y_adjusted);
            u256::ge(&p, &k)
        };
        assert!(compare_result, errors::k());

        update(balance_x, balance_y, reserves);

        (coins_x_out, coins_y_out)
```

Subsequently, the protocol execute `distribute_dex_fees()` and `distribute_fee_on_transfer_fees()` to distribute fees. Treasury Fee Distribution(distribute_dex_fees()):

- The protocol extracts the treasury fee from `metadata.balance_y`.
- Updates the constant product rule: k = x * (y – treasury_fee).

```
// treasury
        let treasury_fee_coins = coin::extract<Y>(&mut metadata.balance_y,
(amount_to_treasury as u64));
        coin::deposit<Y>(admin::get_treasury_address(), treasury_fee_coins);
        // update reserves
        update_reserves<X, Y>();
```

Transfer Fee Distribution (`distribute_fee_on_transfer_fees()`): If token Y is registered as a fee-charging token, the protocol: * Extracts `amount_to_rewards` from `metadata.balance_y` and allocates it to the rewards pool. * Extracts `amount_to_team` from `metadata.balance_y` and assigns it to `metadata.team_balance_y`.

- Calls `update_reserves()` to update the constant product rule: k = x * (y – treasury_fee – amount_to_rewards – amount_to_team).

```
let (amount_to_liquidity, amount_to_rewards, amount_to_team) =
calculate_fee_on_transfer_amounts<Y>(amount_in);

            // extract fees
            let liquidity_coins = coin::extract<Y>(&mut metadata.balance_y,
(amount_to_liquidity as u64));
            // let rewards_coins = coin::extract<Y>(&mut metadata.balance_y,
(amount_to_rewards as u64));
            let team_coins = coin::extract<Y>(&mut metadata.balance_y, (amount_to_team
as u64));

            // distribute fees
            coin::merge(&mut metadata.balance_y, liquidity_coins);
            // rewards fees must go to rewards pool
            if (metadata.rewards_fee > 0) {
                let rewards_coins = coin::extract<Y>(&mut metadata.balance_y,
(amount_to_rewards as u64));
                stake::distribute_rewards<X, Y>(coin::zero<X>(), rewards_coins);
            };
            coin::merge(&mut metadata.team_balance_y, team_coins);
```

```
        // update reserves
        update_reserves<X, Y>();
```

In this scenario, each exchange from X to Y results in a reduction of the constant product k, as various fees are subtracted from the reserve of token Y. If a hacker exploits this mechanism using flash loans to repeatedly swap X for Y, the continuous reduction in k breaks the x * y = k formula. When the quantity of token Y in the pool becomes extremely low, indicating a significantly high value for token Y, a hacker can exploit this situation. With a minimal amount of token Y, the hacker can efficiently exchange for a substantial portion of token X from the pool.

Suggestion:

It is recommended to deduct fees directly from `amount_in` before initiating the token exchange and fee distribution processes. Furthermore, not update the constant product formula during fee distribution.

Resolution:

This issue has been fixed. The client followed our advice.

SV2-2 There is No Slippage Protection During The Distribution of DEX Fees

Severity: Major

Status: Fixed

Code Location:

sources/swap_v2.move#1565-1590

Descriptions:

The `swap_v2.distribute_dex_fees()` function is used in swap functions to distribute DEX fees and update reserves correspondingly.Within the function, when `type_info::type_of<X>() != type_info::type_of<APT>()`, the protocol invokes the `swap_exact_fee_to_apt()` function to exchange token X for APT, and subsequently transfers the acquired APT to the treasury. However, during this exchange process, there is an absence of slippage protection.That will cause a loss of funds because of sandwich attacks.

