Fjord

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Tue Feb 25 2025

Fjord Audit Report

1 Executive Summary

1.1 Project Information

Description	The Fjord Move smart contract includes the Fixed Pool Contract, implementing Fixed Price (Tiered and Non-Tiered) Pools and Overflow Pools, and the Liquidity Bootstrap Pool Contract for managing liquidity bootstrap pools
Type	DeFi
Auditors	MoveBit
Timeline	Sat Jan 18 2025 - Tue Feb 25 2025
Languages	Move
Platform	Movement
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/marigoldlabs/fjord-move-contracts
Commits	989bea443bb9283c21aadc85d7df39d06afaf759 a52712cdc13bbc33b87eeb9847924d96d7f9cc4e 23a24cba3999b63a59c763cd3ca1d14cf3c56262 e06c010e9576fc4aba5de2359e618759d01865ea

1.2 Files in Scope

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

ID	File	SHA-1 Hash
OPO	fixed_pool/sources/overflow_pool.move	1113a31f3d6024a330f2788977b92d76ca34fd72
FPP	fixed_pool/sources/fixed_price_pool.move	7b7668fa3cbff01ef7560c2ad1fc97f180b426ee
DPL	fixed_pool/sources/lib/dynamic_price_lib.move	dd6c42d5fd7a4f7a254fc30f40a2bb9e13b3937f
FPL	fixed_pool/sources/lib/fixed_price_lib.move	b29fe8b9693131f98267989a8c45a814b5237b65
CON	fixed_pool/sources/lib/constant.move	07acfb9df363e30f521fbb298f7324765177b929
ERR	fixed_pool/sources/lib/error.move	f7d018c1aecef844577011a45651ce3f1acea755
PBT	fixed_pool/sources/lib/purchased_by_tier.move	ce8d62ef88cfe07bfdac47acc07f877b71d478ac
M64	fjord_libs/sources/math64.move	607510a9950b40c84ea59be6a5b933916fa786e1
MPR	fjord_libs/sources/merkle_proof.move	c0154d0157c4aecb1ffd519f40bbcd2036a3a2d8
INT	fjord_libs/sources/int.move	f56e478bb234507c54e27f167d135bd7a290395c
M25	fjord_libs/sources/math256.move	b7bdd083c3655909a0cf732e627b89bf508860eb

WEI	fjord_libs/sources/weighted.move	4fb1e18f08dc0a47c524e874727f273ebfc64ec2
FTR	liquidity_bootstrap_pool/sources/fee_treasury.move	3994e645aac5c0280fd157ee2561c9714b37ed98
MOV	fixed_pool/Move.toml	b8054bab186d2de479420120d307d49193622d71
RES	fixed_pool/sources/lib/reserve.move	166d9151c0b241d701bcadedc8e1dc35a8f4b74b
PIM	fixed_pool/sources/pool_impl.move	9856b489a077a4e53dfbb9fc76a88e07ed13305b
MOV1	fjord_libs/Move.toml	3b2c7f0870106e8c17ebf72cef6508cc5907782e
ASS	fjord_libs/sources/asset.move	abd5fb746afc9958c3f919033bebfdea25b50cd4
WHI	fjord_libs/sources/whitelist.move	e32b4aa14d460c38e8c59d62df65d432c53d9166
FPW	fjord_libs/sources/fixed_point_wad.move	4047c382a47269be662ca69afef25efe5335740b
ANT	fjord_libs/sources/antisnipe.move	2afccad2ac8f7d180c0de54535539e3f181f87a0
MOV3	liquidity_bootstrap_pool/Move.toml	00914dfde2a3e5b4b240e8af69d6d9f8480b51a6
RES1	liquidity_bootstrap_pool/sources/reserve.move	6d8aaca3136c5e79e1c26b90d17d14a392901292
LBL	liquidity_bootstrap_pool/sources/libs/liquidity_bootstrap_lib.move	a335c945ae9bc7463294dff12a147973cc25d2da

POO	liquidity_bootstrap_pool/sources/pool.move	23a490093fde3a9ed7abf01b8bb3b765890fe6d4

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	12	12	0
Informational	1	1	0
Minor	2	2	0
Medium	6	6	0
Major	2	2	0
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(Optional)

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 Fjord to identify any potential issues and vulnerabilities in the source code of the Fjord 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 12 issues of varying severity, listed below.

