Hyperion

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Wed Feb 19 2025

Hyperion Audit Report

1 Executive Summary

1.1 Project Information

Description	Hyperion is a fully onchain hybrid orderbook AMM DEX built natively for Aptos
Type	AMM
Auditors	MoveBit
Timeline	Mon Jan 20 2025 - Wed Feb 19 2025
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Hyperionxyz/dex-v3
Commits	6b0da5c668125dedda668a8587e487674124d288 109c66ae7c50fbc13e3309430a2c2bb2f3b74f63 fd37942b26cb67af51be8a11e898ff315e0853cd 7fb51ab9fe158265b0239d9790e5ba3c9827ce46

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV	Move.toml	9fad77eae03c011fa7687c24b3c022a6d75b5365
LP	sources/v3/lp.move	d414b8ed6f80acf49dc12eab7131e9e5be82fb36
CWR	sources/v3/coin_wrapper.move	0cc33194b80c4cb5660fc3dd9b306a674d9a351b
TBI	sources/v3/tick_bitmap.move	63a4edfece3f1de51e475b0c2926166a9e264905
SMA	sources/math/swap_math.move	845f4ce9d67454149cf32d4877ac1195d6d63798
MU2	sources/math/math_u256.move	43ac449ec7866cfb41fffc39c0feb65e20c178d8
MU1	sources/math/math_u128.move	15da3d8499b6a2070705c1e5cdb5743f62071854
FMU6	sources/math/full_math_u64.move	7d733d9837e83a9878afbcac609250a48ecc0ed1
FMU1	sources/math/full_math_u128.move	2622523d07307a80e3bf314336c169b1257c516c
LMA	sources/math/liquidity_math.move	2626f002c8df5f4d8e11ba8e9f6588aa66fce5c8
I32	sources/math/i32.move	a51e803ec56b83cd27a6ca2b1cee1b9c26af9f3a

I12	sources/math/i128.move	181319dc02974d19c50399a60e0d0b8d223fa4c3
I64	sources/math/i64.move	7768a5073f21c3b3ff2a0932e09ccd7af953b13f
BMA	sources/math/bit_math.move	eb760461f9f7fb03a9bf0fb0d93d00555f455958
TMA	sources/math/tick_math.move	1119b0d18b8f39ac5722c5f29efe514c55319142
RAD	sources/adapter/router_adapter.move	3ed39b97c48f4061db11a434793348c26654232a
PV3	sources/v3/pool_v3.move	5e140b4c1bf86710a20a033ae4d68fa63ff8372c
REW	sources/v3/rewarder.move	808ffb9a2ade46c8c43e9a81441b875229eb1fe2
UTI	sources/v3/utils.move	1db160d9356734002637e30d149c4d7abf31b43a
PV31	sources/v3/position_v3.move	bdeefaa33750b1421bb6c6f4923d19ff1060578e
TIC	sources/v3/tick.move	4d90876f4f2121021bcf2fc3a6526c08019e8cec
PBL	sources/v3/position_blacklist.move	577ea4007ce5e8be4e52b5d507ae5155432e119b
RV3	sources/v3/router_v3.move	4dce645dc14a4ca014389248e2c4c0f9d4d05757
PMA	sources/v3/package_manager.move	8cf1ae3174061552e1a3ebaa6e858a764962319d

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	12	12	0
Informational	2	2	0
Minor	2	2	0
Medium	7	7	0
Major	1	1	0
Critical	0	0	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 Hyperion to identify any potential issues and vulnerabilities in the source code of the Hyperion 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
PV3-1	Incorrect Comparison Logic	Medium	Fixed
PV3-2	Swap Fee Should Be Rounded Down	Medium	Fixed
PV3-3	Missing Pause Check and Ability to Change Pause State	Medium	Fixed
PV3-4	Missing Check for Duplicate Currency Pools	Medium	Fixed
REW-1	Incorrect Implementation of `claim_rewards`	Major	Fixed
REW-2	The Check for whether `pool_liquidity` is zero is Missing	Medium	Fixed
REW-3	Pause Should Be Enabled When Removing All Incentives	Minor	Fixed
REW-4	When `time_delta` is Zero, the while loop becomes Unnecessary	Informational	Fixed
RV3-1	Optimize the Order of Checks	Informational	Fixed