```
        if (type_info::type_of<X>() != type_info::type_of<APT>()) {
            let metadata = borrow_global_mut<TokenPairMetadata<X, Y>>
(RESOURCE_ACCOUNT);
            // extract it from balance x from the metadata
            let coin_x_out = coin::extract<X>(&mut metadata.balance_x, amount_in);
            // swap it to APT
            let coin_y_out = swap_exact_fee_to_apt<X>(coin_x_out);
            // deposit APT to treasury
            // assert!(borrow_global<SwapInfo>(RESOURCE_ACCOUNT).fee_to ==
RESOURCE_ACCOUNT, 1);
            coin::deposit<APT>(fee_to(), coin_y_out);
            // update reserves
            update_reserves<X, Y>();
        }
```

Suggestion:

It is recommended to incorporate a certain level of slippage protection during the exchange of X for APT.

Resolution:

This issue has been fixed. The client has updated the implementation logic.

SV2-3 Infinite Recursion in `distribute_dex_fees()` Leading to Transaction Failure

Severity: Major

Status: Fixed

Code Location:

sources/swap_v2.move#1576

Descriptions:

The function `swap_v2.distribute_dex_fees()` aims to calculate and distribute DEX fees based on the type of input X. In this function, the protocol calls `swap_exact_x_to_y_direct()` to exchange X for APT and then transfers the obtained APT to the treasury. However, within the `swap_exact_x_to_y_direct()` function, the protocol again invokes `distribute_dex_fees()`. This recursive calling pattern leads to an infinite loop, resulting in an out-of-gas situation and a failed transaction.

```
  if (type_info::type_of<X>() != type_info::type_of<APT>()) {
            let metadata = borrow_global_mut<TokenPairMetadata<X, Y>>
(RESOURCE_ACCOUNT);
            /// extract it from balance x from the metadata
            let coin_x_out = coin::extract<X>(&mut metadata.balance_x, amount_in);
            /// update reserves
            update_reserves<X, Y>();
            /// swap it to APT
            let (coins_x_out, coin_y_out) = swap_exact_x_to_y_direct<X, APT>(coin_x_out);
            coin::destroy_zero(coins_x_out); /// or others ways to drop `coins_x_out`
            /// deposit APT to treasury
            let swap_info = borrow_global<SwapInfo>(RESOURCE_ACCOUNT);
            coin::deposit<APT>(swap_info.fee_to, coin_y_out);
        }
```

Suggestion:

It is recommended to implement a new function to handle the exchange operation, breaking the cycle and preventing infinite recursion.

SV2-4 Single-step Ownership Transfer Can be Dangerous

Severity: Major

Status: Fixed

Code Location:

sources/swap_v2.move#662-667

Descriptions:

Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. If the admin permissions are given to the wrong address within this function, it will cause irreparable damage to the contract.

```
public entry fun set_admin(sender: &signer, new_admin: address) acquires SwapInfo {
    let sender_addr = signer::address_of(sender);
    let swap_info = borrow_global_mut<SwapInfo>(RESOURCE_ACCOUNT);
    assert!(sender_addr == swap_info.admin, ERROR_NOT_ADMIN);
    swap_info.admin = new_admin;
}
```

Suggestion:

It is recommended to use a two-step ownership transfer pattern, meaning ownership transfer gets to a "pending" state and the new owner should claim his new rights, otherwise the old owner still has control of the contract.

Resolution:

This issue has been fixed. The client followed our advice.

SV2-5 Initializing `fee_to` As `ZERO_ACCOUNT` May Result In Rransferring Fees to The Zero Address

Severity: Major

Status: Fixed

Code Location:

sources/swap_v2.move#235-246

Descriptions:

In the `init_module` function, initializing `fee_to` as `ZERO_ACCOUNT` means that if the `set_fee_to` function is called to set a new address for fee reception, swap fees will be transferred to the zero address.