ID	Title	Severity	Status
ANT-1	Overflow in `get_scale_multiplier` Calculation When `decimals > WAD_DECIMALS`	Medium	Fixed
ANT-2	Missing Chain ID in Signature Validation	Medium	Fixed
LBL-1	The Calculation of `assets_in_scaled` is Incorrect	Critical	Fixed
LBL-2	When Swapping Assets for Shares, Round up the `assets_in` Value	Medium	Fixed
MOV-1	Swap Does Not Exclude Pool Addresses	Major	Fixed
MOV-2	Setting Optimization	Minor	Fixed
PIM-1	Missing the Function to Update the Public Key	Medium	Fixed
PIM-2	The Global Setting Information of the Fixed Pool cannot be Updated	Medium	Fixed
POO-1	Missing Slippage Protection in	Major	Fixed

	`create_liquidity_pool()`		
POO-2	Round up the `shares_in`	Medium	Fixed
POO-3	Repeatedly check the minimum value	Minor	Fixed
POO-4	Duplicate Check	Informational	Fixed

3 Participant Process

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

- `fixed_price_pool::create_with_both_coins` : Creates a new fixed-price pool using two coin types as the share and asset.
- `fixed_price_pool::create_with_coin_and_fa` : Creates a fixed-price pool using a coin as the share and a fungible asset as the asset.
- `fixed_price_pool::create_with_fa_and_coin` : Creates a fixed-price pool using a fungible asset as the share and a coin as the asset.
- `fixed_price_pool::create_with_both_fas` : Creates a fixed-price pool using two fungible assets as the share and asset.
- `fixed_price_pool::cancel_sale` : Cancels a fixed-price pool sale before it starts.
- `fixed_price_pool::toggle_pause` : Toggles the pause state of a fixed-price pool.
- `fixed_price_pool::close` : Closes the fixed-price pool and distributes funds if eligible.
- `overflow_pool::create_with_both_coins` : Creates an overflow pool using two coin types as the share and asset.
- `overflow_pool::create_with_coin_and_fa` : Creates an overflow pool using a coin as the share and a fungible asset as the asset.
- `overflow_pool::create_with_fa_and_coin` : Creates an overflow pool using a fungible asset as the share and a coin as the asset.
- `overflow_pool::create_with_both_fas` : Creates an overflow pool using two fungible assets as the share and asset.
- `overflow_pool::cancel_sale` : Cancels an overflow pool sale before it starts.
- `overflow_pool::toggle_pause` : Toggles the pause state of an overflow pool.
- `overflow_pool::close` : Closes the overflow pool and distributes funds if eligible.
- `fee_treasury::update_recipients` : Updates the list of fee recipients and their respective percentages.
- `fee_treasury::update_asset_swap_fee_recipient` : Sets a new recipient f

- `fee_treasury::update_share_swap_fee_recipient` : Sets a new recipient for share swap fees.
- `fee_treasury::distribute_fee` : Distributes fees from the treasury to recipients based on predefined percentages and processes swap fees for assets and shares.
- `pool::create_with_both_coins` : Creates a liquidity bootstrap pool using both shares and assets as standard coins.
- `pool::create_with_coin_and_fa` : Creates a liquidity bootstrap pool with shares as a coin and assets as a fungible asset.
- `pool::create_with_fa_and_coin` : Creates a liquidity bootstrap pool with shares as a fungible asset and assets as a coin.
- `pool::create_with_both_fas` : Creates a liquidity bootstrap pool using both shares and assets as fungible assets.
- `pool::toggle_pause` : Toggles the pause state of the pool to enable or disable swaps.
- `pool::cancel` : Cancels the pool before the sale starts and refunds all funds to the manager.
- `pool::close` : Closes the pool after the sale ends, distributing unsold shares and assets.
- `pool::close_with_seeding` : Closes the pool with seeding adjustments and distributes assets and shares.
- `pool::set_platform_fee` : Sets the platform fee percentage for the pool.
- `pool::set_swap_fee` : Sets the swap fee percentage for the pool.
- `pool::modify_settings` : Modifies platform and swap fees in a single transaction.
- `reserve::register` : Registers a reserves object containing shares and assets for a pool.
- `reserve::transfer_shares` : Transfers a specific amount of shares from the pool to a specified address.
- `reserve::transfer_assets` : Transfers a specific amount of
- `reserve::withdraw_assets` : Allows the pool owner to withdraw a specific amount of assets as fungible assets.
- `reserve::withdraw_shares` : Allows the pool owner to withdraw a specific amount of shares as fungible assets.

