HyperFluid

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Thu Nov 28 2024

HyperFluid Audit Report

1 Executive Summary

1.1 Project Information

Description	A decentralized exchange (DEX) on Aptos that offers LP staking rewards
Type	DeFi
Auditors	MoveBit
Timeline	Fri Nov 15 2024 - Thu Nov 28 2024
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Hyperfluid/dex
Commits	d2038470ccc55774abe0b01e43e6b930c21cb8a3 79b3ec7defe8a1cbb9b1b06c2993b6d33fbe39a1 a824216a79f6cefc24356c01bb35f2abbe96d133

1.2 Files in Scope

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

ID	File	SHA-1 Hash
CWR	sources/coin_wrapper.move	0cc33194b80c4cb5660fc3dd9b306a674d9a351b
PMA	sources/package_manager.move	d49c4c50306276970b5b1f377e2ef7c2e39f763e
MOV	Move.toml	9fad77eae03c011fa7687c24b3c022a6d75b5365
ROU	sources/router.move	611df3ffe12b7c0fce9a2765a88a93bcd2d743c8
RAD	sources/router_adapter.move	200c42b2d98fe8e002b245be248ddf00f98be919
LPO	sources/liquidity_pool.move	97dd436a28cd14eefc267138132a85c80efbd9b0
MAS	sources/masterchef.move	5128e64156fb480d3138ebff6363444b72b791db

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	8	6	2
Informational	1	1	0
Minor	3	1	2
Medium	3	3	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 HyperFluid to identify any potential issues and vulnerabilities in the source code of the Dex 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 8 issues of varying severity, listed below.

ID	Title	Severity	Status
LPO-1	The Setting of Fee Basis Points Has No Maximum Limit	Medium	Fixed
LPO-2	Mint and Burn Lack Pause Functionality	Medium	Fixed
LPO-3	Redundant `update_user_debt` Call	Minor	Acknowledged
LPO-4	Unused Constants	Informational	Fixed
MAS-1	Remaining Rewards Should Not Be Calculated Based on Balance	Major	Fixed
MAS-2	Redundant `acc_per_share` Update	Minor	Acknowledged
MAS-3	Unused Private Function	Minor	Fixed
MOV-1	No Upgrade Policies	Medium	Fixed

3 Participant Process

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

- `mint` : Adds liquidity to the pool and mints LP tokens.
- `burn` : Burns LP tokens to withdraw liquidity from the pool.
- `claim_fee` : Claims the accrued fees for the LP tokens from the liquidity pool.
- `swap` : Swaps one token for another in the liquidity pool while ensuring constant product and accounting for fees.
- `deposit` : Users deposit LP tokens into the pool, creating or updating their stake and rewards.
- `withdraw` : Users withdraw LP tokens from the pool and update their rewards.
- `claim_pending_rewards_entry` : Claims pending rewards for multiple liquidity pool (LP) addresses. **Admin**
- `add_incentive` : Adds a new incentive to the liquidity pool by updating the reward per second and the total reward over time.
- `close_incentive` : Closes the incentive for the specified pool and records the final reward distribution.

Pauser

- `set_pauser` : Assigns a new pauser for the liquidity pool.
- `set_pause` : Toggles the pause status of the liquidity pool.

Fee manager

- `set_fee_manager` : Assigns a new fee manager for the liquidity pool.
- `set_stable_fee` : Sets the fee rate for stable liquidity pools.
- `set_volatile_fee` : Sets the fee rate for volatile liquidity pools.

4 Findings

LPO-1 The Setting of Fee Basis Points Has No Maximum Limit

Severity: Medium

Status: Fixed

Code Location:

sources/liquidity_pool.move#1010-1018

Descriptions:

The maximum swap fee basis points in the contract is 25%, but there is no check for this maximum limit when setting the fees.

```
const MAX_SWAP_FEES_BPS: u64 = 25; // 0.25%
```

```
public entry fun set_stable_fee(fee_manager: &signer, new_fee_bps: u64) acquires LiquidityPoolConfigs {
    let pool_configs = fee_manager_only_mut_liquidity_pool_configs(fee_manager);
    pool_configs.stable_fee_bps = new_fee_bps;
}

public entry fun set_volatile_fee(fee_manager: &signer, new_fee_bps: u64) acquires LiquidityPoolConfigs {
    let pool_configs = fee_manager_only_mut_liquidity_pool_configs(fee_manager);
    pool_configs.volatile_fee_bps = new_fee_bps;
}
```

Suggestion:

It is recommended to add this maximum value check.