```
fun init_module(sender: &signer) {
    let signer_cap = resource_account::retrieve_resource_account_cap(sender, DEV);
    let resource_signer = account::create_signer_with_capability(&signer_cap);
    move_to(&resource_signer, SwapInfo {
        signer_cap,
        fee_to: ZERO_ACCOUNT,
        admin: DEFAULT_ADMIN,
        liquidity_fee_modifier: 30,  /// 0.3%
        treasury_fee_modifier: 60,   /// 0.6%
        pair_created: account::new_event_handle<PairCreatedEvent>(&resource_signer),
    });
}
```

Suggestion:

It is recommended to set `fee_to` to a valid address during initialization or call `set_fee_to` before performing any swaps.

Resolution:

This issue has been fixed. The client initialized using the treasury address.

SV2-6 When Calculating Fees for Token Info Y Only, There is An Incorrect Passing of `rewards_coins`

Severity: Major

Status: Fixed

Code Location:

sources/swap_v2.move#1705

Descriptions:

The `swap_v2.distribute_fee_on_transfer()` function is designed to distribute fees during a transaction. As shown in the following code, if token info y is registered & token info x not, it calculates only token info y fees.

```
else if (option::is_none<TokenInfo<X>>(&token_info_x) &&
!option::is_none<TokenInfo<Y>>(&token_info_y)) {
            let extracted_token_info_y = option::extract(&mut token_info_y);
            // calculate the fees
            let (amount_to_liquidity, amount_to_rewards, amount_to_team) =
calculate_fee_on_transfer_amounts<Y>(extracted_token_info_y, amount_in);

            // extract fees
            let liquidity_coins = coin::extract<Y>(&mut metadata.balance_y,
(amount_to_liquidity as u64));
            // let rewards_coins = coin::extract<Y>(&mut metadata.balance_y,
(amount_to_rewards as u64));
            let team_coins = coin::extract<Y>(&mut metadata.balance_y, (amount_to_team
as u64));

            // distribute fees
            coin::merge(&mut metadata.balance_y, liquidity_coins);
            // rewards fees must go to rewards pool
            if (metadata.rewards_fee > 0) {
                let rewards_pool = borrow_global_mut<TokenPairRewardsPool<X, Y>>
(RESOURCE_ACCOUNT);
                let rewards_coins = coin::extract(&mut metadata.balance_y,
(amount_to_rewards as u64));

                update_pool<X,Y>(rewards_pool, coin::value(&rewards_coins), 0);
                coin::merge(&mut rewards_pool.balance_y, rewards_coins);
```

```
        };
        coin::merge(&mut metadata.team_balance_y, team_coins);
        // update reserves
        update_reserves<X, Y>();
    }
```

However, it extracts `rewards_coins` from `metadata.balance_y`, but when calling the `update_pool()` function, it passes these rewards coins to `reward_x`,causing confusion in calculation logic.

Suggestion:

It is recommended to pass `rewards_coins` to `reward_y` when calling the `update_pool()` function.

```
update_pool<X,Y>(rewards_pool,0 , coin::value(&rewards_coins));
```

Resolution:

This issue has been fixed. The client has changed the implementation logic.

SV2-7 Centralization Risk

Severity: Major

Status: Acknowledged

Code Location:

sources/swap_v2.move

Descriptions:

Admin

- Admin can offer admin previliges through the `offer_admin_previliges()` function.
- Admin can offer treasury previliges through the `offer_treasury_previliges()` function.
- Admin can cancel previliges through the `cancel_admin_previliges()` function.
- Admin can claim previliges through the `claim_admin_previliges()` function.
- Admin can set dex liquidity fee through the `set_dex_liquidity_fee()` function.
- Admin can set dex treasury fee through the `set_dex_treasury_fee()` function.
- Admin can updates dex fee given a tier through the `update_fee_tier()` function.

Token Owner

- The owner of token can initialize individual token fees through the `initialize_fee_on_transfer()` function.
- The owner of token can set liquidity fee through the `set_liquidity_fee()` function.
- The owner of token can set reward fee through the `set_rewards_fee()` function.
- The owner of token can set team fee through the `set_team_fee()` function.
- The owner of token can add fee on transfer to a pair through the `register_fee_on_transfer_in_a_pair()` function.
- The owner of token can claim team fees in a given pair through the `claim_accumulated_team_fee()` function.
- The owner of token can toggle rewards fee through the `toggle_rewards_fee()` function.

- The owner of token can toggle all individual token fees through the `toggle_all_fees()` function.
- The owner of token can toggle liquidity fee through the `toggle_liquidity_fee()` function.
- The owner of token can toggle team fee through the `toggle_team_fee()` function.

Suggestion:

It is recommended to take some measures to mitigate centralization risk.

Resolution:

The client used multisig to mitigate this issue.

SV2-8 Token Extraction Mismatch in Fee Distribution Logic

Severity: Major

Status: Fixed

Code Location:

sources/swap_v2.move#1557-1570

Descriptions:

The function `swap_v2.distribute_dex_fees()` is used to ensure proper distribution of DEX fees, regardless of the input token. In the case where `type_info::type_of<X>() != type_info::type_of<APT>()`, the line `coin_x_out = coin::extract<X>(&mut metadata.balance_x, amount_in)` extracts the token amount from `metadata.balance_x` using the user-input `amount_in`. However, it seems that the intended behavior might be to use `amount_to_liquidity + amount_to_treasury` instead of `amount_in`.