User

- `redeem` : Allows users to redeem shares or request refunds after a pool closes and redemption delay has passed.
- `fixed_price_pool::buy_exact_shares` : Allows users to purchase a specific number of shares from a fixed-price pool.
- `overflow_pool::buy_overflow` : Allows users to purchase a specific amount of shares in an overflow pool.
- `fee_treasury::deposit_shares_with_fa` : Deposits fungible asset shares into the fee treasury.
- `fee_treasury::deposit_assets_with_fa` : Deposits fungible asset assets into the fee treasury.
- `pool::swap_exact_assets_for_shares` : Swaps a specific amount of assets for a minimum number of shares, ensuring whitelist validation.
- `pool::swap_assets_for_exact_shares` : Swaps a specific number of shares for a maximum amount of assets, ensuring whitelist validation.
- `pool::swap_shares_for_exact_assets` : Swaps shares for a maximum amount of assets, ensuring whitelist validation.
- `pool::swap_exact_shares_for_assets` : Swaps a specific number of shares for a minimum amount of assets, ensuring whitelist validation.
- `reserve::deposit_shares_with_fa` : Deposits fungible asset shares into the pool's reserves.
- `reserve::deposit_assets_with_fa` : Deposits fungible asset assets into the pool's reserves.
- `reserve::deposit_shares_with_coin` : Deposits coin-based shares into the pool's reserves.
- `reserve::deposit_assets` : Deposits coin-based assets into the pool's reserves.
- `reserve::deposit_shares` : Deposits coin-based shares into the pool's reserves.

4 Findings

ANT-1 Overflow in `get_scale_multiplier` Calculation When `decimals > WAD_DECIMALS`

Severity: Medium

Status: Fixed

Code Location:

fjord_libs/sources/antisnipe.move#239

Descriptions:

```
public fun normalize(value: u64, decimals: u8): u256 {
    if (decimals < WAD_DECIMALS) {
        (value as u256) * get_scale_multiplier(decimals)
    } else if (decimals > WAD_DECIMALS) {
        (value as u256) / get_scale_multiplier(decimals)
    } else {
        (value as u256)
    }
}
```

```
inline fun get_scale_multiplier(decimals: u8): u256 {
    math256::pow(10, ((WAD_DECIMALS - decimals) as u256))
}
```

In the `normalize` , `denormalize_down` and `denormalize_up` function, when `decimals > WAD_DECIMALS` , the code calls `the get_scale_multiplier` function. Within `get_scale_multiplier` , the expression `WAD_DECIMALS - decimals` is comput `WAD_DECIMALS` and `decimals` are both unsigned integers, if `decimals` is greater than `WAD_DECIMALS` , this subtraction results in an underflow error.

Suggestion:

Consider using the absolute value of `WAD_DECIMALS - decimals` in `get_scale_multiplier` to avoid underflow, or preprocess the input to ensure valid parameters beget_scale_multiplier.

Resolution:

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

ANT-2 Missing Chain ID in Signature Validation

Severity: Medium

Status: Fixed

Code Location:

fjord_libs/sources/antisnipe.move#89

Descriptions:

```
fun validate_delegator_signature(
    account: address,
    recipient: address,
    amount: u64,
    deadline: u64,
    signature: vector<u8>
) acquires Antisniper {
    let state = borrow_global<Antisniper>(account);
    let current_nonce = *table::borrow_with_default(&state.nonces, recipient, &0u32);
    let message = build_message(recipient, amount, deadline, current_nonce + 1);
    let recovery_id = vector::pop_back(&mut signature);

    let maybe_pubkey =
        secp256k1::ecdsa_recover(
            message,
            recovery_id,
            &secp256k1::ecdsa_signature_from_bytes(signature)
        );

    let recovered_pubkey = option::extract(&mut maybe_pubkey);

    assert!(state.delegator_pubkey == recovered_pubkey, E_INVALID_SIGNATURE);
}
```

The `validate_delegator_signature` function does not incorporate a chain identifier (e.g., chain ID) into the signed message. Although the implementation includes a nonce mechanism to prevent signature replay within the same chain, this approach is insufficient to safeguard against replay attacks across different chains, such as between a testnet and a mainnet.