TIC-1	Incorrect Initialization	Minor	Fixed
TMA-1	Missing Tick Validation	Medium	Fixed
UTI-1	Infinite Recursion Risk in `is_sorted` Function	Medium	Fixed

3 Participant Process

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

Admin

- `add_rewarder` : Adds a new rewarder
- `add_incentive` : Adds a reward incentive to a specified liquidity pool.
- `remove_incentive` : Removes a reward incentive from a specified liqudity pool.
- `update_emissions_rate` : Adjusts the rate at which reward tokens are emitted.
- `block_position` : Adds a specific liquidity position to the blacklist.
- `remove_position_block` : Removes a specific liquidity position from the blacklist.
- `pause/restart_rewarder_manager` : Pauses or restarts the rewarder manager to stop or resume reward distribution.
- `claim_protocol_fees_all` : Claims all accumulated protocol fees from a liquidity pool.

LP

- `create_pool` : Initializes a new liquidity pool for a specific token (or coin) pair.
- `create_liquidity` : Sets up initial liquidity provisions for a pool.
- `add_liquidity` : Deposits tokens into a pool to provide liquidity.
- `remove_liquidity` : Withdraws liquidity from a pool, reclaiming tokens.
- `open_position` : Opens a new liquidity position in the specified pool.
- `claim_fees` : Collects earned fees from a position.
- `claim_rewards` : Claims incentive rewards.

User

- `exact_input_swap_entry` : Executes a token swap where the input amount is fixed.
- `exact_output_swap_entry` : Executes a token swap where the output amount is fixed.
- `swap_batch` : Processes multiple token swaps in a single transaction.

4 Findings

PV3-1 Incorrect Comparison Logic

Severity: Medium

Status: Fixed

Code Location:

sources/v3/pool_v3.move#1277

Descriptions:

The contract uses incorrect comparison operators in multiple places when performing tick comparisons. For example, in the `merge_into_pool` function:

```
if (i32::lte(pool.tick, tick_lower)) {
```

Here, `<=` is used, but the correct logic should be `<`. This could lead to errors in various tick-related calculations.

Examples of Incorrect Comparisons

1. In price limit checks:

```
if(a2b) {
    assert!(
        sqrt_price_limit < pool_mut.sqrt_price && sqrt_price_limit >=
tick_math::min_sqrt_price(),
        ESQRT_PRICE_LIMIT_UNAVAILABLE
    );
} else {
    assert!(
        sqrt_price_limit > pool_mut.sqrt_price && sqrt_price_limit <=
tick_math::max_sqrt_price(),
        ESQRT_PRICE_LIMIT_UNAVAILABLE
    );
};
```

2. In fee growth calculations:

```
let (fee_growth_above_a, fee_growth_above_b) = if (i32::lte(tick_current, tick_upper))
{
```

Suggestion:

Refer to Uniswap's implementation and adjust the comparison logic to ensure correctness.

Resolution:

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

PV3-2 Swap Fee Should Be Rounded Down

Severity: Medium

Status: Fixed

Code Location:

sources/v3/pool_v3.move#1550-1555

Descriptions:

When updating `feeGrowthGlobalX128` during a swap, the calculation should round down instead of rounding up. The protocol's security should always take precedence over user benefits.

```
// update global fee tracker
if (state.liquidity > 0) {
    state.fee_growth_global = state.fee_growth_global +
        full_math_u128::mul_div_ceil((fee_amount as u128), Q64, state.liquidity);
};
```

This could lead to unintended precision issues, favoring users over the protocol.

Uniswap V3 correctly uses **floor rounding**:

```
// update global fee tracker
if (state.liquidity > 0)
    state.feeGrowthGlobalX128 += FullMath.mulDiv(step.feeAmount, FixedPoint128.Q128,
state.liquidity);
```

Suggestion:

Modify the rounding direction to **floor rounding** to ensure correct fee accumulation.