```
 let (amount_to_liquidity, amount_to_treasury) = calculate_dex_fees_amounts<X>
(amount_in);
        // if X is not APT, swap the amounts into APT
        if (type_info::type_of<X>() != type_info::type_of<APT>()) {
            let metadata = borrow_global_mut<TokenPairMetadata<X, Y>>
(RESOURCE_ACCOUNT);
            // extract it from balance x from the metadata
            let coin_x_out = coin::extract<X>(&mut metadata.balance_x, amount_in);
            // swap it to APT
            let coin_y_out = swap_exact_fee_to_apt<X>(coin_x_out);
            // deposit APT to treasury
            // assert!(borrow_global<SwapInfo>(RESOURCE_ACCOUNT).fee_to ==
RESOURCE_ACCOUNT, 1);
            coin::deposit<APT>(fee_to(), coin_y_out);
            // update reserves
            update_reserves<X, Y>();
        }
```

If `amount_in` is used in this context, it means the function might be extracting a token amount for swapping that doesn't match the calculated fees (`amount_to_liquidity + amount_to_treasury`), potentially resulting in an incorrect fee distribution. This could lead

to `fee_to()` receiving more fees than expected, as it would be based on the user-provided amount rather than the calculated fees.

Suggestion:

It is recommended to use the total calculated fees (`amount_to_liquidity + amount_to_treasury`) when extracting tokens for swapping.

Resolution:

This issue has been fixed. The client has updated the implementation logic.

SV2-9 Incorrect Fee Handling in `swap_with_no_fee()`

Severity: Major

Status: Fixed

Code Location:

sources/swap_v2.move#1597-1645

Descriptions:

In the `swap_with_no_fee()` function, the protocol extracts amounts from `TokenPairMetadata<X, APT>` based on the values of the parameters `amount_x_out` and `amount_y_out`.

```
 fun swap_with_no_fee<X, APT>(
        amount_x_out: u64,
        amount_y_out: u64
    ): (Coin<X>, Coin<APT>) acquires TokenPairReserve, TokenPairMetadata {
        assert!(amount_x_out > 0 || amount_y_out > 0,
ERROR_INSUFFICIENT_OUTPUT_AMOUNT);

        let reserves = borrow_global_mut<TokenPairReserve<X, APT>>
(RESOURCE_ACCOUNT);
        assert!(amount_x_out < reserves.reserve_x && amount_y_out < reserves.reserve_y,
ERROR_INSUFFICIENT_LIQUIDITY);

        let metadata = borrow_global_mut<TokenPairMetadata<X, APT>>
(RESOURCE_ACCOUNT);

        let coins_x_out = coin::zero<X>();
        let coins_y_out = coin::zero<APT>();
        if (amount_x_out > 0) coin::merge(&mut coins_x_out, extract_x(amount_x_out,
metadata));
        if (amount_y_out > 0) coin::merge(&mut coins_y_out, extract_y(amount_y_out,
metadata));
        let (balance_x, balance_y) = token_balances<X, APT>();
```

However, when this function is called from `swap_exact_fee_to_apt()`, it is passed the arguments 0 and `amount_in` x.