Suggestion:

Incorporate a unique chain identifier into the signed message. This ensures that the signature is valid only for a specific chain and cannot be reused across chain.

Resolution:

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

LBL-1 The Calculation of `assets_in_scaled` is Incorrect

Severity: Critical

Status: Fixed

Code Location:

liquidity_bootstrap_pool/sources/libs/liquidity_bootstrap_lib.move#88-93

Descriptions:

In the `preview_assets_in()` function, if `div_wad_down(assets_in_scaled, shares_out_scaled) > max_share_price`, the protocol calculates `assets_in_scaled` as follows:

```
assets_in_scaled = div_wad_down(
    shares_out_scaled,
    max_share_price
);
```

This calculation is **incorrect**. Here's why:

Mathematical Relationship

- The share price is defined as:
  `share_price = assets_in_scaled / shares_out_scaled`
- If the share price exceeds the `max_share_price`, the correct calculation for `assets_in_scaled` should be:
  `assets_in_scaled = shares_out_scaled * max_share_price`

Suggestion:

It is recommended to replace the incorrect division operation with the correct multiplication operation:

```
assets_in_scaled = mul_wad_down(shares_out_scaled, max_share_price);
```

Resolution:

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

LBL-2 When Swapping Assets for Shares, Round up the `assets_in` Value

Severity: Medium

Status: Fixed

Code Location:

liquidity_bootstrap_pool/sources/libs/liquidity_bootstrap_lib.move#95

Descriptions:

The function `swap_assets_for_exact_shares()` allows users to swap a specific number of shares for a maximum amount of assets. The `preview_assets_in_internal()` function is used to calculate the amount of assets the user needs to pay. Within the function, the protocol calculates the amount as follows: it rounds down the value, but it should round up instead.

```
public fun denormalize_down(value: u256, decimals: u8): u64 {
    if (decimals < WAD_DECIMALS) {
        (value / (get_scale_multiplier(decimals)) as u64)
    } else if (decimals > WAD_DECIMALS) {
        (value * (get_scale_multiplier(decimals)) as u64)
    } else {
        (value as u64)
    }
}
```

Suggestion:

It is recommended to change the function from `denormalize_down()` to `denormalize_up()` when swapping assets for shares.

Resolution:

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

MOV-1 Swap Does Not Exclude Pool Addresses

Severity: Major

Status: Fixed

Code Location:

fixed_pool/Move.toml

Descriptions:

Summary

Potential price control risks mean that the creator can manipulate prices to a certain extent.

Vulnerability Detail

```
fun swap_shares_for_assets<AssetCoin>(
    caller: &signer,
    pool: Object<PoolSettings>,
    settings: &PoolSettings,
    pool_state: &mut PoolState,
    recipient: address,
    assets_out: u64,
    shares_in: u64,
    share_reserve: u64,
    swap_fees: u64
) acquires EventStore, PoolSignerCapabilityStore {
    let pool_address = object::object_address(&pool);
    let pool_signer = &create_pool_signer();
    let pool_reserve = object::convert(pool);

    let recipient_address =
        if (recipient == @0x0) {
            signer::address_of(caller)
        } else {
            recipient
        };
```

Root Cause

No self-pool address is excluded.

Internal pre-conditions

Pool creators manipulate prices.

External pre-conditions

none

Attack Path

1. The creator creates a pool
2. When the price drops according to the expected release rate of LBP, the creator can set the recipient to the pool address when purchasing
3. Because all funds, assets and unused funds, shares will be sent to the creator when the pool is closed, the creator only loses gas
4. `pool_reserve` in the pool is affected and the price curve is manipulated
5. The creator can use any other anonymous EOA address
6. Users may suffer losses before the pool liquidity rolls

Impact

Pool Participating Users.