Resolution:

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

PV3-3 Missing Pause Check and Ability to Change Pause State

Severity: Medium

Status: Fixed

Code Location:

sources/v3/pool_v3.move#39

Descriptions:

The `LiquidityPoolConfigsV3` struct includes a `is_paused` field, indicating that the protocol has a pause feature. However, the implementation lacks functionality to check the pause state and modify it.

```
struct LiquidityPoolConfigsV3 has key {
    all_pools: SmartVector<Object<LiquidityPoolV3>>,
    is_paused: bool,
    fee_manager: address,
    pauser: address,
    pending_fee_manager: address,
    pending_pauser: address,
    tick_spacing_list: vector<u64>
}
```

Suggestion:

It is recommended to improve the pause functionality.

Resolution:

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

PV3-4 Missing Check for Duplicate Currency Pools

Severity: Medium

Status: Fixed

Code Location:

sources/v3/pool_v3.move#336

Descriptions:

The contract does not validate whether `token_a` and `token_b` are identical when creating a pool. Allowing pools with the same token could lead to undefined behavior.

Suggestion:

Before creating a pool, add a validation step to ensure `token_a` and `token_b` are not the same.

```
assert!(token_a != token_b, ERROR_IDENTICAL_TOKENS);
```

Resolution:

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

REW-1 Incorrect Implementation of `claim_rewards`

Severity: Major

Status: Fixed

Code Location:

sources/v3/rewarder.move#219-279

Descriptions:

In the `claim_rewards` function, if the user has not previously called the `claim_rewards` function, or the administrator has added new rewarders, the `reward_length` will be less than `manager_length + 1`. In this case, a new ticket is created to participate in reward claiming. The `seconds_per_liquidity_inside` of the new ticket is set to `0`, while `position_seconds_per_liquidity_inside` is derived from `pool.seconds_per_liquidity_global`, which keeps increasing. This allows the user to claim more rewards, including rewards accrued before the user added liquidity. Even if the user adds new liquidity, they can immediately claim rewards.

```
public(package) fun claim_rewards(
    pool_signer: &signer,
    user: address,
    position_id: address,
    reward_manager: &mut RewarderManager,
    reward_tickets: vector<PositionReward>,
    seconds_per_liquidity_global_current: u128,
    position_seconds_per_liquidity_inside: u128,
    pool_liquidity: u128,
    position_liquidity: u128
): (vector<FungibleAsset>, vector<PositionReward>) {
    let pool_id = signer::address_of(pool_signer);
    flash(reward_manager, seconds_per_liquidity_global_current, pool_liquidity);
    let rewards_list = vector::empty<FungibleAsset>();
    let manager_length = vector::length(&reward_manager.rewarders);
    let reward_length = vector::length(&reward_tickets);
    let reward_ticket_van = vector::empty<PositionReward>();
    assert!(manager_length >= reward_length, EREWARDS_LENGTH_ERR);
    let i = 0;
    while(manager_length != 0) {
```

```
            manager_length -= 1;
            let manager = reward_manager.rewarders.borrow_mut(manager_length);
            let ticket = if((reward_length) < (manager_length+1)) {
                reward_ticket_van.push_back(PositionReward{
                    seconds_per_liquidity_inside: 0,
                    amount_owned: 0,
                });
                reward_ticket_van.borrow_mut(i)
            } else {
                reward_tickets.borrow_mut(manager_length)
            };
            let delta_seconds_per_liquidity_inside =
                position_seconds_per_liquidity_inside
                    - ticket.seconds_per_liquidity_inside;
            let amount_owed = ticket.amount_owned +
                ((((delta_seconds_per_liquidity_inside * position_liquidity ) >> 64)
                    * (manager.emissions_per_second as u128)) as u64);
            let reward = dispatchable_fungible_asset::withdraw(
                pool_signer,
                manager.reward_store,
                amount_owed
            );
            ticket.amount_owned = 0;
            ticket.seconds_per_liquidity_inside = position_seconds_per_liquidity_inside;
            manager.user_owed = manager.user_owed - amount_owed;
            event::emit(ClaimRewardsEvent{
                pool_id,
                position_id,
                reward_fa: fungible_asset::store_metadata(manager.reward_store),
                amount: amount_owed,
                owner: user,
                index: manager_length
            });
            i += 1;
            rewards_list.push_back(reward);
        };
        rewards_list.reverse();
        vector::reverse(&mut reward_ticket_van);
        vector::append(&mut reward_tickets, reward_ticket_van);
        (rewards_list, reward_tickets)
    }
```