```
 fun swap_exact_fee_to_apt<X>(coins_in: Coin<X>): Coin<APT> acquires
TokenPairReserve, TokenPairMetadata {
```

```
        // Grab token pair metadata
        let metadata = borrow_global_mut<TokenPairMetadata<X, APT>>
(RESOURCE_ACCOUNT);
        // get the value of coins in u64
        let amount_in = coin::value<X>(&coins_in);

        // deposit amount_in x into balance x
        coin::merge(&mut metadata.balance_x, coins_in);

        // Get amount y
        let (coins_x_out, coins_y_out) = swap_with_no_fee<X, APT>(0, amount_in);
        coin::destroy_zero(coins_x_out); // or others ways to drop `coins_x_out`

            coins_y_out
    }
```

This implies that the protocol attempts to extract an amount of APT tokens corresponding to the fee in token X, rather than converting token X to APT. This results in a loss of funds for the protocol.

Suggestion:

It is recommended to convert token X into APT and then transfer it to the `fee_to` address.

Resolution:

This issue has been fixed. The client has deleted this function.

SV2-10 Update the Reserves within the `swap()` Function

Severity: Medium

Status: Fixed

Code Location:

sources/swap_v2.move#498

Descriptions:

In the `swap_exact_x_to_y_direct()` function, the protocol swaps token X to token Y and subsequently calls `update_reserves()` to update the constant product.

```
public(friend) fun swap_exact_x_to_y_direct<X, Y>(
    coins_in: coin::Coin<X>
): (coin::Coin<X>, coin::Coin<Y>) acquires TokenPairReserve, TokenPairMetadata {
    let amount_in = coin::value<X>(&coins_in);
    deposit_x<X, Y>(coins_in);
    let (rin, rout, _) = token_reserves<X, Y>();
    let amount_out = swap_utils_v2::get_amount_out(amount_in, rin, rout);
    let (coins_x_out, coins_y_out) = swap<X, Y>(0, amount_out);
    // update reserves
    update_reserves<X, Y>();
    assert!(coin::value<X>(&coins_x_out) == 0, errors::insufficient_output_amount());
    (coins_x_out, coins_y_out)
}
```

However, a best practice, as exemplified in the PancakeSwap code, is to call the `update()` function within the `swap()` function to handle the updates. This ensures that the reserves are consistently and efficiently updated during the swapping process.

Suggestion:

It is recommended to call the `update()` function within the `swap()` function to update the constant product.

Resolution:

This issue has been fixed. The client followed our advice

SV2-11 Update `magnified_dividends_per_share` Values When `staked_tokens` Reaches Zero

Severity: Medium

Status: Fixed

Code Location:

sources/swap_v2.move#552-555

Descriptions:

In the `swap_v2.unstake_tokens()` function, the protocol transfers staked tokens to the user and subsequently deducts the corresponding amount from `pool_info.staked_tokens`. However, a issue arises when `pool_info.staked_tokens` reaches zero,the protocol fails to update `pool_info.magnified_dividends_per_share_x` and `pool_info.magnified_dividends_per_share_y`. This inconsistency results in a mismatch between the current state of `pool_info` and its initialized state.