Suggestion:

It is recommended to add exclusion pool address check code:

```
let recipient_address =
    if (recipient == @0x0 || recipient == pool_address) {
        signer::address_of(caller)
    } else {
        recipient
    };
```

Resolution:

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

MOV-2 Setting Optimization

Severity: Minor

Status: Fixed

Code Location:

fixed_pool/Move.toml#1

Descriptions:

1. In the following code, when `max_assets_in` is set to 0, setting max_assets_in_deviation at the same time is meaningless.

```
// maxTotalAssetsInDeviation > 0.1% || maxTotalAssetsInDeviation < 1%
if (max_assets_in > 0) {
    if (max_assets_in_deviation < 10
        || max_assets_in_deviation > 100
        || min_assets_in > max_assets_in) {
        abort E_INVALID_MAX_ASSETS_CONFIG
    }
};
```

2. In the settings of the `validate_base_pool_settings` function, check that the expected range of `asset_decimals` is 2-18. In fact, there seems to be no token with a precision smaller than 6. We can modify the range to 6-18.

```
if (asset_decimals < 2 || asset_decimals > 18) {
    error::abort_invalid_asset_decimals();
};
```

3. Ambiguous function calls and usage, Proof acquisition is currently only used in test files, which may be `#[test-only]` or redundant code.

```
public fun empty_root(): vector<u8> {
    vector[]
}

public fun empty_proof(): vector<vector<u8>> {
```

```
    vector[]
}
```

Suggestion:

It is recommended to modify the inspection scope of the settings to cover the expected design.

Resolution:

The client adopted our suggestions and the remaining parts will be manually checked to confirm their correctness.

PIM-1 Missing the Function to Update the Public Key

Severity: Medium

Status: Fixed

Code Location:

fixed_pool/sources/pool_impl.move#247

Descriptions:

In the `pool_impl` contract, the code hardcodes the delegator public key initially, and there is no function to update it.

```
// Hardcode the delegator public key initially, may add entry to update if needed
    let delegator_pubkey=
x"5e9eebe7176d9a1a34f99eaab408d772e1f2b2d3543e07c942d8cc96fe97ef8b9cebcae6629a

    move_to(
      publisher,
      GlobalSettings {
        fee_recipient: @fixed_pool_fee_recipient,
        delegator_pubkey,
      }
    );
```

Suggestion:

It is recommended to add a function that allows updating the delegator public key.

Resolution:

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

PIM-2 The Global Setting Information of the Fixed Pool cannot be Updated

Severity: Medium

Status: Fixed

Code Location:

fixed_pool/sources/pool_impl.move#249-255

Descriptions:

In the liquidity bootstrap pool, we noticed that the protocol supports the factory owner to update the global setting information.

```
public entry fun modify_settings(
        caller: &signer,
        platform_fee: u64,
        swap_fee: u64
    ) acquires GlobalSettings, GlobalEventStore {
        assert!(is_fc(caller), E_UNAUTHORIZED);
        assert!(is_valid_bips(platform_fee), E_MAX_FEE_EXCEEDED);
        assert!(is_valid_bips(swap_fee), E_MAX_FEE_EXCEEDED);

        let global_settings = borrow_mut_global_settings();
        let global_event_store = borrow_mut_global_event_store();

        global_settings.platform_fee = from_bips(platform_fee);
        global_settings.swap_fee = from_bips(swap_fee);

        event::emit_event(
            &mut global_event_store.platform_fee_set_handle,
            PlatformFeeSet { fee: platform_fee }
        );
        event::emit_event(
            &mut global_event_store.swap_fee_set_handle,
            SwapFeeSet { fee: swap_fee }
        );
    }
```

However, in the fixed pool contract, this functionality is not mentioned

Suggestion:

It is recommended to add a function that allows updating the information.

Resolution:

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

POO-1 Missing Slippage Protection in `create_liquidity_pool()`

Severity: Major

Status: Fixed

Code Location:

liquidity_bootstrap_pool/sources/pool.move#1269-1277

Descriptions:

In the `create_liquidity_pool()` function, the code calls `mosaic::router::add_liquidity_both_assets()` to add liquidity to the pool.

```
let (x_left, y_left, lp_tokens) =
    mosaic::router::add_liquidity_both_assets(
        pool_signer,
        lp,
        shares,
        0,
        assets,
        0
    );
```

However, the function does not implement slippage protection, making it vulnerable to sandwich attacks.

Suggestion:

It is recommended to implement slippage protection.