Resolution:

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

LPO-2 Mint and Burn Lack Pause Functionality

Severity: Medium

Status: Fixed

Code Location:

sources/liquidity_pool.move#374

Descriptions:

The liquidity pool contract includes a pause functionality, which is checked during swap operations. For example:

```
assert!(!safe_liquidity_pool_configs().is_paused, ESWAPS_ARE_PAUSED);
```

However, the mint and burn operations do not include this pause check.

Suggestion:

It is recommended to add pause judgment in mint and burn.

Resolution:

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

LPO-3 Redundant `update_user_debt` Call

Severity: Minor

Status: Acknowledged

Code Location:

sources/liquidity_pool.move#761

Descriptions:

In the `burn` function, if `primary_fungible_store::balance(signer::address_of(lp), pool) == 0` (indicating the user has burned all their LP tokens), the `claim_fees` function is invoked internally to claim fees. However, `claim_fees` already calls `update_user_debt` as follows:

```
update_user_debt(signer::address_of(lp), unchecked_mut_fees_accounting(&pool), old_lp_amount, lp_amount);
```

Despite this, `update_user_debt` is called again at the end of the burn function, which is redundant.

Suggestion:

It is recommended to remove the final call to the `update_user_debt` function in the `burn` function.

LPO-4 Unused Constants

Severity: Informational

Status: Fixed

Code Location:

sources/liquidity_pool.move#31

Descriptions:

There are unused constants in the contract.

```
const EINCORRECT_SWAP_AMOUNT: u64 = 4;
const ENOT_STORE_OWNER: u64 = 5;
const ERROR_COIN_NOT_PUBLISHED: u64 = 1;
const ERROR_WITHDRAW_INSUFFICIENT: u64 = 4;
const ERROR_COIN_NOT_REGISTERED: u64 = 7;
const ERROR_POOL_USER_INFO_NOT_EXIST: u64 = 10;
const ERROR_ZERO_ACCOUNT: u64 = 11;
```

Suggestion:

It is recommended to remove unused constants if there's no further design.

Resolution:

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

MAS-1 Remaining Rewards Should Not Be Calculated Based on Balance

Severity: Major

Status: Fixed

Code Location:

sources/masterchef.move#483-486

Descriptions:

The MasterChef contract calculates the remaining rewards using the balance, as shown below:

```
let remaining = primary_fungible_store::balance(
    object::object_address(&pool_info_object),
    metadata
);
```

This approach is incorrect. For example, in the `close_incentive function`, using this method to calculate the remaining rewards can result in unclaimed rewards for previously staked users because some rewards might still be unclaimed. The same issue exists in the `add_incentive` function.

Suggestion:

It is recommended to introduce an additional variable to explicitly track the quantity of remaining reward tokens.

Resolution:

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

MAS-2 Redundant `acc_per_share` Update

Severity: Minor

Status: Acknowledged

Code Location:

sources/masterchef.move#496

Descriptions:

In the add_incentive function, if `pools.pool_mapping.contains(metadata)` returns true, the `update_pool_token` function is called to update the pool's reward information, including `acc_per_share` and `last_reward_timestamp`. However, the subsequent logic still updates `acc_per_share` again:

```
pool_info.acc_per_share += (pool_info.reward_per_sec as u128) * ((now - pool_info.last_reward_timestamp) as u128);
```

Since `last_reward_timestamp` is already equal to now after the previous update, this calculation is redundant and ineffective.

Suggestion:

It is recommended to remove this redundant update.

MAS-3 Unused Private Function

Severity: Minor

Status: Fixed

Code Location:

sources/masterchef.move#595-610

Descriptions:

The `harvest_pending` function is intended to claim pending rewards, but it is a private function and is not called anywhere in the code.

```
fun harvest_pending(pool_info_object: Object<PoolInfo>, user_amount: u128,
user_reward_debt: u128, acc_per_share: u128): FungibleAsset acquires PoolInfo,
ObjectRef {
    let pending = calc_pending_rewards(
        user_amount,
        user_reward_debt,
        acc_per_share
    );

    let signer =
&object::generate_signer_for_extending(&ObjectRef[object::object_address(&pool_info_obj

    let token_metadata =
PoolInfo[object::object_address(&pool_info_object)].token_metadata;
    primary_fungible_store::withdraw(
        signer,
        token_metadata,
        pending
    )
}
```

Suggestion:

It is recommended to remove this private function.

Resolution:

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

MOV-1 No Upgrade Policies

Severity: Medium

Status: Fixed

Code Location:

Move.toml#1-5

Descriptions:

The protocol has no escalation measures, which may result in the contract not being able to be used under a single, stable, well-known account address that doesn't change.

https://aptos.dev/en/build/smart-contracts/book/package-upgrades

Suggestion:

It is recommended to add upgrade_policy = "compatible" to the Move.toml file.

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.