```
// Tranfer staked tokens out
        if (amount > 0) {
            transfer_out<X>(&mut user_info.staked_tokens, sender, amount);
            pool_info.staked_tokens = pool_info.staked_tokens - amount;
        };
```

Suggestion:

It is recommend to update `pool_info.magnified_dividends_per_share_x` and `pool_info.magnified_dividends_per_share_y` to their initial values when `pool_info.staked_tokens` becomes zero.

Resolution:

This issue has been fixed. The protocol now initializes the corresponding value when staked tokens are 0.

SV2-12 The FeeChangeEvent Structure is Not Being Utilized

Severity: Minor

Status: Fixed

Code Location:

sources/swap_v2.move#222

Descriptions:

The `FeeChangeEvent` structure is intended to monitor changes in various fees, but it's not being utilized within the contract. As a result, there's an inability to promptly track changes in fees.

Suggestion:

It is recommended that when updating fees, utilize `FeeChangeEvent` to trigger the event.

Resolution:

This issue has been fixed. The client has been removed from the code; fee change is now being tracked in `admin.move` and `fee_on_transfer.move`.

SV2-13 Redundant Operations in the Code

Severity: Minor

Status: Fixed

Code Location:

sources/swap_v2.move#821-826;

sources/swap_v2.move#792-793

Descriptions:

In the function `swap_v2.distribute_dex_fees()`, it is unnecessary for the protocol to extract `liquidity_fee_coins` from `metadata.balance_y` and then immediately merge it back into `metadata.balance_y`.

```
        // liquidity
        let liquidity_fee_coins = coin::extract<Y>(&mut metadata.balance_y,
(amount_to_liquidity as u64));
        coin::merge(&mut metadata.balance_y, liquidity_fee_coins);
```

In the `distribute_fee_on_transfer_fees()` function, there is a similar issue with the handling of `liquidity_coins`.

Suggestion:

It is recommended to remove the unnecessary extraction and merging of `liquidity_fee_coins` in the `distribute_dex_fees()` function.

Resolution:

This issue has been fixed. The client removed the redundant code operations.

SV2-14 Accessibility Contradiction in the Utilization of `swap_exact_x_to_y_direct()` Function

Severity: Minor

Status: Fixed

Code Location:

sources/swap_v2.move#516,602

Descriptions:

The function `swap_exact_x_to_y_direct()` is a friend function, yet it's only called within the current module and not in any other modules. Therefore, it behaves as a private function, which contradicts the intended access permissions for this function.The function `swap_exact_y_to_x_direct()` and `update_pool()` also suffers from the same issue.

Suggestion:

It is recommended to modify this function to be a private function.

Resolution:

This issue has been fixed. The client has been modified to a private function.

SV2-15 The Necessity of Controlling Return Value Order in the `token_reserves()` Function

Severity: Minor

Status: Fixed

Code Location:

sources/swap_v2.move#653

Descriptions:

The function `token_reserves()` adjusts the order of returned values by sorting currencies, which might not be necessary. As the order has already been adjusted before calling this function within the current contract, the if statement is executed every time. To prevent confusion, we believe that the control over the sequence should occur when receiving the return values of this function, rather than within the current function. We also compared this to PancakeSwap's code, which similarly does not control the sequence within the current function.

Suggestion:

It is recommended to consistency with PancakeSwap, Unless Specifically Designed for Other Functionalities.

Resolution:

This issue has been fixed. The client followed the suggestion and canceled the reordering of the return values.

SV2-16 Unused Constant

Severity: Minor

Status: Fixed

Code Location:

sources/swap_v2.move#49

Descriptions:

The main consequence of the Unused Constants defect is the increase in gas costs during module deployment, leading to gas wastage.

```
const ERROR_ONLY_ADMIN: u64 = 0;
const ERROR_NOT_CREATOR: u64 = 2;
const ERROR_TOKENS_NOT_SORTED: u64 = 9;
const ERROR_X_NOT_REGISTERED: u64 = 16;
const ERROR_Y_NOT_REGISTERED: u64 = 16;
const ERROR_NOT_FEE_TO: u64 = 18;
const ERROR_NOT_EQUAL_EXACT_AMOUNT: u64 = 19;
const ERROR_NOT_RESOURCE_ACCOUNT: u64 = 20;
const ERROR_EXCESSIVE_FEE: u64 = 22;
const ERROR_MUST_BE_INFERIOR_TO_TWENTY: u64 = 24;
const ERROR_NO_REWARDS: u64 = 28;
```

Suggestion:

It is recommended to remove unused constants or utilize them in the code.

Resolution:

This issue has been fixed. The client has already used these error constants.

SV2-17 Code Redundancy in The `toggle_individual_token_liquidity_fee()` Function

Severity: Minor

Status: Fixed

Code Location:

sources/swap_v2.move#743-759

Descriptions:

The code below has redundant blocks of code for both branches where `type_info::type_of<CoinType>() == type_info::type_of<X>()` and `type_info::type_of<CoinType>() == type_info::type_of<Y>()`. Regardless of which branch is taken, the same logic is executed. This redundancy could be streamlined to improve code readability and maintainability.