Suggestion:

Adjust the reward distribution logic to resolve this issue.

Resolution:

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

REW-2 The Check for whether `pool_liquidity` is zero is Missing

Severity: Medium

Status: Fixed

Code Location:

sources/v3/rewarder.move#287

Descriptions:

The `pool_v3.add_incentive()` function enables an admin to deposit a specified amount of reward tokens into a liquidity pool as an incentive. During this process, the protocol calls the `flash()` function to update the reward information. Within this logic, the `delta_emissions_per_liquidity` is calculated using the formula:

```
delta_emissions_per_liquidity = ((delta as u128) << 64) / pool_liquidity
```

Here, `pool_liquidity` is derived from the `get_incentive_liquidity()` function. However, there is a issue: the protocol does not verify whether `pool_liquidity` is zero before performing the division. If `pool_liquidity` is zero, the division operation will result in a runtime error, causing the function to fail unexpectedly.

Suggestion:

It is recommended to check whether `pool_liquidity` is zero.

Resolution:

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

REW-3 Pause Should Be Enabled When Removing All Incentives

Severity: Minor

Status: Fixed

Code Location:

sources/v3/rewarder.move#144-168

Descriptions:

In the `remove_incentive` function, if `total - rewarder.user_owed == amount`, the `pause` flag should be set to `true`, indicating that rewards can no longer be distributed.

```
public(package) fun remove_incentive(
    pool_signer: &signer,
    manager: &mut RewarderManager,
    seconds_per_liquidity_global_current: u128,
    pool_liquidity: u128,
    index: u64,
    amount: u64
): FungibleAsset {
    flash(manager, seconds_per_liquidity_global_current, pool_liquidity);
    let rewarder = vector::borrow_mut(&mut manager.rewarders, index);
    assert!(!rewarder.pause, EREWARD_TOO_LESS_TO_REMOVE);
    let total = fungible_asset::balance(rewarder.reward_store);
    assert!(total - rewarder.user_owed >= amount, EINSUFICIEENT_BALANCE);
    event::emit(RemoveIncentiveEvent{
        pool_id: signer::address_of(pool_signer),
        reward_metadata: fungible_asset::store_metadata(rewarder.reward_store),
        amount,
        index
    });
    if total - rewarder.user_owed == amount {
        rewarder.pause = true; // Set pause to true when all incentives are removed
    }
    dispatchable_fungible_asset::withdraw(
        pool_signer,
        rewarder.reward_store,
        amount
    )
}
```

Suggestion:

When all incentives are removed, set `pause` to `true`.

Resolution:

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

REW-4 When `time_delta` is Zero, the while loop becomes Unnecessary

Severity: Informational

Status: Fixed

Code Location:

sources/v3/rewarder.move#277

Descriptions:

In the `flash()` function, if `time_now` is equal to `manager.last_updated_time`, the `time_delta` will be zero. In this case, the while loop becomes unnecessary because there is no time difference to process, and the loop will not perform any meaningful operations.

```
let time_now = timestamp::now_seconds();
        let time_delta = time_now - manager.last_updated_time;

        let i = 0;
        let length = vector::length(&manager.rewarders);
        // skip update rewarder when manager paused
        if(manager.pause) length = 0;
        while(i < length) {
            let rewarder = vector::borrow_mut(&mut manager.rewarders, i);
            if(!rewarder.pause) {
                let delta = rewarder.emissions_per_second * time_delta;
                let delta_emissions_per_liquidity = ((delta as u128) << 64) / pool_liquidity;
                let increment = if(delta < fungible_asset::balance(rewarder.reward_store)) {
                    rewarder.emissions_per_liquidity_latest =
                        rewarder.emissions_per_liquidity_latest + delta_emissions_per_liquidity;
                    delta
```

Suggestion:

It is recommended to add an early return condition when `time_delta` is zero.