Resolution:

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

POO-2 Round up the `shares_in`

Severity: Medium

Status: Fixed

Code Location:

liquidity_bootstrap_pool/sources/pool.move#2094

Descriptions:

The function allows users to exchange a certain number of shares for assets. In the function, the protocol calculates the `shares_in` as follows:

```
public fun preview_shares_in(
        asset_balance: u64,
        share_balance: u64,
        virtual_assets: u64,
        total_purchased: u64,
        sale_end: u64,
        sale_start: u64,
        weight_start: u256,
        weight_end: u256,
        share_decimals: u8,
        asset_decimals: u8,
        assets_out: u64,
        max_share_price: u256
    ): u64 {
        let (asset_reserve, share_reserve, asset_weight, share_weight) =
            compute_reserves_and_weights(
                asset_balance,
                share_balance,
                virtual_assets,
                total_purchased,
                sale_end,
                sale_start,
                weight_start,
                weight_end
            );

        let share_reserve_scaled = normalize(share_reserve, share_decimals);
        let asset_reserve_scaled = normalize(asset_reserve, asset_decimals);
```

```
    let assets_out_scaled = normalize(assets_out, asset_decimals);

    let shares_in =
        weighted::get_amount_in(
            assets_out_scaled,
            share_reserve_scaled,
            asset_reserve_scaled,
            share_weight,
            asset_weight
        );

    if (div_wad_down(assets_out_scaled, shares_in) > max_share_price) {
        shares_in = div_wad_down(assets_out_scaled, max_share_price);
    };

    denormalize_down(shares_in, share_decimals)
}
```

It should round up instead of rounding down.

Suggestion:

It is recommended to round up the `shares_in` instead of rounding down.

Resolution:

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

POO-3 Repeatedly check the minimum value

Severity: Minor

Status: Fixed

Code Location:

liquidity_bootstrap_pool/sources/pool.move#1917

Descriptions:

Duplicate check of `settings.min_assets_in > 0` in `swap_exact_assets_for_shares_return_shares_out` and `swap_assets_for_exact_shares_return_assets_in` functions.

```
public fun swap_exact_assets_for_shares_return_shares_out<ShareCoin, AssetCoin>(
...
        if (settings.min_assets_in > 0) {
            handle_min_reserve(
                settings,
                state,
                assets_in - swap_fees,
                recipient
            );
        };
...
    fun handle_min_reserve(
        settings: &PoolSettings,
        state: &mut PoolState,
        assets_in_minus_fees: u64,
        recipient: address
    ) {
        if (settings.min_assets_in > 0) {
            let contributed_assets =
                table::borrow_mut_with_default(
                    &mut state.assets_contributed,
                    recipient,
                    0
                );
            *contributed_assets = *contributed_assets + assets_in_minus_fees;
```

```
    }
}
```

Suggestion:

It is recommended to remove redundant checks.

Resolution:

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

POO-4 Duplicate Check

Severity: Informational

Status: Fixed

Code Location:

liquidity_bootstrap_pool/sources/pool.move#2372-2373

Descriptions:

In the `close()` function, the protocol first verifies if it is closable and if the sale has ended.

```
public entry fun close<ShareCoin, AssetCoin>(
    pool: Object<PoolSettings>
) acquires PoolState, PoolSettings, EventStore, PoolSignerCapabilityStore {
    let pool_address = object::object_address(&pool);
    let state = borrow_state_mut(pool_address);
    let settings = borrow_settings(pool_address);

    assert_closable(settings, state);
    assert_sale_ended(settings);
    assert_not_lp_rollover(settings);

    close_internal<ShareCoin, AssetCoin>(
        pool,
        settings,
        state
    );
}
```

In the `canClose()` function, the protocol checks if `timestamp::now_seconds() < settings.sale_end`.

```
inline fun can_close(settings: &PoolSettings, pool: &PoolState): bool {
    if (pool.closed
        || (timestamp::now_seconds() < settings.sale_end
            && pool.actual_sale_end == 0)) { false }
    else { true }
}
```

In the `is_sale_ended()` function, the protocol checks if `timestamp::now_seconds() >= settings.sale_end`.

```
inline fun is_sale_eneded(settings: &PoolSettings): bool {
    timestamp::now_seconds() >= settings.sale_end
}
```

These checks are redundant.

Suggestion:

It is recommended to remove the `assert_sale_ended()` check.

Resolution:

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

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.