```
let metadata = borrow_global_mut<TokenPairMetadata<X, Y>>(RESOURCE_ACCOUNT);
        let token_info = borrow_global<TokenInfo<CoinType>>(signer::address_of(sender));
        // if cointype = x
        if (type_info::type_of<CoinType>() == type_info::type_of<X>()) {
            // if activate = true
            if (activate == true) {
                metadata.liquidity_fee = metadata.liquidity_fee +
token_info.liquidity_fee_modifier;
            // if activate = false
            } else {
                metadata.liquidity_fee = metadata.liquidity_fee -
token_info.liquidity_fee_modifier;
            }
        // if cointype = y
        } else if (type_info::type_of<CoinType>() == type_info::type_of<Y>()) {
            // if activate = true
            if (activate == true) {
                metadata.liquidity_fee = metadata.liquidity_fee +
token_info.liquidity_fee_modifier;
            // if activate = false
            } else {
                metadata.liquidity_fee = metadata.liquidity_fee -
token_info.liquidity_fee_modifier;
```

```
        }
    } else { assert!(false, 1); }
```

Suggestion:

It is recommend to consolidate the common logic for both branches and eliminate the need for duplicate code.

Resolution:

This issue has been fixed. The client has removed this function.

SV2-18 Residual Coin Unable to be Extracted

Severity: Minor

Status: Acknowledged

Code Location:

sources/swap_v2.move#518-559

Descriptions:

Due to precision loss, there is a persistent issue of residual coins in the `RewardsPoolUserInfo` that cannot be extracted. Every time the `update_pool` function is called during a transaction to distribute rewards from the `rewards_pool` to users staking and to update `acc_token_per_share`, the value of `acc_token_per_share` is rounded down due to precision loss. As a result, users are consistently unable to claim the full `rewards_pool`. However, the `rewards_pool` continues to accumulate in the `RewardsPoolUserInfo`. This means that with each transaction, if there is precision loss, the unrecoverable portion of the `rewards_pool accumulates` in the `RewardsPoolUserInfo`, making it unclaimable.

```
if (reward_x > 0) {
            // acc_token_per_share = acc_token_per_share + (reward * precision_factor) /
total_stake;
            x_token_per_share_u256 = u256::add(
                u256::from_u128(last_magnified_dividends_per_share_x),
                u256::div(
                    u256::mul(u256::from_u64(reward_x), u256::from_u128(precision_factor)),
                    u256::from_u64(total_staked_token)
                )
            );
```

Suggestion:

It is recommended to add a feature to claim the remaining rewards.

SV2-19 Redundant Pair Creation Check in `init_rewards_pool()` Function

Severity: Minor

Status: Fixed

Code Location:

sources/swap_v2.move#282

Descriptions:

The purpose of the function `router_v2.create_rewards_pool ()` is to create a rewards pool for a pair of tokens (X, Y or Y, X).

```
public entry fun create_rewards_pool<X, Y>(
        sender: &signer,
        is_x_staked: bool
    ) {
        assert!(((swap_v2::is_pair_created<X, Y>() || swap_v2::is_pair_created<Y, X>())),
E_PAIR_NOT_CREATED);
        assert!(!((swap_v2::is_pool_created<X, Y>() || swap_v2::is_pool_created<Y, X>())),
E_POOL_EXISTS);

        if (swap_utils::sort_token_type<X, Y>()) {
            swap_v2::init_rewards_pool<X, Y>(sender, is_x_staked);
        } else {
            swap_v2::init_rewards_pool<Y, X>(sender, !is_x_staked);
        }
    }
```

After checking whether a pair of tokens (X, Y or Y, X) has been created using `swap_v2.is_pair_created()` and raising an error (E_PAIR_NOT_CREATED), the code proceeds to call `swap_v2.init_rewards_pool()`. However, within the `init_rewards_pool()` function, there is an additional check for the creation of the pair. This redundant pair creation check inside the `init_rewards_pool` function is unnecessary and duplicates the validation already performed in the calling function.