Resolution:

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

RV3-1 Optimize the Order of Checks

Severity: Informational

Status: Fixed

Code Location:

sources/v3/router_v3.move#663

Descriptions:

The assert! statement is placed after computations and state changes, causing unnecessary gas consumption if the condition fails. Moving it earlier ensures the transaction aborts before expensive operations, saving gas.

```
assert!(amount_out >= amount_out_min, EAMOUNT_OUT_TOO_LESS);
```

Suggestion:

Moving it earlier ensures the transaction aborts before expensive operations, saving gas.

Resolution:

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

TIC-1 Incorrect Initialization

Severity: Minor

Status: Fixed

Code Location:

sources/v3/tick.move#76-141

Descriptions:

In the `update` function of the tick contract, the `initialized` field of the tick is always set to `true`. However, initialization should only occur when `liquidity_gross_before == 0`.

```
let fee_growth_updated = if (liquidity_gross_before == 0) {
    if(i32::lte(tick_to_update, tick_current)) {
        info.fee_growth_outside_a = fee_growth_global_a;
        info.fee_growth_outside_b = fee_growth_global_b;
        info.seconds_per_liquidity_oracle_outside = seconds_per_liquidity_oracle;
        info.seconds_per_liquidity_incentive_outside = seconds_per_liquidity_incentive;
        info.emissions_per_liquidity_incentive_outside = emissions_per_liquidity;
        true
    } else {
        false
    }
} else {
    false
};
info.initialized = true;
```

Suggestion:

Modify the redundant initialization logic to ensure that `initialized` is only set to `true` when necessary.

Resolution:

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

TMA-1 Missing Tick Validation

Severity: Medium

Status: Fixed

Code Location:

sources/math/tick_math.move#40-44

Descriptions:

When adding or removing liquidity, there is no validation to ensure that `tick_lower` and `tick_upper` are within the valid range. The absence of this check may lead to unexpected issues.

```
public fun check_tick(tick_lower: I32, tick_upper: I32) {
    assert!(i32::lt(tick_lower, tick_upper), ETICK_LOWER_BIGGER_THAN_UPPER);
    assert!(i32::gte(tick_lower, min_tick()), ETICK_LOWER_BEYOND_MINIMUM);
    assert!(i32::lte(tick_upper, max_tick()), ETICK_UPPER_BEYOND_MAXIMUM);
}
```

Suggestion:

Ensure this validation is applied when adding and removing liquidity.

Resolution:

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

UTI-1 Infinite Recursion Risk in `is_sorted` Function

Severity: Medium

Status: Fixed

Code Location:

sources/v3/utils.move#8-13

Descriptions:

Outer functions call `is_sorted()` to determine if two tokens are sorted. However, if two identical tokens are passed as input, `is_sorted()` returns `false` because it relies on `is_smaller_than()`. As a result, the outer function swaps the order of the arguments and calls `is_sorted()` again, leading to an infinite recursion and excessive gas consumption.

```
#[view]
public fun is_sorted(token_1: Object<Metadata>, token_2: Object<Metadata>): bool {
    let token_1_addr = object::object_address(&token_1);
    let token_2_addr = object::object_address(&token_2);
    comparator::compare(&token_1_addr, &token_2_addr).is_smaller_than()
}
```

Suggestion:

To prevent infinite recursion, redesign the `is_sorted()` logic without using `is_smaller_than()`. Implement a custom comparison or validation mechanism that avoids recursive calls and argument swapping.

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.