```
public(friend) fun init_rewards_pool<X, Y>(
```

```
        sender: &signer,
        is_x_staked: bool
    ) acquires SwapInfo {
        assert!(is_pair_created<X, Y>(), ERROR_PAIR_NOT_CREATED);
        assert!(!exists<TokenPairRewardsPool<X, Y>>(RESOURCE_ACCOUNT),
ERROR_ALREADY_INITIALIZED);

        let sender_addr = signer::address_of(sender);
```

Suggestion:

It is recommended to remove the redundant pair creation check inside the `init_rewards_pool()` function.

Resolution:

This issue has been fixed. The client followed our suggestions.

SV2-20 The Conventions for Using Boolean Values in Conditional Statements

Severity: Informational

Status: Fixed

Code Location:

sources/swap_v2.move#263,264,745,754,777,786,809,818,1271,1287,1306,1323,1340

Descriptions:

The code contains numerous conditional statements similar to the following:

```
assert!(xxx == true, 1);
```

```
if (xxx == true) {
...
}
```

Boolean values themselves represent true or false, so in general, there's no need to compare Boolean values explicitly to true.

Suggestion:

It is recommended to update these conditional statements to align with standard development practices.

Resolution:

This issue has been fixed. The client has modified it according to the suggestion.

SV2-21 Function Name Typo

Severity: Informational

Status: Fixed

Code Location:

sources/swap_v2.move#1277

Descriptions:

The function `swap_v2.ser_dex_treasury_fee()` is used to set dex treasury fee.

```
public(friend) fun ser_dex_treasury_fee(
        sender: &signer,
        new_fee: u128
    ) acquires SwapInfo {
        let swap_info = borrow_global_mut<SwapInfo>(RESOURCE_ACCOUNT);
        // assert sender is admin
        assert!(signer::address_of(sender) == swap_info.admin, ERROR_NOT_ADMIN);
        // assert new fee is not equal to the existing fee
        assert!(new_fee != swap_info.treasury_fee_modifier, 1);
        // assert the newer total fee is less than the threshold
        assert!(does_not_exceed_dex_fee_threshold(new_fee +
swap_info.liquidity_fee_modifier) == true, 1);
        // update the fee
        swap_info.treasury_fee_modifier = new_fee;
    }
```

The function name is incorrectly written as `ser_dex_treasury_fee()` and should be corrected to `set_dex_treasury_fee()`.

Suggestion:

It is recommended to change `ser_dex_treasury_fee()` to `set_dex_treasury_fee()`.

Resolution:

This issue has been fixed. The client followed our advice.

SV2-22 The `toggle_individual_token_rewards_fee()` Function's Functionality is Inconsistent With Its Comment

Severity: Informational

Status: Fixed

Code Location:

sources/swap_v2.move#795

Descriptions:

The function `swap_v2.toggle_individual_token_rewards_fee()` is used to toggle the reward fee.

```
// toggle liquidity fee
    public entry fun toggle_individual_token_rewards_fee<CoinType, X, Y>(
        sender: &signer,
        activate: bool,
    ) acquires TokenInfo, TokenPairMetadata {
        // assert sender is token owner
        assert!(is_token_owner<CoinType>(sender), ERROR_NOT_OWNER);
        // TODO: assert TokenInfo<CoinType> is registered in the pair
```

However, the accompanying comment misrepresents its purpose by stating "toggle liquidity fee" creating a discrepancy between the function's actual behavior and the provided comment.

Suggestion:

It is recommended to update the comment to "toggle reward fee".

Resolution:

This issue has been fixed. The client has deleted this function.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.